Pirouette: Query Efficient Single-Server PIR

Jiayi Kang
jiayi.kang@esat.kuleuven.be
COSIC, KU Leuven
Belgium

Leonard Schild
leonard.schild@esat.kuleuven.be
COSIC, KU Leuven
Belgium

Abstract

Private information retrieval (PIR) allows a client to query a public database privately and serves as a key building block for privacy-enhancing applications. Minimizing query size is particularly important in many use cases, for example, when clients operate on low-power or bandwidth-constrained devices. However, existing PIR protocols exhibit large query sizes: to query 2^{25} records, the smallest query size of 14.8 kB is reported in Respire [Burton et al., CCS'24]. Respire is based on fully homomorphic encryption (FHE), where a common approach to lower the client-to-server communication cost is transciphering. When combining the state-of-the-art transciphering [Bon et al., CHES'24] with Respire, the resulting protocol (which we refer to as T-Respire) has a 336 B query size, while incurring a 16.2x times higher server computation cost than Respire.

Our work presents the Pirouette protocol, which achieves a query size of just 36 B without transciphering. This represents a 9.3x reduction compared to T-Respire and a 420x reduction to Respire. For queries over 2^{25} records, the single-core server computation in Pirouette is only 2x slower than Respire and 8.1x faster than T-Respire, and the server computation is highly parallelizable. Furthermore, Pirouette requires no database-specific hint for clients and naturally extends to support queries over encrypted databases.

Keywords

Fully Homomorphic Encryption, Private Information Retrieval

1 Introduction

Private information retrieval (PIR) allows a client to retrieve a record from a public database without revealing to the database server which record is queried. While PIR is directly used for private database queries, it also is a key building block for various privacy-enhancing applications, such as private contact discovery [16, 37] and tracing [79], safe browsing [55], privacy-preserving genome imputation [46], and secure collision-risk assessment for satellites [57].

This work focuses on *single-server* [56] PIR protocols rather than *multi-server* [30] ones, as ensuring that multiple servers do not collude is hard in practice. For a PIR protocol to be non-trivial, the communication cost must be much smaller than the database size; otherwise, the client could simply download the entire database. The first single-server PIR scheme with polylogarithmic communication was introduced in [20], with subsequent improvements [22, 43] leveraging different algebraic structures.

More recent single-server PIR protocols [4, 5, 19, 36, 50, 59, 67, 68, 70, 75] often employ fully homomorphic encryption (FHE) schemes [17, 18, 27, 39, 40]. FHE typically incurs significant communication overhead due to the ciphertext expansion factor, defined as the ratio of ciphertext size to plaintext size. Nevertheless, minimizing *query size* is essential in many PIR applications, particularly when clients operate on low-power devices or when queries are transmitted over long distances, such as in satellite-to-ground communication. On the other hand, lowering the *response size* may not be as critical, as PIR responses are not always sent back to the client but instead can be used as inputs for further computations.

Developing FHE-based PIR protocols with small query sizes while maintaining computation costs reasonable is both crucial and challenging. Prior works [19, 67, 70] arrange the database as a hypercube for computation efficiency. Consequently, a PIR query consists of encrypted indices for all dimensions, which are then compressed into a single ciphertext. With this query packing technique, the smallest query size of 14.8 kB for $N = 2^{25}$ records is reported in Respire [19].

To further lower the query size, a common approach in FHE applications [6, 31] is transciphering [71]. In this approach, the client uses a classical symmetric scheme Π to encrypt the query indices, producing ciphertexts with an expansion factor close to one, which are then sent to the server. The server evaluates the decryption of Π homomorphically to obtain FHE encryptions of the query indices. The combination of the state-of-the-art transciphering method [9, 14] and Respire, which is referred to as T-Respire throughout the paper, gives 336B for $N = 2^{25}$ records, at the cost of 16.2x increase of running time compared to Respire.

The Pirouette protocol. This work introduces the Pirouette protocol, a query-efficient single-server PIR protocol without transciphering. While the Pirouette database uses polynomial rings in the hypercube structure as in [19, 67], the Pirouette query is a fresh learning with errors (LWE) [76] ciphertext instead of a ring learning with errors (RLWE) [64] ciphertext. This fresh LWE ciphertext consists of $(n+1)$ components of modulus q, with the first n components sampled uniformly from $\mathbb{Z}_q$ and the final component consisting of a linear combination of the n random values and the secret key, added to the encoded query index and a small random value. To reduce the query size, the client only needs to send the $(n+1)$-th component together with a PRG seed to generate the first n components pseudorandomly [25]. Furthermore, we design a novel procedure that enables the server to homomorphically extract the query indices for each dimension of the hypercube database from the LWE ciphertext. These encrypted query indices are then used to homomorphically select the requested record.

In terms of performance, for $N = 2^{25}$ records, the query size of Pirouette is only 36B, of which 32B are dedicated to the PRNG seed. The 36B query size is 9.3x smaller than T-Respire and 420x smaller than Respire. The single-core server computation in Pirouette is only 2x slower than Respire and 8.1x faster than T-Respire. The server computation in Pirouette is also highly parallelizable; the

concrete performance is shown in Section as demonstrated in Table 6.

While conventional PIR protocols only consider a public database, the Pirouette protocol can be easily extended to querying an encrypted database. For $\mathcal{N} = 2^{25}$ encrypted records, the query size remains 36B and the throughput is about 7.2x slower when compared to the unencrypted setting.

1.1 Technical overview

In Pirouette, the client does not need to download any database-specific hint; the user only needs to send the evaluation key(s) of the homomorphic scheme to the server, and this corresponds to the entire offline phase. Our protocol follows a similar hypercube database structure and record selection procedure as Respire [19]. Figure 1 illustrates the Pirouette protocol using toy parameters $\mathcal{N} = 2^5$, and the key steps are explained below.

Starting point: the Respire protocol. Respire considers a $(1 + \nu_2 + \nu_3)$-dimensional hypercube database with entries encoded as polynomials, where the first dimension has size 2^{ν_1} and the remaining dimensions have size 2. This results in a database with $\mathcal{N} = 2^{\nu_1+\nu_2+\nu_3}$ records, each indexed by a tuple $(\alpha, \beta_1, \ldots, \beta_{\nu_2+\nu_3})$ where $\alpha \in [2^{\nu_1}]$ and $\beta_i \in \{0, 1\}$ for all $i \in [1, \nu_2+\nu_3]$. Each plaintext element encodes 2^{ν_3} records.

The Respire query is an RLWE ciphertext that can be expanded into (1) RLWE encryptions of the one-hot encoding of α, consisting of 2^{ν_1} encryptions of binary values, (2) RGSW encryptions of $\beta_1, \ldots, \beta_{\nu_2}$, and (3) RGSW encryptions of $\beta_{\nu_2+1}, \ldots, \beta_{\nu_2+\nu_3}$. Using (1) and (2), the server homomorphically selects the plaintext polynomial corresponding to the requested record. The server then applies (3) to obtain the RLWE encryption of the requested record, which is further compressed using ModSwitch and RingSwitch operations.

LWE ciphertext in PIR query. In this work, transmitting an LWE ciphertext (combined with PRG compression) is the key to reducing query size. Specifically, the client sends $\mathsf{LWE}(\mathsf{idx})$, where $\mathsf{idx} \in [\mathcal{N}]$ is the query index within a size-$\mathcal{N}$ database. Since $\mathcal{N}$ is typically large (e.g. 2^{25}), the LWE ciphertext requires high precision. Our implementation uses a 25-bit plaintext modulus and 32-bit ciphertext modulus for LWE. This differs from conventional LWE-based applications [7, 28, 32] that typically considers a small plaintext modulus of 4 or 5 bits. Upon receiving the query, the server homomorphically converts it into formats compatible with the (plain or encrypted) hypercube database.

Decomposing high-precision LWE. The server first performs a homomorphic bit decomposition to the LWE query $\mathsf{LWE}(\mathsf{idx})$, and outputs $\lceil \log_2 \mathcal{N} \rceil$ LWE ciphertexts, each encrypting a single bit $\mathsf{idx}_i \in \mathbb{Z}_2$ such that $\mathsf{idx} = \sum_{i=0}^{\lceil \log_2 \mathcal{N} \rceil - 1} \mathsf{idx}_i \cdot 2^i$. Conventional bit decomposition based on programmable bootstrapping is practically constrained to a small precision of $\nu = 5$ bits and requires ν expensive BlindRotate operations. For practical PIRs, however, the precision $\lceil \log_2 \mathcal{N} \rceil \gg 5$ is much higher. Our proposed method, as detailed in Section 3.1, supports high-precision bit decomposition using only approximately $\frac{3\lceil \log_2 \mathcal{N} \rceil}{5}$ BlindRotate operations. For subsequent computation, these ciphertexts $\{\mathsf{LWE}(\mathsf{idx}_i)\}_{i \in [0, \lceil \log_2 \mathcal{N} \rceil - 1]}$ are converted to RGSW ciphertexts.

Leveraging RLWE′ ciphertexts. In Respire, database records are encoded into the plaintext space $\mathcal{R}_t$ and the one-hot encoding of the first-dimension query index consists of RLWE ciphertexts. Since the noise growth in their multiplication is linear in t, Respire is best suited for databases with small records to control the noise growth.

In contrast, Pirouette encodes database records as polynomials with modulus q, e.g. $d(X) \in \mathcal{R}_q$, and uses RLWE′ ciphertexts for the one-hot encoding of the first-dimension query index. The construction of these RLWE′ ciphertexts is detailed in Section 3.2, and multiplying a record with an RLWE′ ciphertext only gives logarithmic noise growth in q. Specifically, given a vector $\mathbf{g} = [1, B, B^2, \ldots, B^{\ell-1}] \in \mathbb{N}^\ell$ with $\ell = \lfloor \log_B(q) \rfloor + 1$ and $d(X) \in \mathcal{R}_q$, we may always obtain $\{d_i(X)\}_{0 \le i < \ell}$ such that $\sum_i d_i(X) \cdot \mathbf{g}[i] = d(X)$. More importantly it holds that $||d_i(X)||_\infty < B$ where $|| \cdot ||_\infty$ is the max-norm of the coefficient vector of a polynomial. Then, for an RLWE′ ciphertext $\mathbf{c} = [\mathbf{c}_0 \| \cdots \| \mathbf{c}_{\ell-1}]$, the product is computed as $\sum_i d_i(X) \cdot \mathbf{c}_i$ and the noise now grows proportionally to $\ell \cdot B$. Therefore, this approach allows Pirouette to use more efficient FHE parameters and support large database records. Our construction of one-hot encoded RLWE′ ciphertexts is detailed in Section 3.2.

Querying an encrypted database. The Pirouette protocol can be extended to support queries over an encrypted database. This is useful in scenarios where a data owner stores encrypted data in a public cloud, which answers queries from authenticated clients without knowing either the data or the query index. For example, in [13], organizations such as the National Institutes of Health (NIH) store encrypted genotype-phenotype data on the cloud to support queries from authenticated doctors.

In this setting, the database consists of RLWE encryptions of polynomials that encode plaintext records arranged, structured in the same hypercube format. The client still sends an LWE query $\mathsf{LWE}(\mathsf{idx})$, which is bit-decomposed by the server and converted into RGSW ciphertexts $\{\mathsf{RGSW}(\mathsf{idx}_i)\}_{i \in [0, \lceil \log_2 \mathcal{N} \rceil - 1]}$. These RGSW ciphertexts are then used as control bits in CMUXes to homomorphically select the encrypted record.

1.2 Related work

Single-server PIR with linear server computation. Early theoretical works [8, 20, 22, 43] have shown that non-trivial single-server PIR protocols can achieve polylogarithmic communication complexity, and the computation grows at least linearly to the database size $\mathcal{N}$. The effort to design practical PIR protocols has since motivated extensive research [1, 4, 5, 19, 36, 41, 49, 50, 59, 63, 67, 68, 70, 75], with most constructions leveraging fully homomorphic encryption and exploring various trade-offs between communication and computation.

Broadly speaking, these practical constructions can be classified into the following three categories.

- *schemes with a client-specific hint*, where the server stores a hint (such as evaluation keys in FHE-based PIR protocols [1, 4, 5, 19, 41, 63, 67, 70, 75]) for each client. In prior works, queries are RLWE ciphertexts of minimum tens of kilobytes [4, 19, 67, 70], which will be expanded by the server using techniques such as oblivious expansion [4] and

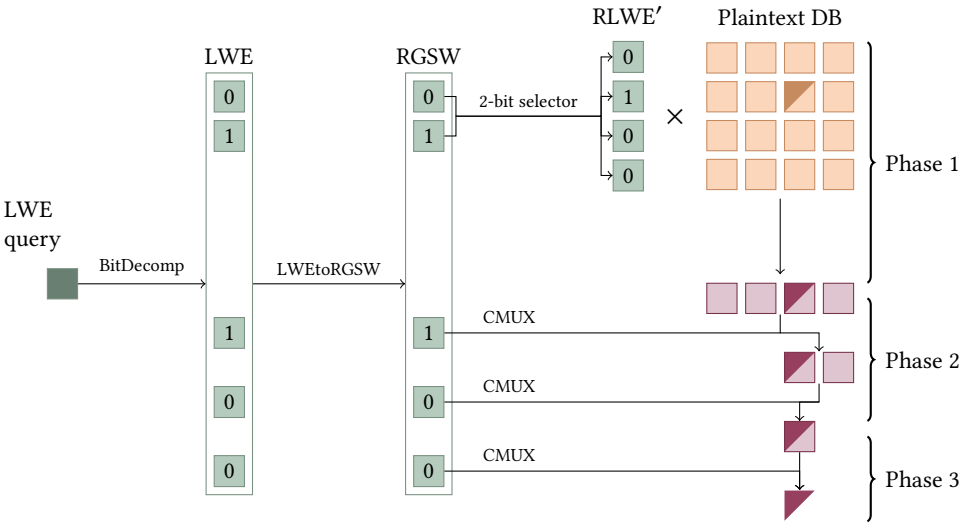

Figure 1: Processing a Pirouette query with $\nu_1 = \nu_2 = 2$ and $\nu_3 = 1$. The preparation step (Phase 0) includes bit decomposition, LWE-to-RGSW conversion, and the construction of a ν_1-bit selector composed of RLWE′ ciphertexts. Phase 1 handles the first dimension processing using the ν_1-bit selector. Phase 2 performs folding to retrieve the desired plaintext ring element using ν_2 RGSW ciphertexts. Finally, Phase 3 retrieves the corresponding database record within the subring using ν_3 RGSW ciphertexts. We refer to Section 4 for more details.

coefficient expansion algorithm [5, 23]. Our Pirouette protocol reduces the query size to tens of bytes by transmitting an LWE ciphertext.

- *schemes with a database-specific hint*, where a client needs to store a large database-dependent hint that can be hundreds of megabytes in size [36, 50]. This setup not only demands large storage from a resource-constraint client, but also requires the hint to be updated whenever the database changes. For schemes in this category, query sizes are hundreds of kilobytes, but the throughput is much faster.
- *schemes without hints*, where the server performs a preprocessing without a need to send any hint to the client. Schemes in this category [49, 59, 68] eliminate the need for offline communication, but query sizes are of orders of megabytes.

Sublinear preprocessing PIR. The linear computation bound for PIR can be bypassed by considering a preprocessing model [8]. Schemes such as [33, 34, 45, 58, 77, 81, 83] perform a client-dependent offline phase with $O(\mathcal{N})$ cost, enabling the server to answer queries in sublinear time with demonstrated concrete efficiency. On the other hand, an RLWE-based doubly-efficient PIR (DEPIR) scheme was constructed by Lin et al. [60], removing the need for client-dependent preprocessing while still providing sublinear online complexities. Since then, DEPIR has attracted growing research interest [61, 74], but no concrete efficiency has been achieved until now [73].

Querying encrypted databases. Querying encrypted databases through an encrypted index is referred to as blind array access in [7], which serves as a building block for sorting encrypted arrays. Additionally, SQL queries over encrypted databases are investigated in [11, 12, 82], and the work [13] presents a secure queryable database for storing and analyzing genotype-phenotype data.

Transciphering. Transciphering is a common approach to reduce client-to-server communication, where a client encrypts the data using some block or stream cipher Π, and the server decrypts Π homomorphically. One line of the work [35, 51, 66] focuses on designing ciphers whose decryption circuits are optimized for homomorphic evaluation. These FHE-friendly ciphers, however, are typically not standardized or used in real-world deployments. Another line of work [2, 9, 14, 78] aims to evaluate standardized ciphers such as AES efficiently in FHE and assesses their practicality.

2 Preliminaries

2.1 Notation

Given $a, b \in \mathbb{Z}$, let $[a, b]$ denote the set $\{a, a+1, \ldots, b\}$, and let $[a]$ denote $[1, a]$. For an integer q, let $\mathbb{Z}_q$ denote the ring of integers modulo q. For a power-of-two N, let $\mathcal{K} = \mathbb{Q}[X](X^N + 1)$, $\mathcal{R} = \mathbb{Z}[X](X^N + 1)$ and $\mathcal{R}_q = \mathcal{R}/(q\mathcal{R})$. We use lowercase bold such as $\mathbf{u}$ for (row) vectors and uppercase bold such as $\mathbf{U}$ for matrices. The inner product between two vectors $\mathbf{u}$ and $\mathbf{v}$ is denoted as $\langle \mathbf{u}, \mathbf{v} \rangle$.

Given a probability distribution χ, let $a \leftarrow \chi$ denote that a is sampled from χ. Given a set S, let $a \xleftarrow{\$} S$ denote that a is sampled uniformly random from S.

2.2 Private information retrieval

We recall the definition of single-server PIR with client-specific hints, where a client holds a reusable secret key sk and uploads the associated evaluation key evk to the server. The evaluation key evk

will be used to answer queries from this client. Similar to [19, 67], we allow the server to preprocess the database offline, which allows efficient query answering during the online phase.

Definition 2.1 (PIR with client-specific hints [19, 30, 67]). The single-server PIR $\Pi_{\mathsf{PIR}} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Extract})$ with client-specific hints consists of efficient algorithms with the following properties:

- $\mathsf{Setup}(1^\lambda, \{d_i\}_{i\in[\mathcal{N}]}) \to (\mathsf{sk}, \mathsf{evk}, \mathsf{db})$. Given the security parameter λ and the $\mathcal{N}$ records in the database, output the secret key sk, the public evaluation key evk, and the preprocessed database db.
- $\mathsf{Query}(\mathsf{sk}, \mathsf{idx}) \to \mathsf{qu}$. Given the secret key sk and an index $\mathsf{idx} \in [\mathcal{N}]$, output the query qu.
- $\mathsf{Answer}(\mathsf{qu}, \mathsf{evk}, \mathsf{db}) \to \mathsf{ans}$. Given the query qu, the evaluation key evk and the preprocessed database db, output the answer ans.
- $\mathsf{Extract}(\mathsf{ans}, \mathsf{sk}) \to d_{\mathsf{idx}}$. Given the answer ans and the secret key sk, output the record d_{idx}.

The following properties are essential for PIR algorithms:

- *Correctness*: If both the client and the server execute the protocol correctly, the client should recover the requested entry. Formally, a PIR protocol has correctness error δ if on any size-$\mathcal{N}$ database $\{d_i\}_{i\in[\mathcal{N}]}$, the following probability

$$\Pr\left[\; d_{\mathsf{idx}} = \hat{d}_{\mathsf{idx}} \;\middle|\; \begin{array}{c} (\mathsf{sk}, \mathsf{evk}, \mathsf{db}) \leftarrow \mathsf{Setup}(1^\lambda, \{d_i\}_{i\in[\mathcal{N}]}) \\ \mathsf{qu} \leftarrow \mathsf{Query}(\mathsf{sk}, \mathsf{idx}) \\ \mathsf{ans} \leftarrow \mathsf{Answer}(\mathsf{qu}, \mathsf{evk}, \mathsf{db}) \\ \hat{d}_{\mathsf{idx}} \leftarrow \mathsf{Extract}(\mathsf{ans}, \mathsf{sk}) \end{array} \right]$$

  is at least $1 - \delta$.
- *Security*: The server should learn nothing about the index queried by the client. Precisely, a PIR protocol is (T, ϵ)-secure if, for any adversary $\mathcal{A}$ running in time T and any two indices $\mathsf{idx}, \mathsf{idx}' \in [\mathcal{N}]$, the following holds.

$$\left|\Pr\left[\; \mathcal{A}(1^{\mathcal{N}}, \mathsf{qu}) = 1 \;\middle|\; \mathsf{qu} \leftarrow \mathsf{Query}(\mathsf{sk}, \mathsf{idx}) \;\right]\right.$$
$$\left. - \Pr\left[\; \mathcal{A}(1^{\mathcal{N}}, \mathsf{qu}) = 1 \;\middle|\; \mathsf{qu} \leftarrow \mathsf{Query}(\mathsf{sk}, \mathsf{idx}') \;\right]\right| \leq \epsilon.$$

2.3 Learning with errors and ring learning with errors

The security of our PIR schemes relies on the hardness of the learning with errors (LWE) problem [76] and the ring learning with errors (RLWE) problem [64, 65].

2.3.1 *LWE.*

The LWE problem is parametrized by a dimension n, integer modulus q, and an error distribution χ over $\mathbb{Z}$.

*Definition 2.2 (*LWE *distribution $\mathcal{D}_{\mathbf{s},\chi}$).* Given $\mathbf{s} \in \mathbb{Z}_q^n$, the LWE distribution $\mathcal{D}_{\mathbf{s},\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by choosing a uniformly random $\mathbf{a} \in \mathbb{Z}_q^n$ and error $e \leftarrow \chi$, and outputting $(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + e \bmod q) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.

We can represent $m \geq 1$ LWE instances with the same secret $\mathbf{s} \in \mathbb{Z}_q^n$ in the matrix form $(\mathbf{A}, \mathbf{b} = \mathbf{A}\cdot\mathbf{s}+\mathbf{e} \bmod q)$, where $\mathbf{A} \in \mathbb{Z}_q^{m\times n}$ and $\mathbf{b}, \mathbf{e} \in \mathbb{Z}_q^m$. Below we describe two versions of the LWE problem, both conjectured to be hard for appropriately chosen parameters. In practice, concrete security estimates of LWE instances are typically estimated using Albrecht et al.'s lattice estimator [3].

Definition 2.3 (Search LWE*).* The search LWE problem is to recover $\mathbf{s} \in \mathbb{Z}_q^n$ from m LWE instances $(\mathbf{A}, \mathbf{b} = \mathbf{A}\cdot\mathbf{s}+\mathbf{e} \bmod q)$ sampled according to the distribution $\mathcal{D}_{\mathbf{s},\chi}$.

Definition 2.4 (Decision LWE*).* The decision LWE problem is to distinguish m LWE instances $(\mathbf{A}, \mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q)$ sampled according to the distribution $\mathcal{D}_{\mathbf{s},\chi}$, from $(\mathbf{A}, \mathbf{b}) \xleftarrow{\$} \mathbb{Z}_q^n \times \mathbb{Z}_q$.

2.3.2 *RLWE.*

The RLWE problem extends the LWE problem to structured algebraic rings. This work only focuses on the ring $\mathcal{R}$, a cyclotomic ring with a power-of-two cyclotomic order.

*Definition 2.5 (*RLWE *distribution $\mathcal{D}_{\mathbf{s},\chi}$).* Given $\mathbf{s} \in \mathcal{R}_q$ and an error distribution χ over $\mathcal{R}_q$, the RLWE distribution $\mathcal{D}_{\mathbf{s},\chi}$ over $\mathcal{R}_q^2$ is sampled by choosing a uniformly random $\mathbf{a} \in \mathcal{R}_q$ and an error $\mathbf{e} \leftarrow \chi$, and outputting $(\mathbf{a}, \mathbf{b} = \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \bmod q) \in \mathcal{R}_q^2$.

Then we describe the two versions of the RLWE problem. To date, no known attack on RLWE problems exploits its additional ring structure compared to LWE. Therefore, the security of an RLWE instance is estimated by transforming it to a corresponding LWE instance with the same dimension, modulus, and an appropriate error distribution.

Definition 2.6 (Search RLWE*).* The search RLWE problem is to recover $\mathbf{s} \in \mathcal{R}_q$ from m RLWE instances $\{(\mathbf{a}_i, \mathbf{b}_i = \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \bmod q)\}_{i\in[m]}$ sampled according to the RLWE distribution $\mathcal{D}_{\mathbf{s},\chi}$.

Definition 2.7 (Decision RLWE*).* The decision RLWE problem is to distinguish m RLWE instances $\{(\mathbf{a}_i, \mathbf{b}_i = \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \bmod q)\}_{i\in[m]}$ sampled according to the RLWE distribution $\mathcal{D}_{\mathbf{s},\chi}$, from m uniformly sampled pairs $\left\{(\mathbf{a}_i, \mathbf{b}_i) \xleftarrow{\$} \mathcal{R}_q^2\right\}_{i\in[m]}$.

Compared to LWE, RLWE provides better efficiency and amortized storage. The efficiency comes from operating over ring elements in $\mathcal{R}$, enabling faster multiplications via the fast Fourier transform (FFT). In terms of storage, a single RLWE sample contains the same information as N LWE samples, but it only requires $2N \log q$ bits instead of $(N^2 + N) \log q$ bits. Therefore, Phase 1-3 of the Pirouette protocol operates over $\mathcal{R}$ and relies on the hardness of RLWE for security.

However, when storing a single element, an LWE sample is smaller than an RLWE sample. Thus, the Pirouette protocol achieves a low query size by transmitting an LWE sample, relying on the hardness of LWE in this phase.

2.4 FHE primitives

The Pirouette protocol adopts a cross-scheme approach, leveraging all of the BFV [17, 40], GSW [44], FHEW [39] and TFHE [27] homomorphic encryption schemes, along with state-of-the-art optimizations [24, 29, 31, 42, 47, 48, 62, 69, 80]. This subsection outlines the relevant FHE ciphertext types, basic operations and subroutines.

2.4.1 *FHE ciphertexts.*

Previous PIR works such as [36, 50] use LWE ciphertexts, and [19, 67, 70] use RLWE ciphertexts and the ring variant of the Gentry-Sahai-Waters (GSW) [44] (RGSW) ciphertexts. In contrast, this work incorporates all three types of ciphertexts, additionally utilizing RLWE′ ciphertexts to minimize noise growth.

Let t denote a plaintext modulus, $q \gg t$ denote a ciphertext modulus, and $\Delta = \lfloor q/t \rceil$ denote a scaling factor. Given a base B_q, let $\mathbf{g} = [1, B_q, B_q^2, \ldots, B_q^{\ell-1}] \in \mathbb{Z}_q^\ell$ denote the gadget vector of length $\ell = \lfloor \log_B(q) \rfloor + 1$, and $\mathbf{G} = \mathrm{diag}(\mathbf{g}^\intercal, \mathbf{g}^\intercal) \in \mathbb{Z}_q^{2\ell \times 2}$ denote the gadget matrix.

- $\mathsf{LWE}_{\mathbf{s}}^{n,q}(m) = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$ denotes an LWE ciphertext that satisfies
$$b = \langle \mathbf{a}, \mathbf{s} \rangle + m + e \bmod q,$$
where $\mathbf{s} \in \mathbb{Z}^n$ is the secret key, $m \in \mathbb{Z}_q$ is (the encoding of) a message, and $e \in \mathbb{Z}$ is an error in a pre-determined error distribution. If m encodes a message $\overline{m} \in \mathbb{Z}_t$ as $m = \Delta \cdot \overline{m}$, the ciphertext can also be denoted as $\mathsf{LWE}_{\mathbf{s}}^{n,q}(\overline{m}, \Delta)$.
- $\mathsf{RLWE}_s^{N,q}(m) = (a, b) \in \mathcal{R}_q^2$ denotes an RLWE ciphertext that satisfies
$$b = a \cdot s + m + e \bmod q,$$
where $s \in \mathcal{R}$ is the secret key, $m \in \mathcal{R}_q$ is (the encoding of) a message, and $e \in \mathcal{R}$ is an error in a pre-determined error distribution. If m encodes a message $m \in \mathcal{R}_t$ as $m = \Delta \cdot \overline{m}$, the ciphertext can also be denoted as $\mathsf{RLWE}_s^{N,q}(\overline{m}, \Delta)$.
- $\mathsf{RLWE}_s'^{N,q}(m) = (\mathbf{a}^\intercal, \mathbf{b}^\intercal) \in \mathcal{R}_q^{\ell \times 2}$ denotes an RLWE′ ciphertext that satisfies
$$\mathbf{b}^\intercal = \mathbf{a}^\intercal \cdot s + m \cdot \mathbf{g}^\intercal + \mathbf{e}^\intercal \bmod q$$
where $s \in \mathcal{R}$ is the secret key, $m \in \mathcal{R}_q$ is (the encoding of) a message, and $e \in \mathcal{R}$ is an error in a pre-determined error distribution. In other words,
$$\mathsf{RLWE}_s'^{N,q}(m) = \Big[\mathsf{RLWE}_s^{N,q}(m) \| \mathsf{RLWE}_s^{N,q}(B_q m) \| \cdots \| \mathsf{RLWE}_s^{N,q}(B_q^{\ell-1} m) \Big].$$
- $\mathsf{RGSW}_s^{N,q}(m) = (\mathbf{a}^\intercal, \mathbf{b}^\intercal) \in \mathcal{R}_q^{2\ell \times 2}$ denotes an RGSW ciphertext that satisfies
$$\mathbf{b}^\intercal = \mathbf{a}^\intercal \cdot s + m \cdot \mathbf{G} \cdot \begin{bmatrix} -s \\ 1 \end{bmatrix} + \mathbf{e}^\intercal \bmod q$$
where $m \in \{0, \pm X^v : v \in [0, N-1]\}$ is a message encoded under the secret key $s \in \mathcal{R}$, and $\mathbf{e} \in \mathcal{R}_q^{2\ell}$ is an error term in a pre-determined error distribution. In other words, $\mathsf{RGSW}_s^{N,q}(m) = \Big[\mathsf{RLWE}_s'^{N,q}(-s \cdot m) \| \mathsf{RLWE}_s'^{N,q}(m) \Big]$.

2.4.2 FHE basic operations. Fully homomorphic encryption allows computation over ciphertexts. The basic homomorphic operations used in this work are as follows.

- Add denotes the homomorphic addition between (i) two ciphertexts of the same type (e.g. LWE or RLWE) encrypted under the same secret key, or (ii) one ciphertext and one plaintext.
- Mult denotes the homomorphic multiplication between (i) two ciphertexts encrypted under the same secret key, such as two RLWE ciphertexts, one RGSW ciphertext and one RLWE/RLWE′/RGSW ciphertext, or (ii) one ciphertext and one plaintext.
- Aut_{σ_i} denotes the homomorphic automorphism that maps $\mathsf{RLWE}(m)$ to $\mathsf{RLWE}(\sigma_i(m))$, where $\sigma_i : m(X) \to m(X^i)$ denotes an automorphism in the Galois group $\mathrm{Gal}(\mathcal{K}/\mathbb{Q})$.
- $\mathsf{ModSwitch}_{q \to q'}$ denotes the modulus switching operation (i) from $\mathsf{LWE}_{\mathbf{s}}^{n,q}(m)$ to $\mathsf{LWE}_{\mathbf{s}}^{n,q'}(m)$ for $m \in \mathbb{Z}_q$, or (ii) from $\mathsf{RLWE}_s^{N,q}(m)$ to $\mathsf{RLWE}_s^{N,q'}(m)$ for $m \in \mathcal{R}_q$.
- $\mathsf{BlindRotate}(\mathsf{LWE}(m), \mathsf{bsk}, \mathsf{acc})$ denotes the blind rotation operation, which takes $\mathsf{ct} = (\mathbf{a}, b) = \mathsf{LWE}_{\mathbf{s}}^{n,q}(m)$, the bootstrapping key bsk, and the accumulator $\mathsf{acc} = \mathsf{RLWE}(T(X))$ as inputs. The typical LWE modulus q is N or $2N$, and let $\varphi(\mathsf{ct}) \coloneqq b - \langle \mathbf{a}, \mathbf{s} \rangle$. Then the output is
$$\mathsf{RLWE}\left(T(X) \cdot X^{\varphi(\mathsf{ct}) \bmod 2N}\right)$$
with a constant noise level independent of ct. For brevity, the explicit reference to input bsk can be omitted.
- $\mathsf{SampleExtract}(\mathsf{RLWE}(m), k) \to \mathsf{LWE}(m_k)$ denotes the sample extraction that takes $\mathsf{RLWE}_s^{N,Q}(m)$ and an index $k \in [0, N-1]$ as inputs, and outputs $\mathsf{LWE}_{\overrightarrow{s}}^{N,Q}(m_k)$ where m_k is the k-th coefficient of m and $\overrightarrow{s}$ is the coefficient vector of the RLWE secret s.

2.4.3 FHE subroutines. This work uses the following subroutines built from basic operations in FHE.

- $\mathsf{CMUX}(\mathsf{RGSW}(b), \mathsf{RLWE}(m_0), \mathsf{RLWE}(m_1)) \to \mathsf{RLWE}(m_b)$ denotes a homomorphic CMUX gate. Given an $\mathsf{RGSW}_s^{N,q}$ encryption of a control bit $b \in \{0, 1\}$ and $\mathsf{RLWE}_s^{N,q}$ encryptions of $m_0, m_1 \in \mathcal{R}_q$, CMUX returns an $\mathsf{RLWE}_s^{N,q}$ encryption of
$$m_b = m_0 + b(m_1 - m_0) \in \mathcal{R}_q.$$
- $\mathsf{RingSwitch}_{N \to N_1}$ denotes the ring switching operation from $\mathsf{RLWE}_s^{N,q}(m)$ to $\mathsf{RLWE}_{s_1}^{N_1,q}(\kappa(m))$ for $m \in \mathcal{R}_q$ and $N_1 \mid N$, where
$$\kappa : \mathcal{R}_{N,q} \to \mathcal{R}_{N_1,q}$$
$$\sum_{i=0}^{N-1} f_i X^i \mapsto \sum_{i=0}^{N_1-1} f_{i \cdot N/N_1} X^i.$$
The ring switching procedure is essentially a homomorphic trace operation composed of automorphisms, as explained in [19, 42].
- $\mathsf{LWEtoRGSW}(\mathsf{LWE}(b)) \to \mathsf{RGSW}(b)$ converts an $\mathsf{LWE}_{\mathbf{s}}^{n,q}$ encryption of a bit $b \in \{0, 1\}$ into an RGSW ciphertext $\mathsf{RGSW}_s^{N,Q}(b)$. This operation is also known as circuit bootstrapping [27], and the state-of-the-art construction [80] includes basic operations such as blind rotations and homomorphic automorphisms.

3 Building Blocks

3.1 High-precision homomorphic bit decomposition

In homomorphic bit decomposition, an LWE ciphertext encoding $m \in \mathbb{Z}_{2^k}$ is converted into k LWE ciphertexts, each encoding a single bit $m_i \in \mathbb{Z}_2$ such that $m = \sum_{i=0}^{k-1} m_i \cdot 2^i$. Conventional homomorphic bit decomposition using the programmable bootstrapping [25] is practically limited to a small precision of ν bits (typically 4 or 5), requiring ν expensive BlindRotate operations

to produce ν bit-decomposed ciphertexts. To address this, we instantiate the multi-value bootstrapping [21], reducing the number of BlindRotate operations to just one while maintaining minimal noise growth. Specifically, we consider input LWE ciphertexts with modulus $q = 2N$ and construct lookup tables similar to [78]. The resulting ν-bit decomposition is presented in Algorithm 1 and serves as the base algorithm of our high-precision bit decomposition.

Algorithm 1 ν-bit decomposition for LWE with modulus $q = 2N$

Input: Ciphertext $\mathrm{ct} = \mathsf{LWE}_{\mathbf{s}}^{n,q=2N}(m, N/2^{\nu-1})$, where $m = \sum_{i=0}^{\nu-1} m_i \cdot 2^i$

Output: Ciphertexts $\{\mathsf{LWE}_{\mathbf{S}}^{N,q=Q}(m_i, Q/2)\}_{i\in[0,\nu-1]}$

function BasicBitDecomp(ct, ν)
  $\mathrm{acc} \leftarrow \mathsf{RLWE}_{S}^{N,Q}(\frac{Q}{4})$
  $\mathrm{ct}_{BR} \leftarrow \mathsf{BlindRotate}(\mathrm{ct}, \mathrm{acc})$ ▷ $\mathrm{ct}_{BR} = \mathsf{RLWE}(\frac{Q}{4} \cdot (-1)^{m_{\nu-1}} \cdot X^{\varphi(\mathrm{ct}) \bmod N})$
  $p^{(\nu-1)} \leftarrow \sum_{j=0}^{N-1} X^j$
  $\mathrm{ct}^{(\nu-1)} \leftarrow \mathsf{Add}\left(\mathsf{Mult}\left(p^{(\nu-1)}, \mathrm{ct}_{BR}\right), \frac{Q}{4}\right)$
  $\mathrm{ct}_{out}^{(\nu-1)} \leftarrow \mathsf{SampleExtract}(\mathrm{ct}^{(\nu-1)}, 0)$
  $\mathrm{ct}'_{BR} \leftarrow 2 \cdot \mathrm{ct}_{BR}$
  for $i \leftarrow 0$ to $\nu - 2$ **do**
    $p^{(i)} \leftarrow \sum_{j=1}^{N-1} \mathsf{LSB}_{i+1}(N-j)X^j$
    $\mathrm{ct}^{(i)} \leftarrow \mathsf{Mult}\left(p^{(i)}, \mathrm{ct}'_{BR}\right)$
    $\mathrm{ct}_{out}^{(i)} \leftarrow \mathsf{SampleExtract}(\mathrm{ct}^{(i)}, 0)$
  end for
  return $\{\mathrm{ct}_{out}^{(i)}\}_{i\in[0,\nu-1]}$
end function

To further extend the decomposition to $k = d \cdot \nu$ bits, we apply the digit decomposition method from [62] with a ν-bit base, first breaking a k-bit ciphertext into d ciphertexts of ν bits. For the sake of completeness, we recall the signature of [62, Algorithm 4] in Algorithm 2. This step requires $2 \cdot d$ BlindRotate operations, and the resulting intermediate ciphertexts are used as inputs to our base bit decomposition algorithm, enabling efficient high-precision bit decomposition. The complete algorithm is presented in Algorithm 3.

Notably, our algorithm is more efficient than directly applying the digit decomposition method in [62] with a one-bit base. For bit decomposition of $k = d \cdot \nu$ bits, our method requires $3 \cdot d \approx \frac{3k}{5}$ BlindRotate operations while the one-bit base method requires $2k$ BlindRotate operations.

Algorithm 2 ν-bit digit decomposition

Input: Ciphertext $\mathrm{ct} = \mathsf{LWE}_{\mathbf{s}}^{n,q}(m, q/2^{d\cdot\nu})$, where $m = \sum_{i=0}^{d-1} m_i \cdot 2^{d\cdot i}$ and $q = 2^k$ for some $k \in \mathbb{Z}$

Output: Ciphertexts $\{\mathsf{LWE}_{\mathbf{S}}^{n,Q=2N}(m_i, Q/2^{\nu})\}_{i\in[0,d-1]}$

3.2 Construction of ν-bit selectors

A homomorphic selector selects messages based on encrypted control bits. Given two messages m_0, m_1 and an encrypted control bit

Algorithm 3 $(d \cdot \nu)$-bit homomorphic bit decomposition

Input: Ciphertext $\mathrm{ct} = \mathsf{LWE}_{\mathbf{s}}^{n,q}(m, q/2^{d\cdot\nu})$, where $m = \sum_{i=0}^{d\cdot\nu-1} m_i \cdot 2^i$ and $q/2^{(d-1)\cdot\nu} = 2N$

Output: Ciphertexts $\{\mathsf{LWE}_{\mathbf{S}}^{N,Q}(m_i, Q/2)\}_{i\in[0,d\cdot\nu-1]}$

function BitDecomp(ct, d, ν)
  $\{\mathrm{ct}_k\}_{k\in[d]} \leftarrow$ DigitDecomp$(q, 2N, \mathrm{ct})$ ▷ [62, Algorithm 4]
  return $\{$BasicBitDecomp$(\mathrm{ct}_k, \nu)\}_{k\in[d]}$
end function

$\mathbf{Enc}(b)$, it homomorphically computes

$$\mathbf{Enc}(m_b) = m_0 + (m_1 - m_0) \cdot \mathbf{Enc}(b). \tag{1}$$

This concept extends to a *ν-bit selector*, which operates on 2^ν messages $\{m_i\}_{i\in[0,2^\nu-1]}$ and ν encrypted control bits $\{\widetilde{\mathbf{Enc}}(b_j)\}_{j\in[0,\nu-1]}$. The selector outputs an encryption of m_b where the index $b = \sum_j b_j 2^j$. This requires generating 2^ν ciphertexts $\{\mathbf{Enc}(\delta_{i,b})\}_{i\in[0,2^\nu-1]}$ from the encrypted control bits and homomorphically computing

$$\mathbf{Enc}(m_b) = m_0 + \sum_{i=1}^{2^\nu-1} (m_i - m_0) \cdot \mathbf{Enc}(\delta_{i,b}). \tag{2}$$

In the Pirouette instantiation of the selector, messages m_i are elements in $\mathcal{R}_q$ and $\mathbf{Enc}(\cdot)$ is the RLWE′ encryption. This ensures the noise growth during the computation of (1) and (2) remains low, increasing only logarithmic in q. As a remark, the ν-bit selector in (2) is conceptually equivalent to the 2^ν-ary CMUX gate in [52, 53], except that the CMUX gate uses RGSW ciphertexts to select RLWE ciphertexts and our selector uses RLWE′ ciphertexts to select unencrypted elements in $\mathcal{R}_q$.

Furthermore, Pirouette instantiates $\widetilde{\mathbf{Enc}}()$ as RGSW encryptions. The naive way to generate 2^ν ciphertexts $\{\mathsf{RLWE}'(\delta_{i,b})\}_{i\in[0,2^\nu-1]}$ from control bits $\{\mathsf{RGSW}(b_j)\}_{j\in[0,\nu-1]}$ is to compute

$$\prod_{j=0}^{\nu-1} \left(\mathsf{RGSW}(b_j) - i_j\right) \cdot \mathsf{RLWE}'(1), \quad \text{for all } i \in [0, 2^\nu - 1],$$

where i_j is the bit decomposition of i such that $i = \sum_j i_j 2^j$. However, this method requires $2^\nu \cdot \nu$ RGSW-RLWE′ multiplications, making it computationally expensive. To optimize this process, we reuse intermediate results for different i by constructing a binary tree, where the leaves are desired ciphertexts $\{\mathsf{RLWE}'(\delta_{i,b})\}_{i\in[0,2^\nu-1]}$ and the inner nodes store intermediate results. As such, our optimization instantiates the homomorphic traversal algorithm in [31] with RLWE′ ciphertexts.

Our tree starts from the root node, $\mathsf{RLWE}'(1)$, and follows a branching process. Each branching step takes a parent node $\mathsf{RLWE}'(n)$ and a control bit $\mathsf{RGSW}(b_i)$ as inputs, and outputs the following left child LC and right child RC

$$\begin{aligned} \mathsf{LC}(n, b_i) &= \mathsf{Mult}(\mathsf{RGSW}(b_i), \mathsf{RLWE}'(n)) = \mathsf{RLWE}'(b_i \cdot n) \\ \mathsf{RC}(n, b_i) &= n - \mathsf{LC}(n, b_i). \end{aligned} \tag{3}$$

The resulting procedure requires only $(2^\nu - 1)$ RGSW-RLWE′ multiplications – just $1/\nu$ of the naive approach. We further present an example in Figure 2 for illustration purposes.

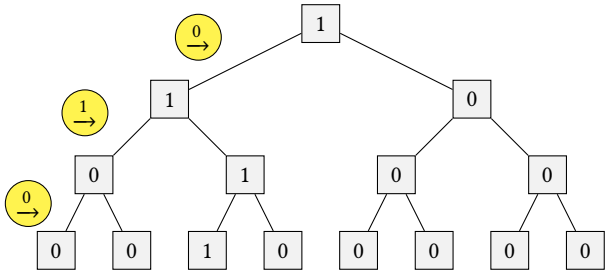

Figure 2: The construction of $\{\mathsf{RLWE}'(\delta_{i,b})\}_{i\in[0,2^3-1]}$ from $\{\mathsf{RGSW}(b_i)\}_{i\in[0,2]}$ for $b_2=0, b_1=1, b_0=0$. RLWE′ ciphertexts are represented as grey squares, while RGSW ciphertexts are represented as yellow circles. The root node is $\mathsf{RLWE}'(1)$; given the controller bit $\mathsf{RGSW}(b_2=0)$, the left child $\mathsf{LC}(1,0)$ and right child $\mathsf{RC}(1,0)$ are derived following the branching algorithm (3). The process continues recursively, where each node is generated from its parent node and the corresponding control bit at that level.

4 Protocol

This section presents the Pirouette protocol and its extension to querying encrypted databases.

4.1 The Pirouette protocol

At a high level, the Pirouette protocol builds upon the record-retrieval framework established in [19, 41, 67, 70], particularly utilizing the database structure and response compression technique from [19]. It consists of three phases: first-dimension processing, folding and rotation, each operating on independent RLWE and RGSW ciphertexts.

Unlike prior works [19, 67, 70] that compress these RLWE and RGSW ciphertexts into a query, Pirouette simplifies the query process so that the querier only needs to send the LWE ciphertext of the queried index. The server then homomorphically generates the necessary input ciphertexts for different phases while keeping noise growth minimal. Additionally, our approach removes the small database record limitation in [19] by efficiently generating and using RLWE′ ciphertexts for the first-dimension processing phase.

Below we present the Pirouette protocol.

Setup. For a given security parameter λ, the cryptographic and PIR-related parameters are defined as shown in Table 1. Each plaintext element in $\mathcal{R}_{N,p}$ encodes 2^{ν_3} records, so a size-$\mathcal{N}$ database is represented as $\frac{\mathcal{N}}{2^{\nu_3}} := 2^{\nu_1+\nu_2}$ elements in the ring $\mathcal{R}_{N,p}$. In addition to preprocessing the database, the setup algorithm samples the collection of secret keys $\mathsf{sk} = \{\mathbf{s}\in\mathbb{Z}^n, s\in\mathcal{R}_N, s_1\in\mathcal{R}_{N_1}\}$ and generates evaluation keys evk required for the key-switching operations. Under the standard circular security assumption, the evaluation key does not leak information about the secret key.

Query. Using the LWE secret key $\mathbf{s}\in\mathbb{Z}^n$, the querier generates

$$\mathsf{qu} = \mathsf{LWE}^{n,q}_{\mathbf{s}}(\mathsf{idx})$$

for the desired index $\mathsf{idx}\in[\mathcal{N}]$.

Answer. Using the evaluation keys evk, the server homomorphically expands the query qu, referred to as Phase 0. Specifically, the server computes $\{\tilde{\mathsf{ct}}_k\}_{k\in[0,\log(\mathcal{N})-1]} \leftarrow \mathtt{BitDecomp}(\mathsf{qu})$ using Algorithm 3, with appropriate parameters for the `BasicBitDecomp` subroutine. Then the server performs

$$\mathsf{ct}_k \leftarrow \mathsf{LWEtoRGSW}(\tilde{\mathsf{ct}}_k), \forall k\in[0,\log(\mathcal{N})-1]$$

using the conversion method in [80] along with necessary modulus switching operations. Furthermore, we use the first ν_1 ciphertexts $\{\mathsf{ct}_k\}_{k\in[0,\nu_1-1]}$ to build a ν_1-bit selector $\{\mathsf{ct}'_i\}_{i\in[0,2^{\nu_1}-1]}$ that are RLWE′ ciphertexts.

Let $\mathsf{db} = \{\mathsf{db}_i\in\mathcal{R}_{N,p}\}_{i\in[0,2^{\nu_1+\nu_2}-1]}$ denote the preprocessed database, where we define $\vec{\mathsf{db}}_i := [\mathsf{db}_{i\cdot 2^{\nu_2}}, \mathsf{db}_{i\cdot 2^{\nu_2}+1}, \ldots, \mathsf{db}_{i\cdot 2^{\nu_2}+2^{\nu_2}-1}]$. As in [19], the server then proceeds with the following computations using db and the output from Phase 0:

(1) First dimension: compute

$$[\mathsf{ct}^{(1)}_0, \ldots, \mathsf{ct}^{(1)}_{2^{\nu_2}-1}] \leftarrow \sum_{i=0}^{2^{\nu_1}-1} \mathsf{Mult}(\vec{\mathsf{db}}_i, \mathsf{ct}'_i),$$

where $\mathsf{Mult}(\vec{\mathsf{db}}_i, \mathsf{ct}'_i)$ is a 2^{ν_2}-array of RLWE ciphertexts, and the j-th RLWE ciphertext is derived from the multiplication between the plaintext $\vec{\mathsf{db}}_i[j]$ and the RLWE′ ciphertext ct'_i.

(2) Folding: Let $\mathsf{ct}^{(2)}_{0,j} = \mathsf{ct}^{(1)}_j, \forall j\in[0,2^{\nu_2}-1]$. Then for each $r\in[\nu_2]$ and $j\in[0,2^{\nu_2-r}]$, compute

$$\mathsf{ct}^{(2)}_{r,j} \leftarrow \mathsf{CMUX}\left(\mathsf{ct}_{\nu_1+r},\ \mathsf{ct}^{(2)}_{r-1,j},\ \mathsf{ct}^{(2)}_{r-1,j+2^{\nu_2-r}}\right).$$

(3) Rotation: Let $\mathsf{ct}^{(3)}_0 = \mathsf{ct}^{(2)}_{\nu_2,0}$. Then for each $r\in[\nu_3]$, compute

$$\mathsf{ct}^{(3)}_r \leftarrow \mathsf{CMUX}\left(\mathsf{ct}_{\nu_1+\nu_2+r},\ \mathsf{ct}^{(3)}_{r-1},\ X^{-2^{\nu_3-r}}\cdot\mathsf{ct}^{(3)}_{r-1}\right).$$

The final output $\mathsf{ct}^{(3)}_{\nu_3}\in\mathcal{R}^2_{N,Q}$ is an RLWE ciphertext, and the server further performs $\mathsf{ModSwitch}_{Q\to Q_1}$ and $\mathsf{RingSwitch}_{N\to N_1}$ to obtain $\mathsf{ans}\in\mathcal{R}^2_{N_1,Q_1}$.

Extract. Using the RLWE secret key $s_1\in\mathcal{R}_{N_1}$, the querier decrypts the received ciphertext ans to retrieve the desired record in $\mathcal{R}_{N_1,p}$.

Table 1: Setup parameters in Pirouette

Parameter	Meaning
n	LWE dimension
q	LWE modulus for query
N	RLWE dimension during computation
Q	RLWE modulus during computation
N_1	RLWE dimension for response
Q_1	RLWE modulus for response
$\mathcal{N}$	total number of records in the database
$\mathcal{R}_{N_1,p}$	a single database record
$\mathcal{R}_{N,p}$	plaintext ring element that packs $\frac{N}{N_1}$ records
ν_1	bit-length of the first dimension
ν_2	folding dimension, satisfying $\nu_1+\nu_2=\log(\mathcal{N}\cdot\frac{N_1}{N})$
ν_3	rotation dimension $\log\frac{N}{N_1}$

Furthermore, we describe the correctness and security of the Pirouette protocol.

Correctness. Recall that (R)LWE samples include noise components, whose variance may grow when combining or transforming them. Hence, the overall correctness of any algorithm that outputs a (R)LWE ciphertext $\mathbf{c}$ will be characterized on the one hand, by the soundness of the approach and on the other hand by a failure probability ϵ. This probability describes the likelihood that, after decrypting and decoding $\mathbf{c}$, we obtain a result that is different from the expected message. Hence, once soundness is established, overall correctness will depend on specific parameter choices, except for pathological cases.

As Pirouette leverages a set of well-established algorithms, its correctness hinges on the correctness of the individual components, sometimes referred to *atomic patterns* [10]. The formulae giving the failure probabilities of the building blocks used can be found in their respective works. Specifically: [62] for `DigitDecomp`, [78] for the analysis of `BasicBitDecomp`, [80] for LWEtoRGSW, [19] for Respire and e.g. [25, 38] for the correctness of general FHE primitives such as blind-rotation. During the evaluation, we determine a set of parameters such that the overall failure probability is sufficiently low. In practice, a common choice of ϵ is $\epsilon \leq 2^{-40}$.

Security. In line with previous works [1, 4, 5, 19, 36, 41, 49, 50, 59, 63, 67, 68, 70, 75] on PIR, our threat model considers an honest-but-curious server, which follows the protocol correctly but tries to deduce information from clients' inputs. To reach the security goal, the client encrypts its query before sending it to the server, which performs homomorphic computations on ciphertexts. As such, the server has access to the LWE query, RLWE ciphertexts generated during the evaluation, and public evaluation keys evk. The client's security is guaranteed by the hardness of LWE and RLWE problems along with the circular security assumption, commonly adopted in most FHE-based applications including the previous PIR schemes.

4.2 Extending Pirouette to encrypted databases

Pirouette can be extended to support private queries over encrypted databases, where a data owner outsources encrypted data to a public cloud server. In this setting, the cloud can respond to queries from authenticated clients without learning the underlying data or the query index.

Precisely, the plaintext database structure is the same as Pirouette: a size-$\mathcal{N}$ database is represented as $\frac{\mathcal{N}}{2^{\nu_3}} \coloneqq 2^{\nu_1+\nu_2}$ elements in the ring $\mathcal{R}_{N,p}$. The server however stores the RLWE encryptions of these polynomials, ensuring data confidentiality from the server. The Query and Extract procedures are identical to Pirouette; thus, we only detail the Answer procedure performed below.

Answer. Using the evaluation keys evk, the server homomorphically expands the query qu, referred to as Phase 0. Specifically, the server computes $\{\tilde{\mathsf{ct}}_k\}_{k\in[0,\log(\mathcal{N})-1]} \leftarrow \mathtt{BitDecomp}(\mathsf{qu})$ using Algorithm 3, with appropriate parameters for the `BasicBitDecomp` subroutine. Then the server performs

$$\mathsf{ct}_k \leftarrow \mathsf{LWEtoRGSW}(\tilde{\mathsf{ct}}_k), \forall k \in [0, \log(\mathcal{N}) - 1]$$

using the conversion method in [80] along with necessary modulus switching operations.

Let $\mathsf{db} = \{\mathsf{db}_i \in \mathcal{R}_{N,p}\}_{i\in[0,2^{\nu_1+\nu_2}-1]}$ denote the preprocessed database, and $\mathsf{ct} = \{\mathsf{ct}_i = \mathsf{RLWE}(\mathsf{db}_i)\}_{i\in[0,2^{\nu_1+\nu_2}-1]}$ denoted the encrypted database stored in the server, which performs the following computation using ct and the output from Phase 0:

(1) Folding: Let $\mathsf{ct}^{(1)}_{0,j} = \mathsf{ct}_j, \forall j \in [0, 2^{\nu_1+\nu_2} - 1]$. Then for each $r \in [\nu_1 + \nu_2]$ and $j \in [0, 2^{\nu_1+\nu_2-r}]$, compute

$$\mathsf{ct}^{(1)}_{r,j} \leftarrow \mathsf{CMUX}\left(\mathsf{ct}_{\nu_1+\nu_2+r},\ \mathsf{ct}^{(1)}_{r-1,j},\ \mathsf{ct}^{(1)}_{r-1,j+2^{\nu_1+\nu_2-r}}\right).$$

(2) Rotation: Let $\mathsf{ct}^{(2)}_0 = \mathsf{ct}^{(1)}_{\nu_2,0}$. Then for each $r \in [\nu_3]$, compute

$$\mathsf{ct}^{(2)}_r \leftarrow \mathsf{CMUX}\left(\mathsf{ct}_{\nu_1+\nu_2+r},\ \mathsf{ct}^{(2)}_{r-1},\ X^{-2^{\nu_3-r}} \cdot \mathsf{ct}^{(2)}_{r-1}\right).$$

The final output $\mathsf{ct}^{(2)}_{\nu_3} \in \mathcal{R}^2_{N,Q}$ is an RLWE ciphertext, and the server further performs $\mathsf{ModSwitch}_{Q\to Q_1}$ and $\mathsf{RingSwitch}_{N\to N_1}$ to obtain $\mathsf{ans} \in \mathcal{R}^2_{N_1,Q_1}$.

5 Implementation and Evaluation

In this section, we evaluate Pirouette and compare it to Respire and Respire in combination with a transciphering approach (T-Respire). Our code is available as an uploaded artifact under Additional Materials with a README file that details installation and benchmarking instructions.

5.1 Parameter selection and experimental setup

We implement our approach by relying on the OpenFHE library and manually optimized routines in time-critical sections. We run all experiments using an Intel(R) Xeon(R) Gold 6248R CPU with 512 GB of RAM, and give results for both the sequential and parallel setting over 16 cores.

We optimize parameters separately for each sub-procedure used by Pirouette, as recommended in [10], in order to provide a failure probability of at most 2^{-40} and a standard security parameter of $\lambda = 128$ bits. The security of our parameters was estimated through Albrecht et al.'s lattice estimator [3]. We note that we rely on binary keys i.e. the coefficients are sampled uniformly from $\{0, 1\}$.

Table 2 gives the relevant parameters for the digit decomposition and subsequent bit decomposition. In Table 2, we specify several values of the LWE dimension n. Indeed, the bit decomposition may output LWE samples with a different dimension and modulus than the input, through the use of modulus- and key-switching. We exploit this fact to improve the performance of the subsequent LWE to RGSW conversion, as a smaller LWE dimension n has a direct impact on the computational complexity of the BlindRotate procedure which is linear in n (see [26]).

Next, Table 3 describes the parameters employed for the scheme switching step. The values $B_{\mathsf{RGSW}}, \ell_{\mathsf{RGSW}})$ denote the gadget basis and gadget length for the resulting RGSW samples. Furthermore, we note that $\ell_{\mathsf{RGSW}} \neq \lceil \log_{B_{\mathsf{RGSW}}}(Q) \rceil + 1$. This is due to the fact that we rely on an *approximate* gadget [25, 80] to improve performance. Finally, we construct our ν_1-bit selectors with a basis of $B = 2^4$ and $\ell = 2$ digits, and give the output parameters in Table 4 together with the values of ν_i.

Table 2: Parameters employed for bit-decomposition

Parameter	n_{in}	n_{out}	N	$\log_2(q_{in})$	$\log_2(q_{out})$	$\log_2(Q)$	$\log_2(Q_{ksk})$	B	B_{ksk}	σ^2
Value	1300	600	2^{11}	32	12	56	42	2^{14}	2^3	3.19^2

Table 3: Parameters employed for scheme-switching and subsequent evaluation of Phase 1 - 3. Note that during Phase 1, we modulus switch to $Q = 268496897 \cdot 268460033$ to exploit the CRT and decrease the number of modular additions/multiplications

Parameter	n	N	$\log_2(q)$	$\log_2(Q)$	B	B_{rgsw}	ℓ_{rgsw}	σ^2
Value	512	2^{11}	12	56	2^8	2^4	8	3.19^2

Table 4: Parameters of the response and Respire dimensions. Note that the values of ν_2 depend on each database size.

Parameter	N_1	Q_1	ν_1	ν_2	ν_3
Value	512	2^{20}	11	$\{7, 9, 12\}$	2

We note that we instantiate the PRNG query compression using AES-CTR as described in [72]. Likewise, we use AES-CTR for our transciphering benchmarks. We motivate this choice by observing that both LWE and AES-CTR are unauthenticated and AES-CTR allows for block-wise parallel transciphering.

5.2 Pirouette performance and benchmarks

We compare Pirouette to the combination of a state-of-the-art transciphering method [9, 14] and Respire which we denote by T-Respire and give an overview of our results in Table 5 and Table 6. Furthermore, we give a breakdown of the timings in Figure 3.

We begin the discussion by observing that Pirouette has a large overhead in terms of offline communication. Our approach performs several blind-rotations which require blind-rotation keys. It is possible to only use the same key for each one, however this would come with a degradation of performance as the basis parameter L would need to be unified. Furthermore, during the execution of Pirouette and Respire, most memory is consumed by the database itself: as the database is known ahead of time, we apply number theoretic transforms on each polynomial record. Unlike a DFT, the NTT is in general not norm preserving, hence the transformed records are significantly larger and e.g. the 8GB database consumes 128GB of RAM after this transformation.

We note that the best computation time and throughput are given by Respire, which is not surprising as the least amount of work is performed. Next is the parallelized version of Pirouette followed by T-Respire and Pirouette *without* parallelization which show similar results. Figure 3 visualizes the benefit obtained from parallelization. In practice, a majority of the time is spent in the LWEtoRGSW step, which needs to be repeated for every bit in the index. However, since this conversion step is independent between bits, it can be trivially parallelized obtaining major speedups. We observe that parallelization of `BasicBitDecomp` has a minor impact which is attributable to the fact that it is performed proportionally to the number of digits to be decomposed, in our cases between 4 and 5 times. In the case of `DigitDecomp`, any variations in timings are not due to the parallelization, but rather noise in the environment, as the algorithm cannot be paralellized in a trivial manner.

Finally, Table 7 demonstrates the major benefit of our approach. In all cases, Pirouette's query size is the smallest by a factor of 100-400 when compared to respire, but also by a factor of 4-9 if we combine respire and the transciphering approach. We stress that in Pirouette's case, 32 out of the 36B are dedicated to the seed of the PRNG as we also choose a random IV. If we decide to always use an IV equal to zero, the query size drops further to 20B.

5.3 Performance evaluation of Pirouette extensions for encrypted databases

We evaluate the option of applying Pirouette on encrypted databases and give an overview of the results in Table 8. We note that we rely on timings obtained from the sequential evaluation of Pirouette in Subsection 5.2 to determine the timings, and do not implement this approach.

We justify this extrapolation as follows. We observe that the noise growth will, in fact be *slower* than for an unencrypted database: in our setting CMUX only induces an additive noise growth, whereas Respire's Phase 1 the noise grows multiplicatively w.r.t. the norm of the polynomials that encode our database. Furthermore, the depth of subsequent CMUX performed before reaching Phase 3 corresponds to $\nu_1 + \nu_2$, compared to the unencrypted setting in which the selector construction and Phase 2 already require a depth of $\nu_1 + \nu_2$ and the noise growth of Phase 1 still needs to be taken into account. Finally, it can be seen that the performance in the encrypted database case is dominated by CMUX and the timings in Table 8 are computed by determining the time necessary to perform a single CMUX and scale this value by the number of products necessary. For the timings in the parallelized setting, we note that in the folding stage for a fixed r all CMUXes may be computed independently. Note that the timings obtained in this manner are in fact an upper bound of the true performance, as more optimised parameters may be determined that take the slower noise growth into account.

Briefly, Table 8 implies that without parallelization, performing the queries will quickly become too inefficient, with timings ranging from 3 minutes to 1.5 hours. On the other hand, parallelizing the process yields more practical timings ranging from 16 seconds to less than 2 minutes.

Table 5: Performance overview of Respire, T-Respire and T-Respire with parallelization (32 cores), using database sizes identical as in [19].

Database	Metric	Respire	T-Respire	T-Respire (par.)
$2^{20} \times 256$ B	Offline Comm.	3.9MB	91MB	91MB
(256 MB)	Query Size	4.1 KB	144B	144B
	Response Size	2.0 KB	2.0 KB	2.0KB
	Computation	1.5 s	217s	15s
	Throughput	170MB/s	1.1MB/s	17MB/s
$2^{22} \times 256$ B	Offline Comm.	3.9 MB	91MB	91MB
(1 GB)	Query Size	7.7 KB	208B	208B
	Response Size	2.0 KB	2.0 KB	2.0KB
	Computation	4s	296s	22s
	Throughput	256MB/s	3.4MB/s	46.2MB/s
$2^{25} \times 256$ B	Offline Comm.	3.9 MB	91MB	91GB
(8 GB)	Query Size	14.8 KB	336B	336B
	Response Size	2 KB	2 KB	2KB
	Computation	29.9 s	486s	59.8s
	Throughput	273MB/s	16MB/s	136MB/s

Table 6: Overview of Pirouette performance, using database sizes identical as in [19] and 32-core parallelization.

Database	Metric	Pirouette	Pirouette (par. Phase 0)	Pirouette (Full par.)
$2^{20} \times 256$ B	Offline Comm.		1.2GB	
(256 MB)	Query Size		36B	
	Response Size		2.5KB	
	Computation	19s	8s	7s
	Throughput	13MB/s	32MB/s	36MB/s
$2^{22} \times 256$ B	Offline Comm.		1.2GB	
(1 GB)	Query Size		36B	
	Response Size		2.5KB	
	Computation	26s	12s	9s
	Throughput	39MB/s	85MB/s	109MB/s
$2^{25} \times 256$ B	Offline Comm.		1.2GB	
(8 GB)	Query Size		36B	
	Response Size		2.5KB	
	Computation	60s	51s	14s
	Throughput	137MB/s	150MB/s	585MB/s

Table 7: Query size reduction factor using Pirouette

Database Size	Respire	T-Respire
256M	116x	4x
1GB	219x	5.7x
8GB	420x	9.3x

Table 8: Extrapolated performance of Pirouette on encrypted databases. The parallelized setting uses 32 cores.

Database size	256MB	1GB	8GB
Time (s.)	202	788	6177
Time (s.) (par.)	16	42	101

6 Conclusion

In this work we introduce Pirouette, a PIR protocol, that improves on Respire [19] by significantly decreasing the query size. For a database of 2^{25} records, the query size is just 36B. Moreover, if the query seed is set once by the server, then the query size drops to only 32 bits, resulting in an exceptionally low expansion factor of $32/25 = 1.28$.

However, certain aspects of the approach may be further improved on. Specifically, we note that many subprocedures in Pirouette utilize the blind-rotation technique first described in [26]. Recently,

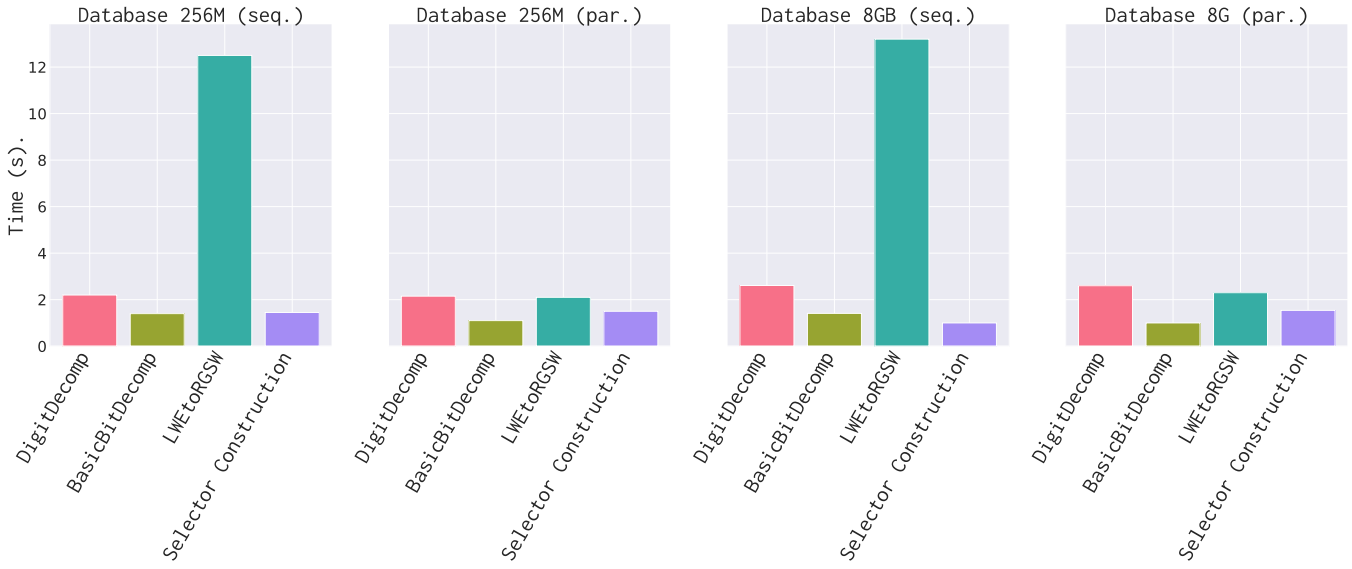

Figure 3: Breakdown of the time spent in Phase 0 of Pirouette for database sizes 256MB and 8GB

several works [15, 54] introduced blind-rotation approaches that rely internally on the NTRU scheme. The primary advantage of these methods is the superior performance in terms of time complexity, which is roughly double that of conventional blind-rotation algorithms. By incorporating these methods into Pirouette, the overall computation time may be decreased substantially, which we leave as future work.

References

[1] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR: Private Information Retrieval for Everyone. *PoPETs* 2016, 2 (April 2016), 155–174. doi:10.1515/popets-2016-0010

[2] Ehud Aharoni, Nir Drucker, Gilad Ezov, Eyal Kushnir, Hayim Shaul, and Omri Soceanu. 2023. E2E near-standard and practical authenticated transciphering. Cryptology ePrint Archive, Paper 2023/1040. https://eprint.iacr.org/2023/1040

[3] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. *J. Math. Cryptol.* 9, 3 (2015), 169–203. http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml

[4] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. 2021. Communication-Computation Trade-offs in PIR. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 1811–1828.

[5] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. 2018. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 962–979. doi:10.1109/SP.2018.00062

[6] Diego F. Aranha, Antonio Guimarães, Clément Hoffmann, and Pierrick Méaux. 2024. Secure and efficient transciphering for FHE-based MPC. Cryptology ePrint Archive, Paper 2024/1702. https://eprint.iacr.org/2024/1702

[7] Sofiane Azogagh, Zelma Aubin Birba, Marc-Olivier Killijian, and Félix Larose-Gervais. 2024. RevoLUT : Rust Efficient Versatile Oblivious Look-Up-Tables. Cryptology ePrint Archive, Paper 2024/1935. https://eprint.iacr.org/2024/1935

[8] Amos Beimel, Yuval Ishai, and Tal Malkin. 2000. Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing. In *CRYPTO 2000 (LNCS, Vol. 1880)*, Mihir Bellare (Ed.). Springer, Berlin, Heidelberg, 55–73. doi:10.1007/3-540-44598-6_4

[9] Sonia Belaïd, Nicolas Bon, Aymen Boudguiga, Renaud Sirdey, Daphné Trama, and Nicolas Ye. 2025. Further Improvements in AES Execution over TFHE: Towards Breaking the 1 sec Barrier. Cryptology ePrint Archive, Paper 2025/075. https://eprint.iacr.org/2025/075

[10] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2023. Parameter Optimization and Larger Precision for (T)FHE. *Journal of Cryptology* 36, 3 (09 Jun 2023).

[11] Song Bian, Haowen Pan, Jiaqi Hu, Zhou Zhang, Yunhao Fu, Jiafeng Hua, Yi Chen, Bo Zhang, Yier Jin, Jin Dong, and Zhenyu Guan. 2025. Engorgio: An Arbitrary-Precision Unbounded-Size Hybrid Encrypted Database via Quantized Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2025/198. https://eprint.iacr.org/2025/198

[12] Song Bian, Zhou Zhang, Haowen Pan, Ran Mao, Zian Zhao, Yier Jin, and Zhenyu Guan. 2023. HE3DB: An Efficient and Elastic Encrypted Database Via Arithmetic-And-Logic Fully Homomorphic Encryption. In *ACM CCS 2023*, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM Press, 2930–2944. doi:10.1145/3576915.3616608

[13] Jacob Blindenbach, Jiayi Kang, Seungwan Hong, Caline Karam, Thomas Lehner, and Gamze Gürsoy. 2024. SQUiD: ultra-secure storage and analysis of genetic data for the advancement of precision medicine. *Genome Biology* 25, 1 (2024), 1–27.

[14] Nicolas Bon, David Pointcheval, and Matthieu Rivain. 2024. Optimized Homomorphic Evaluation of Boolean Functions. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024, 3 (Jul. 2024), 302–341. doi:10.46586/tches.v2024.i3.302-341

[15] Charlotte Bonte, Ilia Iliashenko, Jeongeun Park, Hilder V. L. Pereira, and Nigel P. Smart. 2022. FINAL: Faster FHE Instantiated with NTRU and LWE. In *ASIACRYPT 2022, Part II (LNCS, Vol. 13792)*, Shweta Agrawal and Dongdai Lin (Eds.). Springer, Cham, 188–215. doi:10.1007/978-3-031-22966-4_7

[16] Nikita Borisov, George Danezis, and Ian Goldberg. 2015. DP5: A Private Presence Service. *PoPETs* 2015, 2 (April 2015), 4–24. doi:10.1515/popets-2015-0008

[17] Zvika Brakerski. 2012. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Berlin, Heidelberg, 868–886. doi:10.1007/978-3-642-32009-5_50

[18] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*, Shafi Goldwasser (Ed.). ACM, 309–325. doi:10.1145/2090236.2090262

[19] Alexander Burton, Samir Jordan Menon, and David J. Wu. 2024. Respire: High-Rate PIR for Databases with Small Records. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 1463–1477. doi:10.1145/3658644.3690328

[20] Christian Cachin, Silvio Micali, and Markus Stadler. 1999. Computationally Private Information Retrieval with Polylogarithmic Communication. In *EUROCRYPT'99 (LNCS, Vol. 1592)*, Jacques Stern (Ed.). Springer, Berlin, Heidelberg, 402–414. doi:10.1007/3-540-48910-X_28

[21] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. 2019. New Techniques for Multi-value Input Homomorphic Evaluation and Applications. In *CT-RSA 2019 (LNCS, Vol. 11405)*, Mitsuru Matsui (Ed.). Springer, Cham, 106–126. doi:10.1007/978-3-030-12612-4_6

[22] Yan-Cheng Chang. 2004. Single Database Private Information Retrieval with Logarithmic Communication. In *ACISP 04 (LNCS, Vol. 3108)*, Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan (Eds.). Springer, Berlin, Heidelberg, 50–61.

doi:10.1007/978-3-540-27800-9_5

[23] Hao Chen, Ilaria Chillotti, and Ling Ren. 2019. Onion Ring ORAM: Efficient Constant Bandwidth Oblivious RAM from (Leveled) TFHE. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 345–360. doi:10.1145/3319535.3354226

[24] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. 2021. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In *ACNS 21International Conference on Applied Cryptography and Network Security, Part I (LNCS, Vol. 12726)*, Kazue Sako and Nils Ole Tippenhauer (Eds.). Springer, Cham, 460–479. doi:10.1007/978-3-030-78372-3_18

[25] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *ASIACRYPT 2016, Part I (LNCS, Vol. 10031)*, Jung Hee Cheon and Tsuyoshi Takagi (Eds.). Springer, Berlin, Heidelberg, 3–33. doi:10.1007/978-3-662-53887-6_1

[26] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2017. Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. In *ASIACRYPT 2017, Part I (LNCS, Vol. 10624)*, Tsuyoshi Takagi and Thomas Peyrin (Eds.). Springer, Cham, 377–408. doi:10.1007/978-3-319-70694-8_14

[27] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology* 33, 1 (Jan. 2020), 34–91. doi:10.1007/s00145-019-09319-x

[28] Ilaria Chillotti, Marc Joye, and Pascal Paillier. 2021. Programmable Bootstrapping Enables Efficient Homomorphic Inference of Deep Neural Networks. In *Cyber Security Cryptography and Machine Learning - 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8-9, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 12716)*, Shlomi Dolev, Oded Margalit, Benny Pinkas, and Alexander A. Schwarzmann (Eds.). Springer, 1–19. doi:10.1007/978-3-030-78086-9_1

[29] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. 2021. Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In *ASIACRYPT 2021, Part III (LNCS, Vol. 13092)*, Mehdi Tibouchi and Huaxiong Wang (Eds.). Springer, Cham, 670–699. doi:10.1007/978-3-030-92078-4_23

[30] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. 1995. Private Information Retrieval. In *36th FOCS*. IEEE Computer Society Press, 41–50. doi:10.1109/SFCS.1995.492461

[31] Kelong Cong, Debajyoti Das, Jeongeun Park, and Hilder V. L. Pereira. 2022. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, 563–577. doi:10.1145/3548606.3560702

[32] Kelong Cong, Robin Geelen, Jiayi Kang, and Jeongeun Park. 2024. Revisiting Oblivious Top-k Selection with Applications to Secure k-NN Classification. In *Selected Areas in Cryptography - SAC 2024 - 31st International Conference, Montreal, QC, Canada, August 28-30, 2024, Revised Selected Papers, Part I (Lecture Notes in Computer Science, Vol. 15516)*, Maria Eichlseder and Sébastien Gambs (Eds.). Springer, 3–25. doi:10.1007/978-3-031-82852-2_1

[33] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. 2022. Single-Server Private Information Retrieval with Sublinear Amortized Time. In *EUROCRYPT 2022, Part II (LNCS, Vol. 13276)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, Cham, 3–33. doi:10.1007/978-3-031-07085-3_1

[34] Henry Corrigan-Gibbs and Dmitry Kogan. 2020. Private Information Retrieval with Sublinear Online Time. In *EUROCRYPT 2020, Part I (LNCS, Vol. 12105)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, Cham, 44–75. doi:10.1007/978-3-030-45721-1_3

[35] Orel Cosseron, Clément Hoffmann, Pierrick Méaux, and François-Xavier Standaert. 2022. Towards Case-Optimized Hybrid Homomorphic Encryption - Featuring the Elisabeth Stream Cipher. In *ASIACRYPT 2022, Part III (LNCS, Vol. 13793)*, Shweta Agrawal and Dongdai Lin (Eds.). Springer, Cham, 32–67. doi:10.1007/978-3-031-22969-5_2

[36] Alex Davidson, Gonçalo Pestana, and Sofia Celi. 2023. FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval. *PoPETs* 2023, 1 (Jan. 2023), 365–383. doi:10.56553/popets-2023-0022

[37] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. 2018. PIR-PSI: Scaling Private Contact Discovery. *PoPETs* 2018, 4 (Oct. 2018), 159–178. doi:10.1515/popets-2018-0037

[38] Léo Ducas and Daniele Micciancio. 2014. Improved Short Lattice Signatures in the Standard Model. In *CRYPTO 2014, Part I (LNCS, Vol. 8616)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, Berlin, Heidelberg, 335–352. doi:10.1007/978-3-662-44371-2_19

[39] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *EUROCRYPT 2015, Part I (LNCS, Vol. 9056)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Berlin, Heidelberg, 617–640. doi:10.1007/978-3-662-46800-5_24

[40] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Report 2012/144. https://eprint.iacr.org/2012/144

[41] Craig Gentry and Shai Halevi. 2019. Compressible FHE with Applications to PIR. In *TCC 2019, Part II (LNCS, Vol. 11892)*, Dennis Hofheinz and Alon Rosen (Eds.). Springer, Cham, 438–464. doi:10.1007/978-3-030-36033-7_17

[42] Craig Gentry, Shai Halevi, Chris Peikert, and Nigel P. Smart. 2012. Ring Switching in BGV-Style Homomorphic Encryption. In *SCN 12 (LNCS, Vol. 7485)*, Ivan Visconti and Roberto De Prisco (Eds.). Springer, Berlin, Heidelberg, 19–37. doi:10.1007/978-3-642-32928-9_2

[43] Craig Gentry and Zulfikar Ramzan. 2005. Single-Database Private Information Retrieval with Constant Communication Rate. In *ICALP 2005 (LNCS, Vol. 3580)*, Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung (Eds.). Springer, Berlin, Heidelberg, 803–815. doi:10.1007/11523468_65

[44] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *CRYPTO 2013, Part I (LNCS, Vol. 8042)*, Ran Canetti and Juan A. Garay (Eds.). Springer, Berlin, Heidelberg, 75–92. doi:10.1007/978-3-642-40041-4_5

[45] Ashrujit Ghoshal, Mingxun Zhou, and Elaine Shi. 2024. Efficient Pre-processing PIR Without Public-Key Cryptography. In *EUROCRYPT 2024, Part VI (LNCS, Vol. 14656)*, Marc Joye and Gregor Leander (Eds.). Springer, Cham, 210–240. doi:10.1007/978-3-031-58751-1_8

[46] Gamze Gürsoy, Eduardo Chielle, Charlotte M Brannon, Michail Maniatakos, and Mark Gerstein. 2022. Privacy-preserving genotype imputation with fully homomorphic encryption. *Cell systems* 13, 2 (2022), 173–182.

[47] Shai Halevi, Yuriy Polyakov, and Victor Shoup. 2019. An Improved RNS Variant of the BFV Homomorphic Encryption Scheme. In *CT-RSA 2019 (LNCS, Vol. 11405)*, Mitsuru Matsui (Ed.). Springer, Cham, 83–105. doi:10.1007/978-3-030-12612-4_5

[48] Shai Halevi and Victor Shoup. 2014. Algorithms in HElib. In *CRYPTO 2014, Part I (LNCS, Vol. 8616)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, Berlin, Heidelberg, 554–571. doi:10.1007/978-3-662-44371-2_31

[49] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. 2023. Private Web Search with Tiptoe. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, Jason Flinn, Margo I. Seltzer, Peter Druschel, Antoine Kaufmann, and Jonathan Mace (Eds.). ACM, 396–416. doi:10.1145/3600006.3613134

[50] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3889–3905.

[51] Clément Hoffmann, Pierrick Méaux, and Thomas Ricosset. 2020. Transciphering, Using FiLIP and TFHE for an Efficient Delegation of Computation. In *INDOCRYPT 2020 (LNCS, Vol. 12578)*, Karthikeyan Bhargavan, Elisabeth Oswald, and Manoj Prabhakaran (Eds.). Springer, Cham, 39–61. doi:10.1007/978-3-030-65277-7_3

[52] Robin Jadoul, Axel Mertens, Jeongeun Park, and Hilder V. L. Pereira. 2024. NTRU-Based FHE for Larger Key and Message Space. In *ACISP 24, Part I (LNCS, Vol. 14895)*, Yannan Li Tianqing Zhu (Ed.). Springer, Singapore, 141–160. doi:10.1007/978-981-97-5025-2_8

[53] Marc Joye and Pascal Paillier. 2022. Blind Rotation in Fully Homomorphic Encryption with Extended Keys. In *Cyber Security, Cryptology, and Machine Learning - 6th International Symposium, CSCML 2022, Be'er Sheva, Israel, June 30 - July 1, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13301)*, Shlomi Dolev, Jonathan Katz, and Amnon Meisels (Eds.). Springer, 1–18. doi:10.1007/978-3-031-07689-3_1

[54] Kamil Kluczniak. 2022. NTRU-v-um: Secure Fully Homomorphic Encryption from NTRU with Small Modulus. In *ACM CCS 2022*, Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi (Eds.). ACM Press, 1783–1797. doi:10.1145/3548606.3560700

[55] Dmitry Kogan and Henry Corrigan-Gibbs. 2021. Private Blocklist Lookups with Checklist. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 875–892.

[56] Eyal Kushilevitz and Rafail Ostrovsky. 1997. Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval. In *38th FOCS*. IEEE Computer Society Press, 364–373. doi:10.1109/SFCS.1997.646125

[57] Svenja Lage, Felicitas Hoermann, Felix Hanke, and Michael Karl. 2025. Privacy-Preserving Collision-Risk Assessment for LEO Satellites. https://fhe.org/conferences/conference-2025/resources Talk at The 4th Annual FHE.org Conference on Fully Homomorphic Encryption, Sofia, Bulgaria, 2025.

[58] Arthur Lazzaretti and Charalampos Papamanthou. 2023. TreePIR: Sublinear-Time and Polylog-Bandwidth Private Information Retrieval from DDH. In *CRYPTO 2023, Part II (LNCS, Vol. 14082)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, Cham, 284–314. doi:10.1007/978-3-031-38545-2_10

[59] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz. 2024. Hintless Single-Server Private Information Retrieval. In *CRYPTO 2024, Part IX (LNCS, Vol. 14928)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 183–217. doi:10.1007/978-3-031-68400-5_6

[60] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. 2023. Doubly Efficient Private Information Retrieval and Fully Homomorphic RAM Computation from Ring

LWE. In *55th ACM STOC*, Barna Saha and Rocco A. Servedio (Eds.). ACM Press, 595–608. doi:10.1145/3564246.3585175

[61] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. 2025. Black Box Crypto is Useless for Doubly Efficient PIR. Cryptology ePrint Archive, Paper 2025/552. https://eprint.iacr.org/2025/552

[62] Zeyu Liu, Daniele Micciancio, and Yuriy Polyakov. 2022. Large-Precision Homomorphic Sign Evaluation Using FHEW/TFHE Bootstrapping. In *ASIACRYPT 2022, Part II (LNCS, Vol. 13792)*, Shweta Agrawal and Dongdai Lin (Eds.). Springer, Cham, 130–160. doi:10.1007/978-3-031-22966-4_5

[63] Ming Luo, Feng-Hao Liu, and Han Wang. 2024. Faster FHE-Based Single-Server Private Information Retrieval. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 1405–1419. doi:10.1145/3658644.3690233

[64] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On Ideal Lattices and Learning with Errors over Rings. In *EUROCRYPT 2010 (LNCS, Vol. 6110)*, Henri Gilbert (Ed.). Springer, Berlin, Heidelberg, 1–23. doi:10.1007/978-3-642-13190-5_1

[65] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2013. A Toolkit for Ring-LWE Cryptography. In *EUROCRYPT 2013 (LNCS, Vol. 7881)*, Thomas Johansson and Phong Q. Nguyen (Eds.). Springer, Berlin, Heidelberg, 35–54. doi:10.1007/978-3-642-38348-9_3

[66] Pierrick Méaux, Jeongeun Park, and Hilder V. L. Pereira. 2024. Towards Practical Transciphering for FHE with Setup Independent of the Plaintext Space. *CiC* 1, 1 (2024), 20. doi:10.62056/anxrxrxqi

[67] Samir Jordan Menon and David J. Wu. 2022. SPIRAL: Fast, High-Rate Single-Server PIR via FHE Composition. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 930–947. doi:10.1109/SP46214.2022.9833700

[68] Samir Jordan Menon and David J. Wu. 2024. YPIR: High-Throughput Single-Server PIR with Silent Preprocessing. In *USENIX Security 2024*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association.

[69] Daniele Micciancio and Yuriy Polyakov. 2021. Bootstrapping in FHEW-like Cryptosystems. In *WAHC '21: Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography, Virtual Event, Korea, 15 November 2021*. WAHC@ACM, 17–28. doi:10.1145/3474366.3486924

[70] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response Efficient Single-Server PIR. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 2292–2306. doi:10.1145/3460120.3485381

[71] Michael Naehrig, Kristin E. Lauter, and Vinod Vaikuntanathan. 2011. Can homomorphic encryption be practical?. In *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, Christian Cachin and Thomas Ristenpart (Eds.). ACM, 113–124. https://dl.acm.org/citation.cfm?id=2046682

[72] National Institute of Standards and Technology. 2015. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators*. Technical Report. U.S. Department of Commerce, Washington, D.C. doi:10.6028/NIST.SP.800-90Ar1

[73] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. 2023. Towards Practical Doubly-Efficient Private Information Retrieval. Cryptology ePrint Archive, Report 2023/1510. https://eprint.iacr.org/2023/1510

[74] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. 2024. On Algebraic Homomorphic Encryption and its Applications to Doubly-Efficient PIR. Cryptology ePrint Archive, Paper 2024/1307. https://eprint.iacr.org/2024/1307

[75] Jeongeun Park and Mehdi Tibouchi. 2020. SHECS-PIR: Somewhat Homomorphic Encryption-Based Compact and Scalable Private Information Retrieval. In *ESORICS 2020, Part II (LNCS, Vol. 12309)*, Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider (Eds.). Springer, Cham, 86–106. doi:10.1007/978-3-030-59013-0_5

[76] Oded Regev. 2005. On lattices, learning with errors, random linear codes, and cryptography. In *37th ACM STOC*, Harold N. Gabow and Ronald Fagin (Eds.). ACM Press, 84–93. doi:10.1145/1060590.1060603

[77] Ling Ren, Muhammad Haris Mughees, and I Sun. 2024. Simple and Practical Amortized Sublinear Private Information Retrieval using Dummy Subsets. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 1420–1433. doi:10.1145/3658644.3690266

[78] Leonard Schild, Aysajan Abidin, and Bart Preneel. 2024. Fast Transciphering Via Batched And Reconfigurable LUT Evaluation. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024, 4 (Sep. 2024), 205–230. doi:10.46586/tches.v2024.i4.205-230

[79] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. 2020. Epione: Lightweight Contact Tracing with Strong Privacy. *IEEE Data Eng. Bull.* 43, 2 (2020), 95–107. http://sites.computer.org/debull/A20june/p95.pdf

[80] Ruida Wang, Yundi Wen, Zhihao Li, Xianhui Lu, Benqiang Wei, Kun Liu, and Kunpeng Wang. 2024. Circuit Bootstrapping: Faster and Smaller. In *EUROCRYPT 2024, Part II (LNCS, Vol. 14652)*, Marc Joye and Gregor Leander (Eds.). Springer, Cham, 342–372. doi:10.1007/978-3-031-58723-8_12

[81] Zhikun Wang and Ling Ren. 2024. Single-Server Client Preprocessing PIR with Tight Space-Time Trade-off. Cryptology ePrint Archive, Paper 2024/1845. https://eprint.iacr.org/2024/1845

[82] Zhou Zhang, Song Bian, Zian Zhao, Ran Mao, Haoyi Zhou, Jiafeng Hua, Yier Jin, and Zhenyu Guan. 2024. ArcEDB: An Arbitrary-Precision Encrypted Database via (Amortized) Modular Homomorphic Encryption. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 4613–4627. doi:10.1145/3658644.3670384

[83] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. 2024. Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation. In *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 4296–4314. doi:10.1109/SP54263.2024.00055