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 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 introduces a novel alternative to transciphering for reducing the client-to-server communication: the client transmits only one component of a high-precision LWE ciphertext, from which we design a modular procedure to extract the inputs for the subsequent homomorphic computation. The efficiency of this approach is demonstrated by our Pirouette protocol, which achieves a query size of just 36 B. 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 its variant Pirouette$^{\mathsf{H}}$ enables additional tradeoffs among query size, throughput, and offline communication, demonstrating the flexibility of our design.

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 [22, 45] and tracing [92], safe browsing [65], privacy-preserving genome imputation [55], and secure collision-risk assessment for satellites [67].

This work focuses on *single-server* [66] PIR protocols rather than *multi-server* [38] 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 [28], with subsequent improvements [30, 52] leveraging different algebraic structures.

More recent single-server PIR protocols [4, 6, 27, 44, 59, 69, 79, 80, 82, 88] often employ fully homomorphic encryption (FHE) schemes [23, 26, 35, 47, 48]. 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 bandwidth-constrained devices or when queries are transmitted over long distances, such as in satellite-to-ground communication [67].

Developing FHE-based PIR protocols with small query sizes while maintaining computation costs reasonable is both crucial and challenging. Prior works [27, 79, 82] 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 $\mathcal{N} = 2^{25}$ records is reported in Respire [27].

To further lower the query size, a common approach in FHE applications [7, 39] is transciphering [83]. 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 [11, 19] and Respire, which is referred to as T-Respire throughout the paper, gives 336 B for $\mathcal{N} = 2^{25}$ records, at the cost of 16.2x increase of running time compared to Respire.

This substantial computational overhead hinders the broader use of transciphering. To address this limitation, we explore alternative approaches that reduce the client-to-server communication cost without significantly increasing computation costs. Such techniques can benefit a wide range of FHE applications. In this work, we demonstrate the efficiency of our approach through a novel PIR protocol.

The Pirouette protocol. We present Pirouette, a query-efficient single-server PIR protocol without transciphering. The Pirouette query is a fresh learning with errors (LWE) [89] ciphertext of the query index idx, denoted as $\mathsf{LWE}(\mathsf{idx}, \Delta)$. This fresh LWE ciphertext consists of $(n+1)$ components: the first n components are sampled uniformly from $\mathbb{Z}_q$, and the final component masks the query index idx using these n random values, the secret key, as well as a small error term. The first n components can be generated pseudorandomly in the server using a PRG seed, as proposed in [33]. As such, the client only needs to send the $(n+1)$-th component together with a PRG seed in a PIR query.

The Pirouette database adopts polynomial rings in a hypercube structure as in [27, 79]. Therefore, retrieving a record homomorphically requires query indices idx_i for each dimension. Prior works [27, 79] pack these indices in ring learning with errors (RLWE) [74] ciphertexts, which increases query size. In contrast, the Pirouette query is $\mathsf{LWE}(\mathsf{idx}, \Delta)$, and we design a novel procedure that enables the server to extract encryptions of the indices $\{\mathsf{idx}_i\}$ homomorphically.

In terms of performance, for $\mathcal{N} = 2^{25}$ records, the query size of Pirouette is only 36 B, of which 32 B are dedicated to the PRG seed. The 36 B 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 7.

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 [27]. 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 $\mathsf{LWE}(\mathsf{idx}, \Delta)$, 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 [8, 36, 40] that typically consider a small plaintext modulus of 4 or 5 bits. Upon receiving the query, the server homomorphically converts it into formats compatible with the hypercube database. Unlike many high-throughput PIR works [27, 59, 79], our query directly encrypts the index idx itself, rather than an encoded representation such as a one-hot vector. Consequently, our design naturally supports applications whose outputs are directly used as PIR inputs[1].

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.

We also introduce a variant, Pirouette$^{\mathsf{H}}$, whose query consists of LWE encryptions of all the bits $\{\mathsf{LWE}(\mathsf{idx}_i)\}_{i \in [0, \lceil \log_2 \mathcal{N} \rceil - 1]}$ directly, eliminating the need for high-precision bit decomposition. For a freshly generated query, only the $(n + 1)$-th components of these LWE ciphertexts need to be transmitted together with a PRG seed. This increases the query size from 36 B to 60 B, but improves both offline setup cost and throughput, as shown in Table 7 and Figure 4.

Constructing ν_1-bit RLWE′ selector. In Respire, database records are encoded into the plaintext space $\mathcal{R}_t$, while the one-hot encoding of the first-dimension query index of length 2^{ν_1} is represented using RLWE ciphertexts. Since the noise in multiplications between plaintext and RLWE ciphertext grows linearly with 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 ν_1-bit selector consisting of the one-hot encoding of the first-dimension query index. In Section 3.2, we present a novel construction for such a one-hot encoding of length 2^{ν_1} from ν_1 bits, following an apporach conceptually similar to building a decision tree. Note that 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.

1.2 Related work

Single-server PIR with linear server computation. Early theoretical works [10, 28, 30, 52] 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

[1] In this case, the LWE query is not fresh, hence the first n components cannot be replaced by a PRG seed.

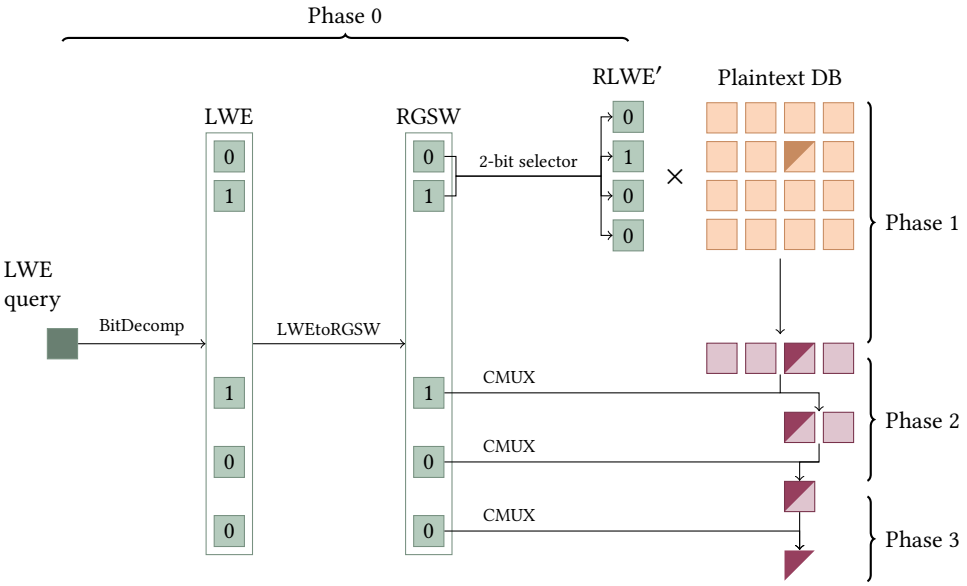


Figure 1: Processing a Pirouette query with $\nu_1 = \nu_2 = 2$ and $\nu_3 = 1$. The query expansion step (Phase 0) includes bit decomposition, LWE-to-RGSW conversion, and the construction of a ν_1-bit selector composed of RLWE′ ciphertexts. Phase 1 handles the first dimension processing using the ν_1-bit 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.

motivated extensive research [1, 4, 6, 27, 44, 50, 58, 59, 69, 73, 79, 80, 82, 88], 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, 6, 15, 27, 50, 73, 79, 82, 88]) for each client. In prior works, hints are on the order of megabytes and are used to expand RLWE queries (of minimum tens of kilobytes) via techniques such as oblivious expansion [4] and coefficient expansion algorithm [6, 31]. In Pirouette, the hint size increases to the order of gigabytes, but this enables a homomorphic query expansion from a compact LWE ciphertext, reducing the query size to only tens of bytes.
- *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 [44, 59]. 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 [58, 69, 80] 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 [10]. Schemes such as [41, 42, 54, 68, 90, 95, 98] perform a client-dependent offline phase with $O(N)$ 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. [71], removing the need for client-dependent preprocessing while still providing sublinear online complexities. Since then, DEPIR has attracted growing research interest [70, 86, 87], but no concrete efficiency has been achieved until now [86, 87].

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 [43, 60, 78] 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, 11, 19, 91, 96] 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 $\langle \mathbf{u}, \mathbf{v} \rangle$. We use notations such as $\mathbf{b}$ or $b(X)$ to denote elements of a polynomial ring. The operator $\lfloor \cdot \rceil$ denotes rounding, which extends coefficient-wise to a polynomial. 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 [27, 79], 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 [27, 38, 79]). The single-server PIR $\Pi_{\mathsf{PIR}} = (\mathsf{Setup}, \mathsf{SetupDB}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Extract})$ with client-specific hints consists of efficient algorithms with the following properties:

- $\mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{db}}) \to (\mathsf{sk}, \mathsf{evk})$. Given the security parameter λ and database parameter $\mathsf{pp}_{\mathsf{db}}$ (e.g. the size limit), output the secret key sk and the public evaluation key evk.
- $\mathsf{SetupDB}(1^\lambda, \{d_i\}_{i\in[\mathcal{N}]}) \to \mathsf{db}$. Given the security parameter λ and the $\mathcal{N}$ records in the database, output 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

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

  is at least $1 - \delta$.
- *Query privacy*: The server should learn nothing about the index queried by the client. Precisely, for any database parameter $\mathsf{pp}_{\mathsf{db}} = \mathsf{pp}_{\mathsf{db}}(\lambda)$ and any probabilistic polynomial-time adversary $\mathcal{A}$, it holds that

$$\left| \Pr[\mathcal{A}^{O_b(\mathsf{sk}, \cdot, \cdot)}(1^\lambda, \mathsf{evk}) = b] - \frac{1}{2} \right| \le \mathsf{negl}(\lambda),$$

  where $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{db}})$, $b \xleftarrow{\$} \{0, 1\}$, and the oracle $O_b(\mathsf{sk}, \mathsf{idx}_0, \mathsf{idx}_1)$ outputs $\mathsf{Query}(\mathsf{sk}, \mathsf{idx}_b)$.

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 [89] and the ring learning with errors (RLWE) problem [74, 75].

2.3.1 *LWE.*

The LWE problem is parametrized by a dimension n, integer modulus q, a key distribution χ_{key} over $\mathbb{Z}$, 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 $m \ge 1$ 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*).* Given m LWE instances $(\mathbf{A}, \mathbf{b} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} \bmod q)$ sampled from $\mathcal{D}_{\mathbf{s},\chi}$ for some $\mathbf{s} \leftarrow \chi_{key}$, the search LWE problem is to recover $\mathbf{s}$.

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 from $\mathcal{D}_{\mathbf{s},\chi}$ with respect to $\mathbf{s} \leftarrow \chi_{key}$, 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. Let χ_{key} denote a key distribution over the ring $\mathcal{R}$.

Definition 2.6 (Search RLWE*).* Given m RLWE instances $\left\{(\mathbf{a}_i, \mathbf{b}_i = \mathbf{a} \cdot \mathbf{s} + \mathbf{e} \bmod q)\right\}_{i\in[m]}$ sampled from $\mathcal{D}_{\mathbf{s},\chi}$ for some $\mathbf{s} \leftarrow \chi_{key}$, the search RLWE problem is to recover $\mathbf{s}$.

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 from $\mathcal{D}_{\mathbf{s},\chi}$ with respect to $\mathbf{s} \leftarrow \chi_{key}$, 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, both a single RLWE sample

of lattice dimension N and N LWE samples can encode N numbers, as detailed in Subsection 2.4.1. An RLWE sample requires only $2N \log q$ bits, compared to $(N^2 + N) \log q$ bits for LWE. Therefore, Phase 1-3 of the Pirouette protocol operate over $\mathcal{R}$ and rely on the hardness of RLWE for security.

However, when encoding 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 [23, 48], GSW [53], FHEW [47] and TFHE [35] homomorphic encryption schemes, along with state-of-the-art optimizations [32, 37, 39, 51, 56, 57, 72, 81, 94]. This subsection outlines the relevant FHE ciphertext types, basic operations and subroutines.

2.4.1 FHE ciphertexts. Previous PIR works such as [44, 59] use LWE ciphertexts, and [27, 79, 82] use RLWE ciphertexts and the ring variant of the Gentry-Sahai-Waters (GSW) [53] (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
$$\mathsf{RLWE}\left(T(X) \cdot X^{\varphi(\mathrm{ct}) \bmod 2N}\right)$$
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 [27, 51].

- $\mathsf{LWEtoRGSW}(\mathsf{LWE}(b)) \to \mathsf{RGSW}(b)$ 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 [35], and the state-of-the-art construction [94] 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 $m = \sum_{i=0}^{k-1} m_i \cdot 2^i$. For a small precision of $\nu = 4$ or 5 bits, conventional homomorphic bit decomposition requires ν expensive BlindRotate operations to produce ν bit-decomposed ciphertexts. To address this, we instantiate the multi-value bootstrapping [29], 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 [91]. The resulting ν-bit decomposition is presented in Algorithm 1.

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

Input: Ciphertext $\mathsf{ct} = \mathsf{LWE}_{\mathbf{s}}^{n,q=2N}(m, N/2^{\nu-1})$, 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)
  $\mathsf{acc} \leftarrow \mathsf{RLWE}_S^{N,Q}(\frac{Q}{4})$
  $\mathsf{ct}_{BR} \leftarrow \mathsf{BlindRotate}(\mathsf{ct}, \mathsf{acc})$
  ▷ $\mathsf{ct}_{BR} = \mathsf{RLWE}(\frac{Q}{4} \cdot (-1)^{m_{\nu-1}} \cdot X^{\varphi(\mathsf{ct}) \bmod N})$ ◁
  $p^{(\nu-1)} \leftarrow \sum_{j=0}^{N-1} X^j$
  $\mathsf{ct}^{(\nu-1)} \leftarrow \mathsf{Add}\left(\mathsf{Mult}\left(p^{(\nu-1)}, \mathsf{ct}_{BR}\right), \frac{Q}{4}\right)$
  $\mathsf{ct}_{out}^{(\nu-1)} \leftarrow \mathsf{SampleExtract}(\mathsf{ct}^{(\nu-1)}, 0)$
  $\mathsf{ct}'_{BR} \leftarrow 2 \cdot \mathsf{ct}_{BR}$
  for $i \leftarrow 0$ to $\nu - 2$ **do**
    $p^{(i)} \leftarrow \sum_{j=1}^{N-1} \mathsf{LSB}_{i+1}(N-j)X^j$
    $\mathsf{ct}^{(i)} \leftarrow \mathsf{Mult}\left(p^{(i)}, \mathsf{ct}'_{BR}\right)$
    $\mathsf{ct}_{out}^{(i)} \leftarrow \mathsf{SampleExtract}(\mathsf{ct}^{(i)}, 0)$
  return $\{\mathsf{ct}_{out}^{(i)}\}_{i\in[0,\nu-1]}$

However, it is impractical to directly apply Algorithm 1 with a high precision such as $\nu = 25$ bits. The performance bottleneck in Algorithm 1 is the BlindRotate, which requires $n \cdot (1 + 2\ell)$ NTTs, with each NTT costing $O(N \log N)$ integer operations. For the algorithm to be applied correctly, we would require that $q = 2N$, implying $N \geq 2^{25}$ as opposed to the more common setting where $N \in \{2^{10}, 2^{11}, 2^{12}\}$. This induces significant degradation in both performance and memory consumption.

To further extend the decomposition to $k = d \cdot \nu$ bits, we apply the digit decomposition method from [72] 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 [72, 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 3 is more efficient than directly applying Algorithm 2 with a one-bit base. For bit decomposition of $k = d \cdot \nu$ bits, our method requires $3 \cdot d$ BlindRotate operations (approximately $\frac{3k}{5}$ when $\nu = 5$), whereas the one-bit base method requires $2k$ BlindRotate operations.

Furthermore, practical values of bit precision ν for the sub procedure (Algorithm 1) are typically no larger than 5. As a concrete example, consider the bit decomposition of a value of $k = 25$ bits. Using $\nu = 5$ bits with a ring dimension $N = 2^{11}$ requires $3 \cdot \lfloor \frac{25}{5} \rfloor = 15$ blind rotations. If we instead increase ν to 9, with a larger ring dimension $N \geq 2^{13}$, the total number of blind-rotations decreases to $3 \cdot \lfloor \frac{25}{9} \rfloor = 9$; but N simultaneously grows by a factor of at least 4. Since the cost of NTT grows superlinearly with N, the overall performance becomes inferior. In practice, the performance loss would actually be worse, since for a larger ring dimension and therefore LWE modulus, we would also require a larger LWE dimension n, further increasing the total number of NTTs.

Algorithm 2 ν-bit digit decomposition

Input: Ciphertext $\mathsf{ct} = \mathsf{LWE}_{\mathbf{s}}^{n,q}(m, q/2^{d\cdot\nu})$, 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 $\{\mathsf{LWE}_{\mathbf{S}}^{n,Q=2N}(m_i, Q/2^{\nu})\}_{i\in[0,d-1]}$

function DigitDecomp(q, ct)
  ▷ *refer to [72, Algorithm 4]* ◁

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

Input: Ciphertext $\mathsf{ct} = \mathsf{LWE}_{\mathbf{s}}^{n,q}(m, q/2^{d\cdot\nu})$, 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)
  $\{\mathsf{ct}_k\}_{k\in[d]} \leftarrow$ DigitDecomp($q, 2N, \mathsf{ct}$)
  return $\{$BasicBitDecomp(ct_k, ν)$\}_{k\in[d]}$

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

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

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

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

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 v-bit selector in (2) is conceptually equivalent to the 2^v-ary CMUX gate in [61, 62], 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^v ciphertexts $\{\mathsf{RLWE}'(\delta_{i,b})\}_{i\in[0,2^v-1]}$ from control bits $\{\mathsf{RGSW}(b_j)\}_{j\in[0,v-1]}$ is to compute

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

where i_j is the bit decomposition of i such that $i = \sum_j i_j 2^j$. However, this method requires $2^v \cdot v$ 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 $\{\mathsf{RLWE}'(\delta_{i,b})\}_{i\in[0,2^v-1]}$ and the inner nodes store intermediate results. As such, our optimization instantiates the homomorphic traversal algorithm in [39] 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 $(2^v - 1)$ RGSW-RLWE′ multiplications – just $1/v$ of the naive approach. We further present an example in Figure 2 for illustration purposes.

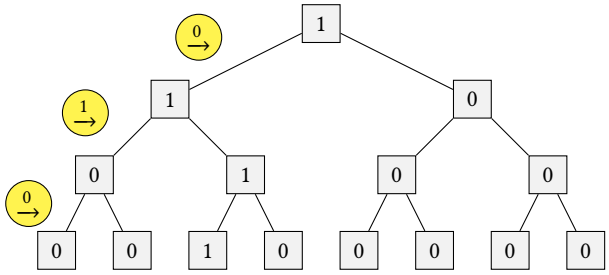


Figure 2: The construction of $\{\mathsf{RLWE}'(\delta_{i,b})\}_{i\in[0,2^3-1]}$ from $\{\mathsf{RGSW}(b_i)\}_{i\in[0,2]}$ for $b_2 = 0, b_1 = 1, b_0 = 0$. RLWE′ ciphertexts are represented as grey squares, while RGSW ciphertexts are represented as yellow circles. The root node is $\mathsf{RLWE}'(1)$; 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 [27, 50, 79, 82], particularly utilizing the database structure and response compression technique from [27]. It consists of three phases: first-dimension processing, folding and rotation, each operating on independent RLWE and RGSW ciphertexts.

Unlike prior works [27, 79, 82] 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 [27] 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 and database parameter $\mathsf{pp}_{\mathsf{db}}$, the cryptographic parameters are defined as shown in Table 1. 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 necessary evaluation keys evk. Under the standard circular security assumption, the evaluation keys evk do not leak information about the secret key.

SetupDB. For a given security parameter λ and a collection of $\mathcal{N}$ records, the parameters related to the preprocessed database are defined as shown in Table 2. Since each plaintext element in $\mathcal{R}_{N,p}$ encodes 2^{ν_3} records, the SetupDB algorithm outputs $\frac{\mathcal{N}}{2^{\nu_3}} := 2^{\nu_1+\nu_2}$ elements in the ring $\mathcal{R}_{N,p}$, denoted by $\mathsf{db} = \{\mathsf{db}_i \in \mathcal{R}_{N,p}\}_{i\in[0,2^{\nu_1+\nu_2}-1]}$.

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

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

for the desired index $\mathsf{idx} \in [\mathcal{N}]$ as follows. First, the querier samples a uniformly random PRG seed of a PRG which outputs n uniformly

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

random elements in $\mathbb{Z}_q$ to compute $\mathbf{a}$. Second, with $\mathbf{a}$, the querier computes $b = \langle \mathbf{a}, \mathbf{s} \rangle + \Delta \cdot \mathsf{idx} + e \bmod q$. The query qu consists of the PRG seed and the b part.

Answer. The server first retrieves the LWE ciphertext $(\mathbf{a}, b)$ from qu. Next, the server homomorphically expands the LWE ciphertext using the evaluation keys evk, 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 [94] 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.

In the preprocessed database $\mathsf{db} = \{\mathsf{db}_i \in \mathcal{R}_{N,p}\}_{i\in[0,2^{\nu_1+\nu_2}-1]}$, we define $\vec{\mathsf{db}}_i \coloneqq [\mathsf{db}_{i\cdot 2^{\nu_2}}, \mathsf{db}_{i\cdot 2^{\nu_2}+1}, \ldots, \mathsf{db}_{i\cdot 2^{\nu_2}+2^{\nu_2}-1}]$. As in [27], the server then proceeds with the following computations using db and the output from Phase 0:

(1) First dimension: compute

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

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

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

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

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

The final output $\mathsf{ct}_{\nu_3}^{(3)} \in \mathcal{R}_{N,Q}^2$ is an RLWE ciphertext, and the server further performs $\mathsf{ModSwitch}_{Q\to Q_1}$ and $\mathsf{RingSwitch}_{N\to N_1}$ to obtain $\mathsf{ans} \in \mathcal{R}_{N_1,Q_1}^2$. We describe ring-switching in detail in Appendix A.1, Algorithm 11.

Extract. The querier retrieves the desired record in $\mathcal{R}_{N_1,p}$ by decrypting the received ciphertext $\mathsf{ans} = (a_1, b_1) \in \mathcal{R}_{N_1,Q_1}^2$ using the RLWE secret key $s_1 \in \mathcal{R}_{N_1}$ as follows

$$\left\lfloor \frac{p}{Q_1}(b_1 - a_1 \cdot s_1 \bmod Q_1) \right\rceil \in \mathcal{R}_{N_1,p}.$$

Table 2: SetupDB parameters in Pirouette

Parameter	Meaning
$\mathcal{N}$	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 $\log \frac{N}{N_1}$

Remark 1. *Pirouette can be extended to support private queries over encrypted databases, which could be relevant to achieve the blind array access in [8] and to analyse sensitive data [15, 16, 18, 63, 97]. 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* CMUX*es to homomorphically select the encrypted record, skipping the first-dimension processing (Phase 1) in the unencrypted setting.*

Below we provide an overview of the correctness and query privacy of the Pirouette protocol. The formal analyses are detailed in Appendix A.

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* [12]. The formulae giving the failure probabilities of the building blocks used can be found in their respective works. Specifically: [72] for `DigitDecomp`, [91] for the analysis of `BasicBitDecomp`, [94] for LWEtoRGSW, [27] for Respire and e.g. [33, 46] 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 $\epsilon \le 2^{-40}$.

Query privacy. In line with previous works [1, 4, 6, 27, 44, 50, 58, 59, 69, 73, 79, 80, 82, 88] on PIR, our threat model considers an honest-but-curious server, which follows the protocol correctly but tries to deduce information from clients' inputs. In Pirouette, the client encrypts its query to an LWE ciphertext $(\mathbf{a}, b)$ and sends the b part together with a PRG seed to the server. The server then generates the $\mathbf{a}$ part from the PRG seed, and performs homomorphic computations on this LWE ciphertext.

The $\mathbf{a}$ part appears pseudorandom under standard security assumptions to an adversary without access to the seed. However, in our protocol, we only use the PRG to compress the randomness, which allows the adversary to learn the seed. This is a standard technique used in lattice-based KEMs [5, 84] and FHE protocols [9, 27, 33, 82]. Changing the distribution of the $\mathbf{a}$ part from a truly uniform distribution to a pseudo-random distribution generated by this public seed does not affect the security if the adversary does not exploit relations between the seed and the pseudo-random output. To express this, we model the PRG as the random oracle

and prove the protocol security in the random oracle model in Appendix A.2. Note that it is possible to remove the dependency on the random oracle model via the indifferentiability framework [77] where the PRG internal block cipher can be modeled as an ideal cipher instead, see [5, Section 5.1.3].

Furthermore, we prove the security of Pirouette for a server with access to the LWE query, RLWE ciphertexts generated during the evaluation, and public evaluation keys evk in Appendix A.2. The client's query privacy is guaranteed by the hardness of LWE and RLWE problems along with the circular security assumption, which is widely adopted in FHE-based applications, including prior PIR schemes.

4.2 The Pirouette$^{\mathsf{H}}$ protocol

In this subsection, we introduce Pirouette$^{\mathsf{H}}$, a variant of Pirouette that achieves slightly higher throughput at the cost of a slightly larger query size. In Pirouette, the query consists of a single LWE encryption of the query index idx, thereby minimizing the query size. By contrast, a Pirouette$^{\mathsf{H}}$ query consists of LWE encryptions of all the bits of the query index idx, and only their $(n+1)$-th components need to be transmitted together with a PRG seed. This design improves the throughput by removing the need for the high-precision bit decomposition step.

In Pirouette$^{\mathsf{H}}$, the Setup, SetupDB, and Extract procedures are the same as in Pirouette. Below we only present the Query and Answer procedures.

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

$$\{(\mathbf{a}_i, b_i) = \mathsf{LWE}_{\mathbf{s}}^{n,q}(\mathsf{idx}_i, \Delta)\}_{i\in[0,\log(\mathcal{N})-1]},$$

for the desired index $\mathsf{idx} = \sum_{i=0}^{\log(\mathcal{N})-1} \mathsf{idx}_i \cdot 2^i \in [\mathcal{N}]$ as follows. First, the querier samples a uniformly random PRG seed of a PRG which outputs $n \cdot \log(\mathcal{N})$ uniformly random elements in $\mathbb{Z}_q$ to compute all $\{\mathbf{a}_i\}_{i\in[0,\log(\mathcal{N})-1]}$. Second, the querier computes $b_i = \langle \mathbf{a}_i, \mathbf{s}\rangle + \Delta \cdot \mathsf{idx}_i + e \bmod q$, for each $i \in [0, \log(\mathcal{N}) - 1]$. The query qu now consists of the PRG seed and all b_i parts.

Answer. In Phase 0, the server skips the high-precision bit decomposition procedure and directly performs

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

Then 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. Phases 1-3 are the same as in Pirouette.

The correctness Pirouette$^{\mathsf{H}}$ follows directly from Pirouette and is therefore omitted for brevity. The security of Pirouette$^{\mathsf{H}}$ is sketched in Appendix A.2.

5 Implementation and Evaluation

In this section, we evaluate [2] Pirouette and its variant, Pirouette$^{\mathsf{H}}$, in comparison to Respire, Respire in combination transciphering (T-Respire), SimplePIR [59], and Spiral [79].

[2] Our implementation is publicly available at https://github.com/KULeuven-COSIC/Pirouette. The code includes 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 32 cores.

We optimize parameters separately for each sub-procedure used by Pirouette, as recommended in [12], 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] commit `787c05a`. We use binary keys i.e. the coefficients are sampled uniformly from $\{0, 1\}$.

Table 3 presents the relevant parameters for the digit decomposition and subsequent bit decomposition. We specify several values of the LWE dimension n, since the bit decomposition can 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 LWEtoRGSW conversion, as a smaller LWE dimension n directly reduces the computational complexity of the BlindRotate procedure, which is linear in n (see [34]).

Next, Table 4 presents the parameters employed for the scheme switching step. The values B_{RGSW} and ℓ_{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 [33, 94] 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 5 together with the values of ν_i.

We note that we instantiate the PRG query compression using CTR-DBRG with AES-128 as described in [85] using a seed size of 32 B, which supports an output length of at most 2^{67} bits. For the parameters in Tables 3, our instantiation for Pirouette$^{\mathsf{H}}$ supports a database size $\mathcal{N}$ of up to 2^{51}. 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 analysis and performance

In this section, we present a high-level overview of the analysis and performance of Pirouette, and its variant, Pirouette$^{\mathsf{H}}$.

Setup cost. The offline communication in Pirouette and Pirouette$^{\mathsf{H}}$ are substantially higher than in other PIR approaches. This increased cost stems directly from Phase 0, which employs numerous subroutines that each require additional key material. Table 6 summarizes the size of each key as a function of its hyperparameters.

In principle, some key material may be reused: e.g. several subroutines perform blind rotation and a single key could be used for all such cases. However, this would substantially degrade performance, as the parameters for each subroutine dictate the output variance and efficiency, and different steps have different requirements. As a concrete example, the first blind rotation must always be performed on an LWE sample with a large vector dimension to preserve the security guarantees of the query. By contrast, after bit decomposition, we only have LWE samples of bits and can therefore utilise a smaller LWE dimension (c.f. n_{in} vs n_{out} in Table 3).

Table 3: 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 4: 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 5: Parameters of the response and Respire dimensions. Note that the values of v_2 depend on each database size.

Parameter	N_1	Q_1	v_1	v_2	v_3
Value	512	2^{20}	11	$\{7, 9, 12\}$	2

Since the asymptotic performance of blind-rotation is linear in n, we chose to trade memory for performance. Furthermore, if smaller keys at rest are desired, key compression techniques such as [2] can be applied.

SetupDB cost. Before running the PIR protocol, the database on the server must undergo a one-time preprocessing step. This setup cost is independent of the number of queries and can be amortised over all future queries. During SetupDB, database entries are mapped to coefficients of polynomials so that they can be combined with Phase 0 outputs through polynomial addition and multiplication. To accelerate subsequent query evaluation, these polynomials are transformed using number-theoretic transforms (NTTs) with a cost of $O(N \log(N))$ arithmetic operations, where N is the ring dimension and polynomial degree.

For a database holding 2^ρ entries with record size κ bits, if each polynomial coefficient encodes p bits, we require $\frac{2^\rho \cdot \kappa}{p \cdot N}$ NTT operations. Consequently, database preprocessing depends on the user-specific parameters N, p and Q (required for the NTT). Similarly to Respire, the pre-processed database (8GB) takes up approximately 128GB of RAM.

If a database entry is modified, the corresponding polynomial(s) encoding that record must be changed and the the NTT reapplied. However, when the record size increases (e.g. if data is appended), it is possible to avoid performing a full NTT for each new record by exploiting the recursive structure of NTT and FFT algorithms.

Communication costs. In terms of online upload communication, both Pirouette and Pirouette$^{\mathsf{H}}$ outperform existing PIR methods, achieving strictly smaller query sizes such as 36 B and 60 B. The query size is entirely independent of the record size and grows logarithmically in the database size.

The online download communication behaves similarly to Respire, yielding a small and nearly constant response sizes (around 3 KB) for the evaluated parameters, as shown in Figure 4. While a smaller response could be achieved for smaller record sizes, this would require reducing either the response ring dimension or modulus, which would be infeasible beyond a certain point without compromising the security level.

Computation costs. The computation cost in Pirouette and Pirouette$^{\mathsf{H}}$ increases with the number of records, as the expensive LWEtoRGSW operation needs to be performed for each bit of the query index. On the other hand, when the record size grows, the cost of Phase 0 and the multiplicative depth of Phase 1-3 remain constant. Only the computation cost of Phases 1-3 increases proportionally.

Similar to Respire, both Pirouette and Pirouette$^{\mathsf{H}}$ benefit significantly from parallelisation. The LWEtoRGSW conversions in Phase 0, the partial sums in Phase 1 and the folding operations in Phase 2 are largely independent and can be performed concurrently. Additionally, the gadget decompositions and subsequent NTTs necessary for blind rotation in phase 0 can also easily be parallelised. The effect of parallelisation is illustrated in Figure 3, which shows substantial speedups for the costly LWEtoRGSW operations. As shown in Table 7, parallelisation improves the overall Pirouette runtime by 2.7×–4.3×.

Pirouette$^{\mathsf{H}}$ vs. Pirouette. Compared to Pirouette, the design of Pirouette$^{\mathsf{H}}$ has several implications. First, eliminating the bit decomposition reduces offline communication by nearly half, as the key material specific to bit decomposition is no longer required. Second, as shown in Figure 4, the omission of bit decomposition most notably improves performance for small database sizes, where the overall computational complexity is dominated by Phases 1–3 rather than Phase 0. Third, the query in Pirouette$^{\mathsf{H}}$ consists of LWE encryptions of all index bits, requiring transmission of all their $(n+1)$-th components along with a PRG seed. This increases the query size from 36 B to 60 B, as shown in Table 7 and Figure 4.

5.3 Comparison with Respire and T-Respire

Respire is a PIR protocol with minimal query size that serves as the foundation for Pirouette. A common strategy to further reduce query size is to incorporate transciphering; we denote Respire combined with a state-of-the-art transciphering method [11, 19] as T-Respire. We compare Pirouette and Pirouette$^{\mathsf{H}}$ against Respire and T-Respire, with an overview of our results in Table 7.

Sequential execution. Pirouette and Pirouette$^{\mathsf{H}}$ demonstrate substantial improvements over Respire and T-Respire. In sequential execution, Pirouette achieves up to a 9.3× reduction in query size

Table 6: Storage requirements for setup in bits and as a function of relevant parameters.

Key	Evaluation	Key-Switching	Squaring	Automorphism	Resp. Compression
Size	$4n \cdot \ell \cdot N \cdot \log_2(Q)$	$N \cdot \ell \cdot (n+1) \cdot \log_2(Q)$	$2N\ell \cdot \log_2(Q)$	$\log_2(N) \cdot 2N\ell \cdot \log_2(Q)$	$2N\ell \cdot \log_2(Q)$

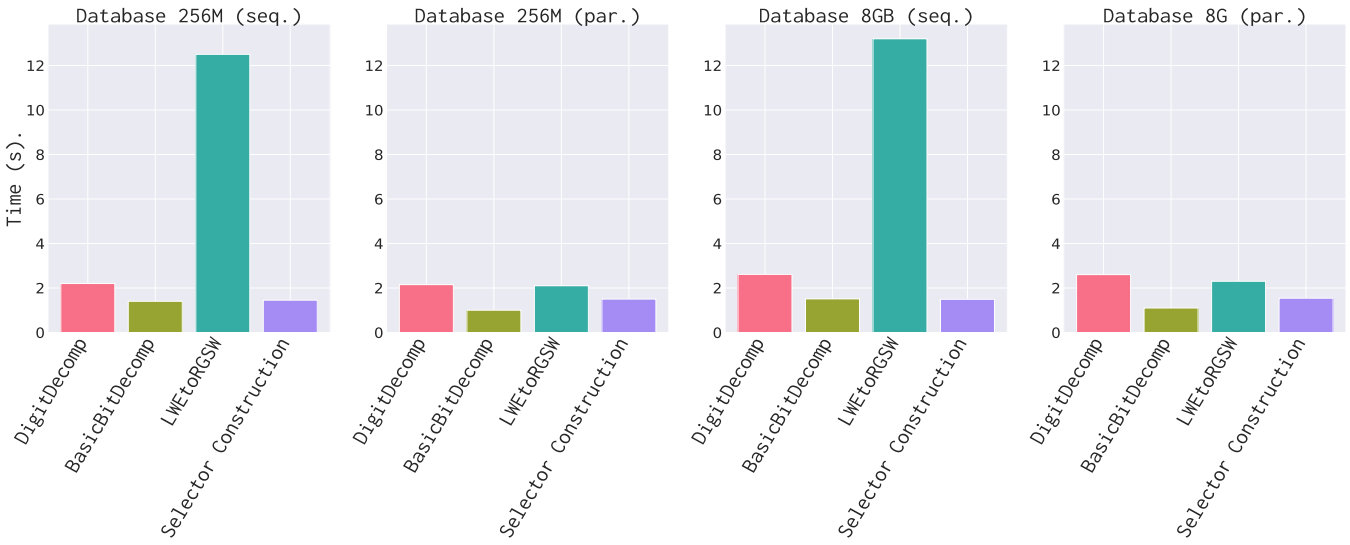


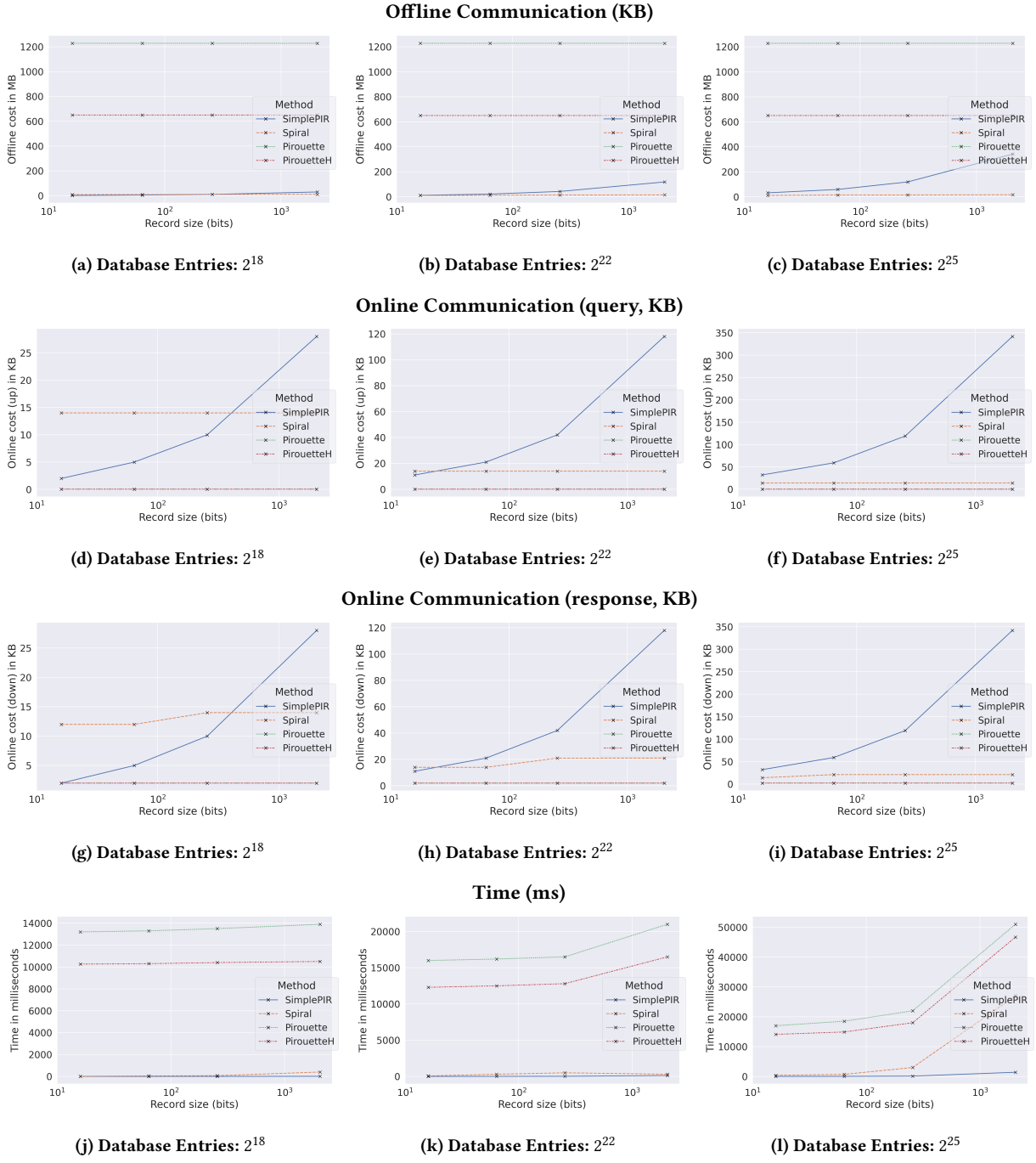
Figure 3: Breakdown of the time spent in Phase 0 of Pirouette for database sizes 256MB and 8GB

Table 7: Performance of Respire, T-Respire, Pirouette, Pirouette^H, and their 32-core parallelised variants where parallelisation applies to transciphering or phase 0, and fully parallelised Pirouette. Database size parameters are taken from Respire.

Database	Metric	Respire	T-Respire	Pirouette	Pirouette^H	T-Respire (par. trans)	Pirouette (par. phase 0)	Pirouette^H (par. phase 0)	Pirouette (full par.)
$2^{20} \times 256$ B (256 MB)	Offline Comm.	4 MB	91 MB	1.2 GB	650 MB	91 MB	1.2 GB	650 MB	1.2 GB
	Query Size	4.1 KB	144 B	36 B	55 B	144 B	36 B	55 B	36 B
	Computation	1.5 s	217 s	19 s	15 s	15 s	8 s	6 s	7 s
	Response Size					2 KB			
	Throughput	170 MB/s	1 MB/s	13 MB/s	17 MB/s	17 MB/s	32 MB/s	42 MB/s	36 MB/s
$2^{22} \times 256$ B (1 GB)	Offline Comm.	4 MB	91 MB	1.2 GB	650 MB	91 MB	1.2 GB	650 MB	1.2 GB
	Query Size	7.7 KB	208 B	36 B	57 B	208 B	36 B	57 B	36 B
	Computation	4 s	296 s	26 s	21 s	22 s	12 s	9 s	9 s
	Response Size					2 KB			
	Throughput	256 MB/s	3 MB/s	39 MB/s	48 MB / s	46 MB/s	85 MB/s	113 MB/s	109 MB/s
$2^{25} \times 256$ B (8 GB)	Offline Comm.	4 MB	91 MB	1.2 GB	650 MB	91 MB	1.2 GB	650 MB	1.2 GB
	Query Size	14.8 KB	336 B	36 B	60 B	336 B	36 B	60 B	36 B
	Computation	30 s	486 s	60 s	55 s	60 s	51 s	46 s	14 s
	Response Size					2 KB			
	Throughput	273 MB/s	16 MB/s	137 MB/s	148 MB/s	136 MB/s	150 MB/s	178 MB/s	585 MB/s

compared to T-Respire and a 420× reduction compared to Respire, while maintaining computation times only 2× slower than Respire and 8.1× faster than T-Respire.

These query size reductions and fast throughput (relative to T-Respire) come at the cost of higher offline communication: Pirouette requires 1.2 GB of offline communication, approximately 13x larger than T-Respire. Pirouette^H provides an alternative tradeoff, reducing offline communication by nearly half compared to Pirouette while achieving slightly better throughput, at the cost of up to 1.7x larger query sizes. Notably, Pirouette^H queries are still up to 5.6x smaller than T-Respire.



(a) Database Entries: 2^{18} (b) Database Entries: 2^{22} (c) Database Entries: 2^{25}

(d) Database Entries: 2^{18} (e) Database Entries: 2^{22} (f) Database Entries: 2^{25}

(g) Database Entries: 2^{18} (h) Database Entries: 2^{22} (i) Database Entries: 2^{25}

(j) Database Entries: 2^{18} (k) Database Entries: 2^{22} (l) Database Entries: 2^{25}

Figure 4: Performance of Pirouette, Pirouette^{H}, Spiral and SimplePIR. Note for online communication, the line for Pirouette appears stacked with Pirouette^{H}.

Parallelised execution. Since PIR servers typically have rich computational resources, it would be ideal to compare fully parallelised versions of these protocols as well. However, the Respire implementation by the authors does not include parallelisation, and implementing parallelisation for Respire is out of the scope. We therefore focus on comparing parallelising the primary performance bottlenecks: transciphering in T-Respire and Phase 0 in Pirouette and Pirouette^{H}.

The corresponding performance with 32-core parallelisation applied only to these bottlenecks is shown in Table 7. While Pirouette

and PIROUETTE^{H} with parallelised phase 0 remain faster than T-Respire with parallelised transciphering, all three protocols approach the runtime of Respire. This demonstrates that both transciphering and Phase 0 benefit substantially from parallelisation, and validates our approach as a viable alternative to transciphering for reducing client-to-server communication.

Asymmetric online communication. As shown in Table 7, Respire query size ranges from 4.1 KB to 14.8 KB, and the response size is always 2 KB. Both T-Respire and PIROUETTE significantly reduce the query size to less than 336 B and 36 B, respectively, while keeping the same 2 KB response size. Consequently, although PIROUETTE achieves minimal total online communication, the 2 KB server-to-client response dominates the overall online communication.

Note that transciphering has been extensively studied to reduce client-to-server communication in FHE, while reducing server-to-client communication in FHE is less explored. Recent works [20, 76] show a promising direction to compress output FHE ciphertexts using additive homomorphic encryption schemes. PIROUETTE shares the same theoretical objective as transciphering to reduce the client-to-server communication, and does not consider the server-to-client direction. This asymmetry could limit the practical applicability of PIROUETTE in practice, and further reducing the 2 KB response size represents a promising direction for future work.

5.4 Holistic performance comparison

We provide a holistic performance comparison of PIROUETTE and PIROUETTE^{H} against two state-of-the-art PIR protocols, SimplePIR [59] and Spiral [79], as shown in Figure 4. The evaluation considers four key dimensions: offline communication, query size, response size, and throughput. SimplePIR and Spiral are optimised for high throughput, whereas PIROUETTE and PIROUETTE^{H} achieve minimal online communication at the expense of offline costs and throughput. Therefore, PIROUETTE and PIROUETTE^{H} would be most applicable in scenarios where: (1) online bandwidth, particularly from client to server, is severely constrained; (2) offline bandwidth is widely available, with offline and online phases exhibiting distinct network characteristics (e.g. wired device initialization vs wireless operation); and (3) the server computation can leverage multi-core parallelisation on dedicated FHE hardware accelerators [13, 14, 49, 93]. While such acceleration can substantially improve throughput, practical deployment is still limited by current runtime and offline communication requirements.

Offline communication. All protocols involve a one-time offline setup that can be reused across multiple queries. SimplePIR's offline phase produces *database-specific* hints that are stored by clients and need to be updated for different databases. In contrast, the offline phases for the other three protocols generate *client-specific* hints, where FHE secret keys are stored by clients and FHE evaluation keys are stored by the server. In this setting, the client storage is small, and a client can query different databases without additional updates; but the server still needs to re-preprocess different databases.

In terms of size, the offline setups in PIROUETTE and PIROUETTE^{H} are 1.2 GB and 650 MB respectively, dominated by the evaluation keys stored on the server, as client storage for the secret key is less than 10 KB. In comparison, offline setups in Spiral and SimplePIR are only tens to hundreds of megabytes. Although this offline cost in PIROUETTE and PIROUETTE^{H} is a one-time expense that does not grow with the number of queries or database updates, effective amortization is impractical due to the prohibitively large number of queries required to offset the offline expense.

Online communication. Query sizes for PIROUETTE and PIROUETTE^{H} are no larger than 60 B, while Spiral and SimplePIR require tens to hundreds of kilobytes. This represents a reduction of roughly three to four orders of magnitude in query size.

Response sizes of PIROUETTE and PIROUETTE^{H} are around 3 KB, leveraging ModSwitch and RingSwitch as in Respire. In contrast, Spiral and SimplePIR generate larger response sizes of tens to hundreds of kilobytes.

Runtime. In terms of runtime, SimplePIR is the fastest, with Spiral achieving comparable performance. For example, for a database of size 2^{22} with 256-bit entries, SimplePIR and Spiral runtimes are 20 ms and 500 ms respectively, whereas PIROUETTE and PIROUETTE^{H} runtimes are 16.5 s and 12.8 s, which are slower by two to three orders of magnitude.

Overall, PIROUETTE represents a PIR protocol optimised for minimal query size. Its variant PIROUETTE^{H} achieves 20 to 30% faster runtime and 2× lower setup cost compared to PIROUETTE, while increasing query size from only 36 B to 60 B. This highlights the potential for navigating tradeoffs among query size, throughput, and offline communication.

6 Conclusion

This work introduces PIROUETTE, a PIR protocol that improves upon Respire [27] by significantly lowering 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 $32/25 = 1.28$. This further demonstrates the practicality of our novel approach to reduce the client-to-server communication, which provides a lower server-side computational overhead than transciphering and is applicable for many other FHE-based applications.

Meanwhile, certain aspects of PIROUETTE may be further improved on. Specifically, we note that many subprocedures in PIROUETTE utilize the blind-rotation technique first described in [34]. Recently, several works [21, 64] 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 stemming from that fact that RGSW samples are replaced by a different type of ciphertext. The latter contain only half as many polynomials and the modified blind-rotation reduces the number of number-theoretic transforms, the primary bottleneck of this step, by the same amount. By incorporating these methods into PIROUETTE, the overall computation time may be decreased substantially, which we leave as future work.

Acknowledgments

This work was supported by Research Council KU Leuven grant C14/24/099, CyberSecurity Research Flanders with reference number VOEWICS02, CyberSecurity Research Flanders with reference number VR20192203, and the Flemish Government through FWO SBO project MOZAIK S003321N. The authors thank Erik Pohle for valuable discussions on PRG security, and Frederik Vercauteren, Aysajan Abidin, and the anonymous PETS reviewers for their helpful feedback on this paper.

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] Erdem Alkim, Joppe W. Bos, Léo Ducas, Patrick Longa, Ilya Mironov, Michael Naehrig, Valeria Nikolaenko, Chris Peikert, Ananth Raghunathan, and Douglas Stebila. 2021. *FrodoKEM: Learning With Errors Key Encapsulation — Algorithm Specifications and Supporting Documentation.* Specification / Technical Report 20210604. FrodoKEM Project. https://frodokem.org/files/FrodoKEM-specification-20210604.pdf

[6] 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

[7] Diego F. Aranha, Antonio Guimarães, Clément Hoffmann, and Pierrick Méaux. 2025. Secure and efficient transciphering for FHE-based MPC. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2025, 3 (2025), 745–780. doi:10.46586/TCHES.V2025.I3.745-780

[8] 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

[9] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. 2024. Plaintext-Ciphertext Matrix Multiplication and FHE Bootstrapping: Fast and Fused. In *CRYPTO 2024, Part III (LNCS, Vol. 14922)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 387–421. doi:10.1007/978-3-031-68382-4_12

[10] 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

[11] 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

[12] 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).

[13] Jonas Bertels, Michiel Van Beirendonck, Furkan Turan, and Ingrid Verbauwhede. 2023. Hardware Acceleration of FHEW. In *26th International Symposium on Design and Diagnostics of Electronic Circuits and Systems, DDECS 2023, Tallinn, Estonia, May 3-5, 2023*, Maksim Jenihhin, Hana Kubátová, Nele Metens, Jaan Raik, Foisal Ahmed, and Jan Belohoubek (Eds.). IEEE, 57–60. doi:10.1109/DDECS57882.2023.10139347

[14] Jonas Bertels, Hilder V. L. Pereira, and Ingrid Verbauwhede. 2025. FINAL bootstrap acceleration on FPGA using DSP-free constant-multiplier NTTs. *IACR Trans. Cryptogr. Hardw. Embed. Syst.* 2025, 3 (2025), 293–316. doi:10.46586/TCHES.V2025.I3.293-316

[15] 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

[16] 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

[17] John Black, Phillip Rogaway, and Thomas Shrimpton. 2003. Encryption-Scheme Security in the Presence of Key-Dependent Messages. In *SAC 2002 (LNCS, Vol. 2595)*, Kaisa Nyberg and Howard M. Heys (Eds.). Springer, Berlin, Heidelberg, 62–75. doi:10.1007/3-540-36492-7_6

[18] 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.

[19] 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

[20] Antonina Bondarchuk, Olive Chakraborty, Geoffroy Couteau, and Renaud Sirdey. 2024. Downlink (T)FHE ciphertexts compression. *IACR Cryptol. ePrint Arch.* (2024), 1921. https://eprint.iacr.org/2024/1921

[21] 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

[22] 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

[23] 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

[24] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. 2019. Leveraging Linear Decryption: Rate-1 Fully-Homomorphic Encryption and Time-Lock Puzzles. In *Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part II* (Nuremberg, Germany). Springer-Verlag, Berlin, Heidelberg, 407–437. doi:10.1007/978-3-030-36033-7_16

[25] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2011. Fully Homomorphic Encryption without Bootstrapping. Cryptology ePrint Archive, Paper 2011/277. https://eprint.iacr.org/2011/277

[26] 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

[27] 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

[28] 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

[29] 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

[30] 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

[31] 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

[32] 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

[33] 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

[34] 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

[35] 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

[36] 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

[37] 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

[38] 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

[39] 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

[40] 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

[41] 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

[42] 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

[43] 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

[44] 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

[45] 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

[46] 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

[47] 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

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

[49] Robin Geelen, Michiel Van Beirendonck, Hilder V. L. Pereira, Brian Huffman, Tynan McAuley, Ben Selfridge, Daniel Wagner, Georgios D. Dimou, Ingrid Verbauwhede, Frederik Vercauteren, and David W. Archer. 2023. BASALISC: Programmable Hardware Accelerator for BGV Fully Homomorphic Encryption. *IACR TCHES* 2023, 4 (2023), 32–57. doi:10.46586/tches.v2023.i4.32-57

[50] 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

[51] 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

[52] 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

[53] 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

[54] 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

[55] 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.

[56] 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

[57] 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

[58] 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

[59] 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.

[60] 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

[61] 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

[62] 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

[63] Jaeseon Kim, Jeongeun Park, and Hyewon Sung. 2025. Private Information Retrieval based on Homomorphic Encryption, Revisited. Cryptology ePrint Archive, Paper 2025/729. https://eprint.iacr.org/2025/729

[64] 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

[65] 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.

[66] 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

[67] 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.

[68] 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

[69] 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

[70] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. 2025. Black Box Crypto is Useless for Doubly Efficient PIR. In *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part VI (Lecture Notes in Computer Science, Vol. 15606)*, Serge Fehr and Pierre-Alain Fouque (Eds.). Springer, 65–93. doi:10.1007/978-3-031-91095-1_3

[71] 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

[72] 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

[73] 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

[74] 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

[75] 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

[76] Rasoul Akhavan Mahdavi, Abdulrahman Diaa, and Florian Kerschbaum. 2023. HE is all you need: Compressing FHE Ciphertexts using Additive HE. *CoRR* abs/2303.09043 (2023). doi:10.48550/ARXIV.2303.09043 arXiv:2303.09043

[77] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. 2004. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In *TCC 2004 (LNCS, Vol. 2951)*, Moni Naor (Ed.). Springer, Berlin, Heidelberg, 21–39. doi:10.1007/978-3-540-24638-1_2

[78] 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

[79] 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

[80] 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.

[81] 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

[82] 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

[83] 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

[84] National Institute of Standards and Technology. 2024. *Module-Lattice-Based Key-Encapsulation Mechanism Standard.* Federal Information Processing Standards FIPS 203. National Institute of Standards and Technology. doi:10.6028/NIST.FIPS.203 Final version.

[85] 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

[86] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. 2024. Towards Practical Doubly-Efficient Private Information Retrieval. In *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part II (Lecture Notes in Computer Science, Vol. 14745)*, Jeremy Clark and Elaine Shi (Eds.). Springer, 264–282. doi:10.1007/978-3-031-78679-2_14

[87] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. 2025. On Algebraic Homomorphic Encryption and Its Applications to Doubly-Efficient PIR. In *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part VI (Lecture Notes in Computer Science, Vol. 15606)*, Serge Fehr and Pierre-Alain Fouque (Eds.). Springer, 34–64. doi:10.1007/978-3-031-91095-1_2

[88] 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

[89] 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

[90] 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

[91] 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

[92] 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

[93] Michiel van Beirendonck, Jan-Pieter D'Anvers, Furkan Turan, and Ingrid Verbauwhede. 2023. FPT: A Fixed-Point Accelerator for Torus Fully Homomorphic Encryption. In *ACM CCS 2023*, Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda (Eds.). ACM Press, 741–755. doi:10.1145/3576915.3623159

[94] 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

[95] Zhikun Wang and Ling Ren. 2025. Single-Server Client Preprocessing PIR with Tight Space-Time Trade-Off. In *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part VI (Lecture Notes in Computer Science, Vol. 15606)*, Serge Fehr and Pierre-Alain Fouque (Eds.). Springer, 94–122. doi:10.1007/978-3-031-91095-1_4

[96] Benqiang Wei, Xianhui Lu, Ruida Wang, Kun Liu, Zhihao Li, and Kunpeng Wang. 2024. Thunderbird: Efficient Homomorphic Evaluation of Symmetric Ciphers in 3GPP by combining two modes of TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024 (2024), 530–573. doi:10.46586/tches.v2024.i3.530-573

[97] 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

[98] 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

A Correctness and security of Pirouette

A.1 Correctness analysis for Pirouette

In this section, we will give explicit descriptions of the sub-procedures used by Pirouette, and include a thorough correctness discussion. Similar to previous works, we rely on the independence heuristic, which assumes that ciphertexts are independent. We proceed as follows: first we will discuss the fundamental aspects of the behaviour of RLWE, RLWE′ and RGSW samples. Next, we discuss the homomorphic evaluation of (negacyclic) lookup tables, which is used internally by the LWEtoRGSW procedure. Finally, we describe the selector construction and illustrate how to estimate the failure probability of Pirouette. Throughout this section, we will study (R)LWE & RGSW samples through the variance of their noise terms, based on which we can ultimately determine the failure probability.

A.1.1 Gadgets and Blind Rotation. We briefly recall basic facts about linear combinations of RLWE samples. Given ciphertexts $\mathsf{c}_i = \mathsf{RLWE}_{\mathsf{s}}^{N,Q}(\mathfrak{m}_i)$, the noise variance $\sigma^2_{c_+}$ of their sum $\mathsf{c}_+ := \sum_i \mathsf{c}_i$ satisfies

$$\sigma^2_{c_+} = \sum \sigma^2_{c_i}.$$

Furthermore, for any $\mathsf{c} = \mathsf{RLWE}_{\mathsf{s}}^{N,Q}(\mathfrak{m})$ and $\mathfrak{p} \in \mathcal{R}$, the noise variance of $\mathsf{d} = \mathsf{c} \cdot \mathfrak{m}$ satisfies

$$\sigma^2_d \leq \frac{||\mathfrak{p}||^2_\infty}{4} \cdot ||\mathfrak{p}||_0 \cdot \sigma^2_c.$$

Next, we recall the gadget product, as described in Algorithm 4, and discuss its output variance in Theorem A.1. At a high level, the purpose of G and by extension the RLWE′ samples is to enable the option of multiplying encrypted data with a polynomial of unknown norm: the RLWE′ samples are parametrised by a *basis* L and number of *digits* ℓ. By decomposing a polynomial w.r.t L, we

enforce a bound on the induced noise. For the sake of completeness, we note that in the exact case $l = \lceil \log_L(Q) \rceil$ but in general enforcing this relationship is not necessary whenever the error can be taken into account, see e.g. [33].

Algorithm 4 Gadget Product: $\otimes : \mathsf{RLWE}' \times \mathcal{R} \to \mathsf{RLWE}$

Input: RLWE′ sample $\vec{c} = \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(\mathfrak{m})$

Input: Polynomial $\mathfrak{p} \in \mathcal{R}$

Output: RLWE sample $c_{\text{out}} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m} \cdot \mathfrak{p})$

1: Extract parameters L, ℓ from G
2: $(\mathfrak{p}^{(i)})_{i \in [\ell]} \leftarrow \mathsf{G}(\mathfrak{p})$
3: $c_{\text{out}} \leftarrow \sum_{i=0} \vec{c}_i \cdot \mathfrak{p}^{(i)}$
4: **return** c_{out}

Theorem A.1. *Let* G *be the (exact) basis decomposition gadget for* L *that is* $\mathsf{G}(\mathfrak{p}) = [\mathfrak{p}^{(0)}, ..., \mathfrak{p}^{(l-1)}]$ *with* $\mathfrak{p} = \sum_{i=0}^{l-1} \mathfrak{p}^{(i)} \cdot L^i$. *Then* $\mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(\mathfrak{m}) = [\mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}), ..., \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m} \cdot L^{\ell-1})]$ *and Algorithm 4 outputs a* RLWE *sample* c_{out} *with variance*

$$\sigma^2_{\otimes} \leq l \cdot \frac{L^2}{4} \cdot N \cdot \sigma^2_c,$$

where σ^2_c *is the noise variance of the components on* $\vec{c}$.

Proof. We note that $||\mathfrak{p}^{(i)}||_\infty \leq L$, $||\mathfrak{p}^{(i)}||_0 \leq N$ and that

$$c_{\text{out}} = \sum_{i=0} \vec{c}^{(i)} \cdot \mathfrak{p}^{(i)},$$

then the claim follows. □

Recall that RGSW samples can be represented as a tuple of RLWE′ samples. We now extend the gadget product to RGSW samples in Algorithm 5 and describe its behaviour in Theorem A.2. This type of product, the external product, is leveraged to multiply RGSW and RLWE samples.

Algorithm 5 External Product $\boxtimes : \mathsf{RGSW} \times \mathsf{RLWE} \to \mathsf{RLWE}$

Input: RGSW sample $\mathsf{C} = \mathsf{RGSW}(\mathfrak{m}_0) = [\vec{c}, \vec{d}] = [\mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(-\mathfrak{s} \cdot \mathfrak{m}_0), \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(\mathfrak{m})]$

Input: RLWE sample $c = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}_1) = [\mathfrak{a}, \mathfrak{b}]$

Output: $c_{\text{out}} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}_0 \cdot \mathfrak{m}_1)$

1: $c^{(0)} \leftarrow \vec{c} \otimes \mathfrak{a}$
2: $c^{(1)} \leftarrow \vec{d} \otimes \mathfrak{b}$
3: $c_{\text{out}} \leftarrow c^{(0)} + c^{(1)}$
4: **return** c_{out}

Theorem A.2. *Let* $\mathsf{C} = \mathsf{RGSW}_{\mathsf{G}}(\mathfrak{m}^{(0)})$ *and* $c = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(1)}) = [\mathfrak{a}, \mathfrak{a} \cdot \mathfrak{s} + \mathfrak{m}^{(1)} + \mathfrak{e}]$. *Let* $\sigma^2_{\otimes}$ *be the noise variance induced by a gadget product. Then, Algorithm 5 on input* c, C *outputs a RLWE sample* $c_{\text{out}} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(0)} \cdot \mathfrak{m}^{(1)})$ *with variance*

$$\sigma^2_{\boxtimes} \leq 2 \cdot \sigma^2_{\otimes} + \mathsf{Var}(\mathfrak{m}^{(0)} \cdot \mathfrak{e})$$

Proof. It holds that $c^{(0)} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(-\mathfrak{a} \cdot \mathfrak{s} \cdot \mathfrak{m}^{(0)})$ and $c^{(1)} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{b} \cdot \mathfrak{m}^{(0)})$ with noise variance $\sigma^2_{\otimes}$. Then, it follows that

$$\begin{aligned} c^{(0)} + c^{(1)} &= \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}((\mathfrak{b} - \mathfrak{a}\mathfrak{s})\mathfrak{m}^{(0)}) \\ &= \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}((\mathfrak{m}^{(1)} + \mathfrak{e})\mathfrak{m}^{(0)}) \\ &\overset{!}{=} \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(0)} \cdot \mathfrak{m}^{(1)}), \end{aligned}$$

where we used the linear homomorphism of RLWE samples. The result follows immediately. □

Corollary A.3. *Let* $\mathsf{C} = \mathsf{RGSW}_{\mathsf{G}}(\mathfrak{m}^{(0)}), \mathfrak{m}^{(0)} \in \{0, 1\}$ *and let* $\sigma^2_{\otimes}$ *be the noise variance induced by a gadget product. Let* σ^2_c *be the noise variance of a RLWE sample* $c = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(1)})$. *Then, Algorithm 5 on input* c, C *outputs a RLWE sample* $c_{\text{out}} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(0)} \cdot \mathfrak{m}^{(1)})$ *with variance*

$$\sigma^2_{\boxtimes} \leq 2 \cdot \sigma^2_{\otimes} + \sigma^2_c$$

Corollary A.3 details the behaviour of the external product in the case the RGSW sample encrypts a binary value.

Based on the external product, we can define the homomorphic MUX gate in Algorithm 6, allowing us to select between two RLWE samples.

Algorithm 6 Homomorphic MUX Gate $\mathsf{CMUX} : \mathsf{RGSW}_{\mathsf{G}} \times \mathsf{RLWE} \times \mathsf{RLWE} \to \mathsf{RLWE}$

Input: RGSW sample $\mathsf{C} = \mathsf{RGSW}_{\mathsf{G}}(b), b \in \{0, 1\}$

Input: RLWE sample $c^{(0)} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(0)})$

Input: RLWE sample $c^{(1)} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(1)})$

Output: $c_{\text{out}} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(b)})$

1: $c^- \leftarrow c^{(0)} - c^{(1)}$
2: $c' \leftarrow \mathsf{C} \boxtimes c^-$
3: $c_{\text{out}} \leftarrow c' + c^{(1)}$
4: **return** c_{out}

Theorem A.4. *Let* $\mathsf{C} = \mathsf{RGSW}_{\mathsf{G}}(b)$ *and let* $\sigma^2_{\otimes}$ *be the noise variance induced by the gadget product. Furthermore, let* $c^{(i)} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(i)}) = [\mathfrak{a}^{(i)}, \mathfrak{a}^{(i)} \cdot \mathfrak{s} + \mathfrak{m}^{(i)} + \mathfrak{e}^{(i)}]$ *with noise variances* $\sigma^2_{(i)}$. *Then, Algorithm 6 outputs a RLWE sample* $c_{\text{out}} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m}^{(b)})$ *with noise variance*

$$\sigma^2_{\mathsf{MUX}} \leq 2 \cdot \sigma^2_{\otimes} + \sigma^2_{(i)}.$$

Proof. Through Theorem A.2, we look at the phase $\mathfrak{p}$ of $c' + c^{(1)} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{p})$:

$$\begin{aligned} \mathfrak{p} &= b \cdot (\mathfrak{m}^{(0)} + \mathfrak{e}^{(0)} - \mathfrak{m}^{(1)} - \mathfrak{e}^{(1)}) + \mathfrak{m}^{(1)} + \mathfrak{e}^{(1)} + \mathfrak{e} \\ &= b \cdot (\mathfrak{m}^{(0)} - \mathfrak{m}^{(1)}) + \mathfrak{m}^{(1)} + b \cdot (\mathfrak{e}^{(0)} - \mathfrak{e}^{(1)}) + \mathfrak{e}^{(1)} + \mathfrak{e} \end{aligned}$$

where $\mathfrak{e}$ is the noise term introduced by the message-independent gadget products. The result follows by inserting $b = 0$ or $b = 1$ into the last equation. □

The homomorphic (C)MUX gate is a crucial building block in the blind-rotation procedure used to refresh ciphertext in the TFHE scheme [33]. The blind-rotation procedure for binary keys is given in Algorithm 7 and its output characterized in Theorem A.5

Algorithm 7 CGGI Blind-Rotation BlindRotate : $\mathsf{LWE} \times \mathsf{RLWE} \times \mathsf{RGSW}_{\mathsf{G}}^{n} \to \mathsf{RLWE}$

Input: LWE sample $c = \mathsf{LWE}_{\vec{s}}^{n,q}(m,1) = [\vec{a}, b] \in \mathbb{Z}_{2N}^{n+1}$

Input: RLWE sample $\mathsf{acc} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p})$

Input: Blind-Rotation key $(\mathsf{evk}_i)_{i\in[n]}$, $\mathsf{evk}_i = \mathsf{RGSW}_{\mathsf{G}}(\vec{s}_i)$

Output: RLWE sample $\mathsf{acc}_{\mathsf{out}} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p} \cdot X^{-b+\langle \vec{a}, \vec{s}\rangle})$

1: $\mathsf{acc}_{\mathsf{out}} \leftarrow \mathsf{acc} \cdot X^{-b}$
2: **for** $i \leftarrow 0$ **to** $n-1$ **do**
3: $\quad \mathsf{acc}_{\mathsf{out}} \leftarrow \mathsf{CMUX}(\mathsf{evk}_i, \mathsf{acc}_{\mathsf{out}}, \mathsf{acc}_{\mathsf{out}} \cdot X^{\vec{a}_i})$
4: **return** $\mathsf{acc}_{\mathsf{out}}$

Theorem A.5. *Let $c = \mathsf{LWE}_{\vec{s}}^{n,q}(m,1) = [\vec{a}, b] \in \mathbb{Z}_{2N}^{n+1}$ with $\vec{s}_i \in \{0,1\}$ and $\mathsf{acc} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p}) = [\mathfrak{a}, \mathfrak{b}]$ be a RLWE sample with noise variance σ_{acc}^2. Furthermore, let $(\mathsf{evk}_i)_{i\in[n]}$ be a blind-rotation key such that $\mathsf{evk}_i = \mathsf{RGSW}_{\mathsf{G}}(\vec{s}_i)$ and let $\sigma_{\otimes}^2$ denote the noise variance induced by the gadget products. Then, Algorithm 7, on input c, acc and evk, outputs an $\mathsf{RLWE}_{\mathfrak{s}}^{N,Q}$ sample $\mathsf{acc}_{\mathsf{out}} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p} \cdot X^{-b+\langle \vec{a}, \vec{s}\rangle})$ with noise variance*

$$\sigma_{\mathsf{out}}^2 \le 2 \cdot n \cdot \sigma_{\otimes}^2 + \sigma_{\mathsf{acc}}^2$$

Proof. Correctness follows from the fact that the secret key $\vec{s}$ is binary and Theorem A.4. For the variance, we note that products by monomials of the form X^k do not affect the variance as they (negacyclically) rotate the error terms. Then, the output variance follows from Theorem A.4 and the fact that we apply Algorithm 6 n times. □

A.1.2 Key Switching. In this section, we recall common methods for changing the key of a given ciphertext, including the homomorphic evaluation of automorphisms on RLWE samples as an extension. We note that all instances of (R)LWE to (R)LWE can be unified as module LWE key switches, but we do not pursue this generalisation for simplicity. We first recall the LWE to LWE key-switching procedure in Algorithm 8 and describe its correctness in Theorem A.6.

Algorithm 8 LWE to LWE Key switch: $\mathsf{KeySwitch}_{\vec{s}\to\vec{S}}^{\mathsf{LWE}}$: $\mathsf{LWE} \times \mathsf{LWE}^{n\times\ell} \to \mathsf{LWE}$

Input: LWE Sample $c = \mathsf{LWE}_{\vec{s}}^{n,q}(m) = [\vec{a}, b]$

Input: Key switching key $(\mathsf{ksk}_{i,j})_{(i,j)\in[n]\times[\ell]}$ with $\mathsf{ksk}_{i,j} = \mathsf{LWE}_{\vec{S}}^{n,q}(\vec{s}_i \cdot L^j)$

Output: LWE Sample $c = \mathsf{LWE}_{\vec{S}}^{n,q}(m)$

1: $c_{\mathsf{out}} \leftarrow [\vec{0}, b]$
2: **for** $i \leftarrow 0$ **to** $n-1$ **do**
3: $\quad$ Write $\vec{a}_i = \sum_{j=0}^{l-1} L^j \tilde{a}_{i,j}$
4: $\quad c_{\mathsf{out}} \leftarrow c_{\mathsf{out}} - \sum_{j=0}^{l-1} \tilde{a}_{i,j} \cdot \mathsf{ksk}_{i,j}$
5: **return** c_{out}

Theorem A.6. *Let $c = \mathsf{LWE}_{\vec{s}}^{n,q}(m) = [\vec{a}, b] = [\vec{a}, \langle \vec{a}, \vec{s}\rangle + m + e]$ and let σ_c^2 be the variance of e. For a basis L, let $(\mathsf{ksk}_{i,j})_{(i,j)\in[n]\times[\ell]}$ be a key-switching key for a target key $\vec{S}$ with $\mathsf{ksk}_{i,j} = \mathsf{LWE}_{\vec{S}}^{n,q}(\vec{s}_i \cdot L^j)$ and LWE variance σ_{ksk}^2. Then, Algorithm 8 on input c and ksk outputs a LWE sample $c_{\mathsf{out}} = \mathsf{LWE}_{\vec{S}}^{n,q}(m)$ with noise variance*

$$\sigma_{\mathsf{out}}^2 \le \sigma_c^2 + n \cdot l \cdot \frac{L^2}{4} \cdot \sigma_{\mathsf{ksk}}^2.$$

Proof. We observe that the result of Algorithm 8 can be written as

$$\begin{aligned}
c_{\mathsf{out}} &= [\vec{0}, b] - \sum_{i=0}^{n-1}\sum_{j=0}^{\ell-1} \tilde{a}_{i,j} \cdot \mathsf{ksk}_{i,j} \\
&= [\vec{0}, \langle \vec{a}, \vec{s}\rangle + m + e] - \sum_{i=0}^{n-1}\sum_{j=0}^{\ell-1} \tilde{a}_{i,j} \cdot \mathsf{LWE}_{\vec{S}}^{n,q}(\vec{s}_i \cdot L^j) \\
&= [\vec{0}, \langle \vec{a}, \vec{s}\rangle + m + e] - \sum_{i=0}^{n-1} \mathsf{LWE}_{\vec{S}}^{n,q}(\sum_{j=0}^{\ell-1} \tilde{a}_{i,j} \cdot \vec{s}_i \cdot L^i) \\
&= [\vec{0}, \langle \vec{a}, \vec{s}\rangle + m + e] - \sum_{i=0}^{n-1} \mathsf{LWE}_{\vec{S}}^{n,q}(\vec{a}_i \cdot \vec{s}_i) \\
&= [\vec{0}, \langle \vec{a}, \vec{s}\rangle + m + e] - \mathsf{LWE}_{\vec{S}}^{n,q}(\langle \vec{a}, \vec{s}\rangle) \\
&\overset{!}{=} \mathsf{LWE}_{\vec{S}}^{n,q}(\langle \vec{a}, \vec{s}\rangle + m + e) - \mathsf{LWE}_{\vec{S}}^{n,q}(\langle \vec{a}, \vec{s}\rangle) \\
&= \mathsf{LWE}_{\vec{S}}^{n,q}(m + e) \\
&\overset{!}{=} \mathsf{LWE}_{\vec{S}}^{n,q}(m),
\end{aligned}$$

where we used the linear homomorphism of LWE samples. The result follows. □

Similarly, we describe the RLWE to RLWE key-switching and correctness in Algorithm 9 and Theorem A.7 respectively.

Algorithm 9 RLWE to RLWE Key Switching $\mathsf{KeySwitch}_{\mathfrak{s}\to\mathfrak{S}}^{\mathsf{RLWE}}$: $\mathsf{RLWE}' \times \mathsf{RLWE} \to \mathsf{RLWE}$

Input: Key switching key $\mathsf{ksk} = \mathsf{RLWE}'^{N,Q}_{\mathfrak{S}}(\mathfrak{s})$

Input: RLWE sample $c = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{m}) = [\mathfrak{a}, \mathfrak{b}]$

Output: RLWE sample $c_{\mathsf{out}} = \mathsf{RLWE}_{\mathfrak{S}}^{N,Q}(\mathfrak{m})$

1: $c_{\mathsf{out}} \leftarrow [\mathfrak{0}, \mathfrak{b}]$
2: $c_{\mathsf{out}} \leftarrow c_{\mathsf{out}} - \mathsf{ksk} \otimes \mathfrak{a}$
3: **return** c_{out}

Theorem A.7. *Let $c = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}\mathfrak{m}) = [\mathfrak{a}, \mathfrak{b}] = [\mathfrak{a}, \mathfrak{a}\mathfrak{s} + \mathfrak{m} + \mathfrak{e}]$ and let σ_c^2 be the variance of $\mathfrak{e}$. For a gadget G, let ksk be a key-switching key for a target key $\mathfrak{S}$ with $\mathsf{ksk} = \mathsf{RLWE}'^{N,Q}_{\mathfrak{S}}(\mathfrak{s})$ and let $\sigma_{\otimes}^2$ be the noise variance induced by the gadget product. Then, Algorithm 9 on input c and ksk outputs a RLWE sample $c_{\mathsf{out}} = \mathsf{RLWE}_{\mathfrak{S}}^{N,Q}(\mathfrak{m})$ with noise variance*

$$\sigma_{\mathsf{out}}^2 \le \sigma_c^2 + \sigma_{\otimes}^2.$$

Proof. We observe that the result of Algorithm 9 can be written as

$$\begin{aligned}
c_{\mathsf{out}} &= [\mathfrak{0}, \mathfrak{b}] - \mathsf{ksk} \otimes \mathfrak{a} \\
&= [\mathfrak{0}, \mathfrak{a}\mathfrak{s} + \mathfrak{m} + \mathfrak{e}] - \mathsf{RLWE}_{\mathfrak{S}}^{N,Q}(\mathfrak{a} \cdot \mathfrak{s}) \\
&\overset{!}{=} \mathsf{RLWE}_{\mathfrak{S}}^{N,Q}(\mathfrak{a}\mathfrak{s} + \mathfrak{m} + \mathfrak{e}) - \mathsf{RLWE}_{\mathfrak{S}}^{N,Q}(\mathfrak{a}\mathfrak{s}) \\
&\overset{!}{=} \mathsf{RLWE}_{\mathfrak{S}}^{N,Q}(\mathfrak{m})
\end{aligned}$$

where we used the linear homomorphism of RLWE samples. The result follows. □

While the RLWE key-switching procedure corresponds in essence to a gadget product, it will be crucial when we aim to evaluate automorphisms homomorphically. The complete procedure is given in Algorithm 10 and we establish its soundness in Theorem A.8. We note that if the output of Algorithm 10 is combined with (e.g. added to) the input RLWE sample, special care must be taken. In that instance, the independence heuristic may not hold, e.g. when $\psi(\mathfrak{e})_i = \mathfrak{e}_i$ for some i.

Algorithm 10 Automorphism Evaluation: $\mathsf{Aut} : \mathsf{RLWE} \times \mathrm{Gal}(\mathcal{K}/\mathbb{Q}) \times \mathsf{RLWE}' \to \mathsf{RLWE}$

Input: RLWE sample $c = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{m})$

Input: Automorphism $\psi \in \mathrm{Gal}(\mathcal{K}/\mathbb{Q})$

Input: Automorphism key $\mathsf{autk} = \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(\psi(\mathfrak{s}))$

Output: RLWE sample $c_{\mathsf{out}} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\psi(\mathfrak{m}))$

1: $c_{\mathsf{out}} \leftarrow [\mathbf{0}, \psi(\mathfrak{b})]$
2: $c_{\mathsf{out}} \leftarrow c_{\mathsf{out}} - \mathsf{autk} \otimes \psi(\mathfrak{a})$
3: **return** c_{out}

Theorem A.8. *Let* $c = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{m}) = [\mathfrak{a}, \mathfrak{b}] = [\mathfrak{a}, \mathfrak{a}\mathfrak{s} + \mathfrak{m} + \mathfrak{e}]$ *and let* σ_c^2 *be the variance of* $\mathfrak{e}$. *For a gadget* G, *let* autk *be an automorphism key for* $\psi \in Gal(\mathcal{K}/\mathbb{Q})$, *that is* $\mathsf{autk} = \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(\psi(\mathfrak{s}))$ *and let* $\sigma_\otimes^2$ *be the noise variance induced by the gadget product. Then, Algorithm 10 on input* c, ψ *and* autk *outputs a RLWE sample* $c_{\mathsf{out}} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\psi(\mathfrak{m}))$ *with noise variance*

$$\sigma_{\mathsf{out}}^2 \leq \sigma_c^2 + \sigma_\otimes^2$$

Proof. We observe that the result of Algorithm 10 can be written as

$$\begin{aligned} c_{\mathsf{out}} &= [\mathbf{0}, \psi(\mathfrak{b})] - \mathsf{autk} \otimes \psi(\mathfrak{a}) \\ &= [\mathbf{0}, \psi(\mathfrak{a})\psi(\mathfrak{s}) + \psi(\mathfrak{m}) + \psi(\mathfrak{e})] - \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\psi(\mathfrak{a}) \cdot \psi(\mathfrak{s})) \\ &\overset{!}{=} \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\psi(\mathfrak{a})\psi(\mathfrak{s}) + \psi(\mathfrak{m}) + \psi(\mathfrak{e})) - \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\psi(\mathfrak{a})\psi(\mathfrak{s})) \\ &\overset{!}{=} \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\psi(\mathfrak{m})), \end{aligned}$$

where we used the linear homomorphism of RLWE samples, the fact that ψ is an automorphism that does not influence norms i.e. $\mathsf{Var}(\psi(\mathfrak{e})) = \mathsf{Var}(\mathfrak{e})$. The result follows. □

We will see later on that the automorphism evaluation is particularly useful in order to isolate specific coefficients encoded in an RLWE sample. However, we shall show that RLWE key-switching can be extended to a notion of ring-switching as introduced in [25] and we adopt the notation from [27].

Let $\mathcal{R}_b = \mathbb{Z}_Q[X]/(X^{N_b} + 1), b \in \{0, 1\}$ and let $N_0 < N_1, N_0 | N_1$. Then, $\Pi : \mathcal{R}_0^{N_1/N_0} \to \mathcal{R}_1$ be the packing function defined as

$$\Pi(\mathfrak{a}^{(0)}, \mathfrak{a}^{(1)}, ..., \mathfrak{a}^{(N_1/N_0-1)}) = \sum_{i=0}^{N_1/N_0-1} X^i \cdot \sum_{j=0}^{N_0-1} \mathfrak{a}_i \cdot X^{j \cdot N_1/N_0}$$

In other words, the coefficients of the polynomials $\mathfrak{a}^{(i)} \in \mathcal{R}_0$ are encoded into $\mathcal{R}_1$ in strides of N_1/N_0 and Π induces an embedding of $\mathcal{R}_0$ into $\mathcal{R}_1$. Based on the packing function, we described the ring-switching algorithm below.

Algorithm 11 Ring Switch: $\mathsf{RingSwitch}_{N_1 \to N_0} : \mathsf{RLWE} \times \mathsf{RLWE}' \to \mathsf{RLWE}$

Input: $\mathsf{acc} = \mathsf{RLWE}_{\mathfrak{s}^{(1)}}^{N_1,Q}(\mathfrak{m}) = [\mathfrak{a}, \mathfrak{b}] \in \mathcal{R}_1^2$

Input: $(\mathsf{rswk})_{i \in [\ell]} = [\Pi(\mathfrak{a}^{(i,0)}, ..., \mathfrak{a}^{(i,N_1/N_0-1)}), \Pi(\mathfrak{a}^{(i,0)} \cdot \mathfrak{s}^{(0)} + \mathfrak{e}^{(i,0)}, ..., \mathfrak{a}^{(i,N_1/N_0-1)} \cdot \mathfrak{s}^{(0)} + \mathfrak{e}^{(i,N_1/N_0-1)}) + L^i \cdot \mathfrak{s}^{(1)}]$

Output: $\mathsf{acc}_{\mathsf{out}} = \mathsf{RLWE}_{\mathfrak{s}}^{N_0,Q}(\sum_{i=0}^{N_0-1} \mathfrak{m}_{i \cdot N_1/N_0}) \in \mathcal{R}_0^2$

1: Interpret $(\mathsf{rswk})_{i \in [n]}$ as RLWE' sample.
2: $[\mathfrak{a}', \mathfrak{b}'] \leftarrow [\mathbf{0}, \mathfrak{b}] - \mathsf{rswk} \otimes \mathfrak{a}$
3: Set $\mathsf{acc}_{\mathsf{out}} = \left[\sum_{i=0} \mathfrak{a}'_{i \cdot N_1/N_0} X^i, \sum_{i=0} \mathfrak{b}'_{i \cdot N_1/N_0} X^i\right]$
4: **Return** $\mathsf{acc}_{\mathsf{out}}$

Theorem A.9. *Let* $\mathsf{acc} = \mathsf{RLWE}_{\mathfrak{s}^{(1)}}^{N,Q}(\mathfrak{m}) \in \mathcal{R}_1^2$ *be a RLWE sample and let* $(\mathsf{swk})_{i[\ell]} = [\Pi(\mathfrak{a}^{(i,0)}, ..., \mathfrak{a}^{(i,N_1/N_0-1)}), \Pi(\mathfrak{a}^{(i,0)} \cdot \mathfrak{s}^{(0)} + \mathfrak{e}^{(i,0)}, ..., \mathfrak{a}^{(i,N_1/N_0-1)} \cdot \mathfrak{s}^{(0)} + \mathfrak{e}^{(i,N_1/N_0-1)}) + L^i \cdot \mathfrak{s}^{(1)}$ *be a ring-switching key. Then, Algorithm 11 outputs a RLWE sample* $\mathsf{acc}_{\mathsf{out}} = \mathsf{RLWE}_{\mathfrak{s}^{(0}}^{N,Q}(\sum_j \mathfrak{m}_{j \cdot N_1/N_0} X^j \in \mathcal{R}_0^2$ *, i.e. a subset of the coefficients of* $\mathfrak{m}$ *and in a smaller ring, and with a variance* σ_{out}^2 *with* $\sigma_{\mathsf{out}}^2 = \sigma_{\mathsf{out}}^2$.

Proof. First we note that rswitch can be easily interpreted as a RLWE' sample of $\mathfrak{s}^{(1)}$ under (replicated) $\mathfrak{s}^{(0)}$. Then, after applying the rlwe key-switch, we extract the corresponding coefficients, which are correct since $\mathfrak{s}^{(0)}$ had been packed previously. □

To close this subsection, we describe a special case of "switching" known as sample extraction. Given $\mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{m})$, the goal of sample extraction is to obtain $\mathsf{LWE}_{\vec{s}}^{n,q}(\mathfrak{m}_i)$ for some index $i \in \mathbb{Z}$ without decryption. We give the procedure in Algorithm 12 and describe its properties below.

Algorithm 12 Sample Extraction $\mathsf{SampleExtract} : \mathsf{RLWE} \to \mathsf{LWE}$

Input: $\mathsf{acc} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p})$

Input: Index $k \in \mathbb{Z}, k < N$

Output: $c_{\mathsf{out}} = \mathsf{LWE}_{\mathsf{Vec}[\mathfrak{s}]}^{n,q}(\mathfrak{p}_k, 1)$

1: $\tilde{\mathfrak{a}} \leftarrow \mathfrak{a} \cdot X^{-k}$
2: $\tilde{\mathfrak{b}} \leftarrow \mathfrak{b} \cdot X^{-k}$
3: $\vec{a} \leftarrow [\tilde{\mathfrak{a}}_0, -\tilde{\mathfrak{a}}_{N-1}, ..., -\tilde{\mathfrak{a}}_1]$
4: $c_{\mathsf{out}} \leftarrow [\vec{a}, \tilde{\mathfrak{b}}_0]$
5: **return** c_{out}

Theorem A.10. *Let* acc *be a RLWE sample of* $\mathfrak{p}$, *then on input* acc *and an index* k, *Algorithm 12 returns a* LWE *sample* $c_{\mathsf{out}} = \mathsf{LWE}_{\mathsf{Vec}[\mathfrak{s}]}^{n,q}(\mathfrak{p}_k)$ *with identical noise variance.*

Proof. First, we note that multiplying an RLWE sample by X^{-k} rotates the k-th term into the constant term of the polynomial. Then, since it can easily be shown that $\sum_i \vec{a}_i \mathsf{Vec}[\mathfrak{s}]_i = (\mathfrak{a} \cdot \mathfrak{s})_0$, soundness follows. Furthermore, the variance is unaffected since the b term in the output LWE corresponds to a coefficient of the input. □

A.1.3 Negacyclic Lookup Table Evaluation. In the following, we recall tools from prior works that enable the evaluation of specific lookup tables (LUTs). We start by defining negacyclic functions.

Definition (Negacyclic Function). Let $f : \mathbb{Z}_q \to \mathbb{Z}_Q$. If

$$f(x + \frac{q}{2}) = -f(x)$$

then f is called *negacylic*. Furthermore, if a LUT fulfils this property, we shall also call it negacylic. Next, Algorithm 13 describes how we can evaluate such functions homomorphically on an LWE sample, and Theorem A.11 describes its soundness properties.

Algorithm 13 Negacyclic LUT Evaluation: EvalNegLUT : $\mathsf{LWE} \times \mathsf{RGSW}^n \times \mathbb{Z}_{2N}^{\mathbb{Z}_Q} \to \mathsf{LWE}$

Input: LWE sample $c = \mathsf{LWE}_{\vec{s}}^{n,2N}(m, 1)$

Input: Blind-rotation key $(\mathsf{evk})_{i\in[n]}$ with $\mathsf{evk}_i = \mathsf{RGSW}(\vec{s}_i)$

Input: Negacyclic function $f : \mathbb{Z}_{2N} \to \mathbb{Z}_Q$

Output: LWE sample $c_{\text{out}} = \mathsf{LWE}_{\mathsf{Vec}[\mathfrak{s}]}^{n,q}(f(m+e))$

1: $\mathfrak{p} \leftarrow \sum_{i=0}^{N-1} f(i) \cdot X^i$
2: $\mathsf{acc} \leftarrow [\mathbf{0}, \mathfrak{p}]$
3: $\mathsf{acc} \leftarrow \mathsf{BlindRotate}(c, \mathsf{acc}, \mathsf{evk})$
4: $c_{\text{out}} \leftarrow \mathsf{SampleExtract}(\mathsf{acc}, 0)$
5: **return** c_{out}

Theorem A.11. *Let $c = \mathsf{LWE}_{\vec{s}}^{n,2N}(m) = [\vec{a}, \langle \vec{a}, \vec{s}\rangle + m + e]$ be an LWE sample and $f : \mathbb{Z}_{2N} \to \mathbb{Z}_Q$ be a negacyclic function. Let evk be a blind-rotation key $(\mathsf{evk})_{i\in[n]}$ with $\mathsf{evk}_i = \mathsf{RGSW}(\vec{s}_i)$ and let σ_{br}^2 denote the variance of blind-rotation applied on a noiseless accumulator (i.e. $\sigma_{\text{acc}}^2 = 0$ in Theorem A.5). Then, on input c, f and $(\mathsf{evk}_i)_{i\in[n]}$, Algorithm 13 outputs an LWE sample $c_{\text{out}} = \mathsf{LWE}_{\mathsf{Vec}[\mathfrak{s}]}^{n,q}(f(m+e), \mathsf{Vec}[\mathfrak{s}])$ with variance*

$$\sigma_{\text{out}}^2 = \sigma_{br}^2.$$

Proof. The output variance follows from Theorem A.5 and the fact that acc in Algorithm 13 is a noiseless RLWE sample, combined with Theorem A.10 that states that sample extraction does not affect variance. We prove correctness and inspect the phase of acc after blind-rotation:

$$\begin{aligned}\mathsf{acc} &= \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p} \cdot X^{-b+\langle\vec{a},\vec{s}\rangle}) \\ &= \mathsf{RLWE}\Bigg(\underbrace{\left(\sum_{i=0}^{N-1} f(i)\cdot X^i\right)\cdot X^{-(m+e)}}_{=:\mathfrak{q}}\Bigg)\end{aligned}$$

Assume that $m + e < N$. Then, it is easy to see that $\mathfrak{q}_0 = \mathfrak{m}_{m+e} = f(m+e)$. If instead we have that $m+e \geq N$, we write $m+e = N+w$ and observe that

$$\begin{aligned}\mathfrak{q} &= \left(\sum_{i=0}^{N-1} f(i)\cdot X^i\right)\cdot X^{-(m+e)} \\ &= \left(\sum_{i=0}^{N-1} f(i)\cdot X^i\right)\cdot X^{N-w} \\ &= \left(\sum_{i=0}^{N-1} f(i)\cdot X^i\right)\cdot -X^{-w} \\ \implies \mathfrak{q}_0 &= -f(w) = f(w+N) = f(m+e)\end{aligned}$$

where the last line follows from the fact that f is negacyclic w.r.t modulus $2N$. Finally, the statement follows from the correctness of the sample extraction. □

To conclude the discussion, we note that Algorithm 13 outputs an LWE sample w.r.t. to the linearized ring key $\mathsf{Vec}[\mathfrak{s}]$ but can be converted back to $\vec{s}$ using Algorithm 8.

A.1.4 Scheme Switching. Based on the previously described negacyclic LUT evaluation, we now discuss the LWEtoRGSW procedure in detail. First, we introduce the coefficient extraction procedure that allows us to isolate any coefficient of an encrypted polynomial. The procedure is given in Algorithm 14 and Theorem A.12 describes its properties.

Algorithm 14 Coefficient Extraction: ExtractCoef : $\mathsf{RLWE} \times \mathbb{Z} \times (\mathsf{RLWE}')^{\log_2(N)} \to \mathsf{RLWE}$

Input: RLWE sample $c = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p})$

Input: Index $j \in \mathbb{Z}$

Input: Automorphism keys $(\mathsf{autk}_i)_{(i-1)\in[\log_2(N)]}$ with $\mathsf{autk}_i = \mathsf{RLWE}_{\mathfrak{s}}'^{N,Q}(\mathfrak{s}(X^{2^i+1}))$

Output: RLWE Sample $c_{\text{out}} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p}_j)$

1: $c_{\text{out}} \leftarrow ((2N)^{-1} \pmod Q) \cdot c \cdot X^{-j}$
2: **for** $i \leftarrow \log_2(N)$ **to** 1 **do**
3: Write $c_{\text{out}} = [\mathfrak{a}, \mathfrak{b}]$
4: Let $\psi_{2^i+1} : \mathfrak{p}(X) \mapsto \mathfrak{p}(X^{2^i+1})$
5: $c_{\text{out}} \leftarrow c_{\text{out}} + \mathsf{Aut}(c_{\text{out}}, \psi_{2^i+1}, \mathsf{autk}_i)$
6: **return** c_{out}

Theorem A.12. *Let $c = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p})$ be a RLWE sample with noise variance σ_c^2, and $j \in \mathbb{Z}$ be an index. Let $(\mathsf{autk}_i)_{(i-1)\in[\log_2(N)]}$ be a set of automorphism keys for all $\psi_{2^i+1} : \mathfrak{p}(X) \mapsto \mathfrak{p}(X^{2^i+1})$ with $\mathsf{autk}_i = \mathsf{RLWE}(\mathfrak{s}(X^{2^i+1}))$ and such that σ_{autk}^2 denotes their noise variance. Then, on input $c, j, (\mathsf{autk}_i)_{(i-1)\in[\log_2(N)]}$, Algorithm 14 outputs a RLWE sample $c_{\text{out}} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p}_j)$ with variance σ_{cext}^2 such that*

$$\sigma_{cext}^2 \leq \sigma_c^2 + N^2 \cdot \sigma_{\mathsf{autk}}^2.$$

Proof. We note that for $i = 0$, ψ_{2^i+1} would not be an automorphism as $\gcd(2, 2N) \neq 1$ hence it is excluded from the set of automorphisms we consider. We leave the explanation of the multiplication by $(2N)^{-1}$ for the end, and note that in Line 1 the j-th coefficient is rotated to the constant coefficient.

Then, we establish correctness inductively and show that in iteration i, it holds that

$$c_{\text{out}} + \mathsf{Aut}(c_{\text{out}}, \psi_{2^i+1}, \mathsf{autk}_i) = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\mathfrak{p}^{(i)}),$$

with

$$\mathfrak{p}^{(i)} = \sum_{j=0}^{2^{i-1}-1} 2^{\log_2(N)-i+1} \cdot \mathfrak{p}_{2^{\log_2(N)-i+1}j} \cdot X^{2^{\log_2(N)-i+1}j}.$$

We start at $i = \log_2(N)$ and note that $\psi_{N+1} : \mathfrak{p}(X) \mapsto \mathfrak{p}(X^{N+1}) = \mathfrak{p}(-X)$, i.e. the sign in front of the polynomial coefficients with odd indices are flipped. Hence, we have that $\mathfrak{p}_j^{(i)} = \mathfrak{p}^{(i+1)} + (-1)^j \mathfrak{p}_j^{(i+1)}$

and the claims holds. Next, we write

$$\begin{aligned}\mathfrak{p}^{(i)} &= \mathfrak{p}^{(i+1)} + \psi_{2^i+1}(\mathfrak{p}^{(i+1)}) \\ &= \mathfrak{p}^{(i+1)} + \psi_{2^i+1}\left(\sum_{j=0}^{2^i-1} 2^{\log_2(N)-i}\mathfrak{p}_{2^{\log_2(N)-i}j} \cdot X^{2^{\log_2(N)-i+1}j}\right) \\ &= \mathfrak{p}^{(i+1)} + \sum_{j=0}^{2^i-1} 2^{\log_2(N)-i}\mathfrak{p}_{2^{\log_2(N)-i}j} \cdot X^{(2^i+1)\cdot 2^{\log_2(N)-i}j} \\ &= \mathfrak{p}^{(i+1)} + \sum_{j=0}^{2^i-1} 2^{\log_2(N)-i}\mathfrak{p}_{2^{\log_2(N)-i}j} \cdot (-1)^j \cdot X^{2^{\log_2(N)-i}j}\end{aligned}$$

From the last line, it is obvious that every second coefficient is cancelled out, proving the claim. Finally, we note that in the final iteration $i = 1$, we will have a factor of $2N$ in front of the constant (and only) coefficient. To correct this, we multiply the ciphertext by

$$((2N)^{-1} \pmod{Q})$$

before entering the loop. To prove the statement on the output variance, first we note that the soundness proof also applies to the input error term, i.e. the input error is not amplified. Next, at every iteration a new error term is introduced through the automorphism evaluation procedure, which is later on combined with itself. As mentioned earlier, the independence heuristic does not apply here and therefore at each iteration is previous noise is amplified by a factor of 4 as here $\mathsf{Var}[\mathfrak{e} + \psi(\mathfrak{e})] \leq 4\,\mathsf{Var}[\mathfrak{e}]$. The additive variance term then stems from the $\log_2(N)$ applications as $4^{\log_2(N)} = N^2$. □

Next, we come to the LWEtoRGSW procedure in Algorithm 15 and we discuss its soundness in Theorem A.13.

Algorithm 15 Scheme switching: $\mathsf{LWEtoRGSW} : \mathsf{LWE} \times \mathbb{Z} \times (\mathsf{RLWE}')^{\log_2(N)} \times (\mathsf{RLWE}') \times \mathsf{RGSW}^n \to \mathsf{RGSW}$

Input: LWE sample $c = \mathsf{LWE}^{n,2N}_{\vec{s}}(b, N)$
Input: Output RGSW basis L, digits $l = \lceil \log_L(Q) \rceil$
Input: Automorphism keys $(\mathsf{autk}_i)_{(i-1)\in[\log_2(N)]}$ with $\mathsf{autk}_i = \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(\mathfrak{s}(X^{2^i+1}))$
Input: Squaring key $\mathsf{sqk} = \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(\mathfrak{s}^2)$
Input: Blind-Rotation key $(\mathsf{evk}_i)_{i\in[n]}$ with $\mathsf{evk}_i = \mathsf{RGSW}(\vec{s}_i)$
Output: $\mathsf{C} = \mathsf{RGSW}(b)$

1: $c \leftarrow c + [\vec{0}, \frac{q}{4l}]$
2: $\mathfrak{p} \leftarrow \sum_{i=0}^{l-1} \frac{L^i}{2} \cdot \sum_{j=0}^{N/l-1} X^j$
3: $\mathsf{acc} \leftarrow [\mathbf{0}, \mathfrak{p}]$
4: $\mathsf{acc} \leftarrow \mathsf{BlindRotate}(c, \mathsf{acc}, \mathsf{evk})$
5: **for** $i \leftarrow 0$ **to** $l - 1$ **do**
6: $\quad \mathsf{acc}^{(i)} \leftarrow \mathsf{ExtractCoef}\left(\mathsf{acc}, i \cdot \frac{2N}{l}, (\mathsf{autk}_i)_{(i-1)\in[\log_2(N)]}\right)$
7: $\quad \mathsf{acc}^{(i)} \leftarrow \mathsf{acc}^{(i)} + [\mathbf{0}, \frac{L^i}{2}]$
8: $\quad$ Write $\mathsf{acc}^{(i)} = [\mathfrak{a}^{(i)}, \mathfrak{b}^{(i)}]$
9: $\quad \mathsf{acc}^{(*,i)} \leftarrow [\mathfrak{b}^{(i)}, \mathbf{0}] + \mathsf{sqk} \otimes \mathfrak{a}^{(i)}$
10: Set $\mathsf{C} = \left[[\mathsf{acc}^{(*,0)}, ..., \mathsf{acc}^{(*,l-1)}]^\top, [\mathsf{acc}^{(0)}, ..., \mathsf{acc}^{(l-1)}]^\top\right]$
11: **return** C

Theorem A.13. *Let $q = 2N, l = \lceil \log_L(Q) \rceil$ and $c = \mathsf{LWE}^{n,2N}_{\vec{s}}(b, N) = [\vec{a}, \langle \vec{a}, \vec{s} \rangle + Nb + e]$ with $|e| \leq \frac{N}{2l}$. Furthermore, let $(\mathsf{evk}_i)_{i\in[n]}$ be a blind-rotation key and $\mathsf{sqk} = \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(\mathfrak{s}^2)$ be a squaring key. Then, Algorithm 15 on input $c, L, \mathsf{sqk}, (\mathsf{evk}_i)_{i\in[n]}$ outputs a RGSW sample $\mathsf{C} = \mathsf{RGSW}_{\mathsf{G}_L}(b)$ with variance*

$$\sigma^2_C \leq N \cdot \frac{||\mathfrak{s}||_\infty}{4} \cdot (\sigma^2_{br} + \sigma^2_{cext}) + \sigma^2_{sq},$$

where $\sigma^2_{BR}, \sigma^2_{cext}, \sigma^2_{sq}$ are the noise variances introduced by blind-rotation, coefficient extraction and the RLWE-prime product with the squaring key.

Proof. We start by showing correctness. First, note that in Line 4, we have

$$\mathsf{acc} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\underbrace{(-1)^b X^{-\bar{e}} \cdot \mathfrak{p}}_{=:\mathfrak{q}}),$$

where $\bar{e} = e + \frac{q}{4l} = e + \frac{N}{2l} > 0$. Therefore, it follows that

$$\forall i < l : \mathfrak{q}_{i \cdot \frac{N}{l}} = (-1)^b \cdot \frac{L^i}{2}$$

and in Line 7, the relation

$$\mathsf{acc}^{(i)} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(b \cdot L^i)$$

holds by the correctness of Algorithm 14. Next, note that

$$\begin{aligned}\mathsf{acc}^{(*,i)} &= [\mathfrak{b}^{(i)}, \bar{5}] + \mathsf{sqk} \otimes \mathfrak{a}^{(i)} \\ &= [\mathfrak{a}^{(i)}\mathfrak{s} + b \cdot L^i + \mathfrak{e}^{(i)}, \bar{5}] + \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{s}^2 \cdot \mathfrak{a}^{(i)}) \\ &\overset{!}{=} \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(-\mathfrak{a}^{(i)}\mathfrak{s}^2 - b \cdot L^i \cdot \mathfrak{s} - \mathfrak{e}^{(i)} \cdot \mathfrak{s}) + \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{s}^2 \cdot \mathfrak{a}^{(i)}) \\ &\overset{!}{=} \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(-b \cdot L^i \cdot \mathfrak{s}),\end{aligned}$$

where $\mathfrak{e}^{(i)}$ is the noise polynomial of $\mathsf{acc}^{(i)}$. Finally, we have obtained for all i, $\mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(b \cdot L^i)$ and $\mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(-b \cdot L^i\mathfrak{s})$ from which we can assemble the desired RGSW sample. For the output variance we note that $\mathsf{acc}^{(i)}$ has a noise variance of

$$\sigma^2 \leq \sigma^2_{br} + \sigma^2_{ext}.$$

When $\mathsf{acc}^{(*,i)}$ is constructed, we first swap the role of $\mathfrak{b}^{(i)}$ leading to an implicit multiplication by $\mathfrak{s}$, the result of which is added to $\mathsf{sqk} \otimes \mathfrak{a}^{(i)}$ and the output variance follows. □

A.1.5 *Large Precision Bit Decomposition.*

In this section, we describe the approach for our large-precision bit decomposition algorithm. First, we recall the homomorphic floor function and large-precision digit-decomposition procedure from [72]. Briefly, these procedures are prefixed by "large-precision" as they operate on significantly larger ciphertext and plaintext modulus compared to conventional settings in which accumulator-based FHE schemes are used.

We start by introducing the modulus switching functionality in Algorithm 16 with a soundness discussion in Theorem A.14. In the context of fully homomorphic encryption, the modulus switching procedure is commonly combined with blind-rotation and key-switching to derive a bootstrapping procedure as the output variance of Algorithm 7 is independent of the input. The homomorphic floor function described in Algorithm 17 relies on this property to elegantly round the (large) plaintext to a multiple of a small

ciphertext modulus q. The soundness of the procedure is discussed in Theorem A.15.

Algorithm 16 Modulus Switch ModSwitch : LWE → LWE

Input: **(Implicit)** Input modulus Q

Input: **(Implicit)** Output modulus q

Input: LWE sample $c = \mathsf{LWE}_{\vec{s}}^{n,Q}(m, \frac{Q}{t}) = [\vec{a}^{(Q)}, b^{(Q)}]$ under modulus Q

Output: LWE sample $c_{\text{out}} = \mathsf{LWE}_{\vec{s}}^{n,q}(m, \frac{q}{t})$ under modulus q

1: $\vec{a}_q \leftarrow \left\lfloor \frac{q}{Q} \cdot \vec{a}^{(Q)} \right\rceil$ ▷ *Applied component-wise*
2: $b_q \leftarrow \left\lfloor \frac{q}{Q} \cdot b \right\rceil$
3: $c_{\text{out}} \leftarrow [\vec{a}_q, b_q]$
4: **return** c_{out}

Theorem A.14. *Let $c = \mathsf{LWE}_{\vec{s}}^{n,Q}(m, \frac{Q}{t}) = [\vec{a}^{(Q)}, b^{(Q)} = \langle \vec{a}^{(Q)}, \vec{s} \rangle + \frac{Q}{t} m + e]$ with error variance σ^2. Furthermore, we assume that $\vec{s} \in \{-1, 0, 1\}^n$. Then Algorithm 16 outputs a LWE sample $c_{\text{out}} = \mathsf{LWE}_{\vec{s}}^{n,q}(m, \frac{q}{t})$ with noise variance*

$$\sigma_{\text{out}}^2 \leq \left(\frac{q}{Q}\right)^2 \sigma^2 + \frac{||\vec{s}||_2^2 + 1}{12}. \tag{4}$$

Proof. We start by writing out the decryption function and multiplication by $\frac{q}{Q}$:

$$\begin{aligned} b^{(Q)} - \left\langle \vec{a}^{(Q)}, \vec{s} \right\rangle &= \frac{Q}{t} m + e + k \cdot Q \\ \implies \frac{q}{Q} \cdot \left(b^{(Q)} - \left\langle \vec{a}^{(Q)}, \vec{s} \right\rangle \right) &= \frac{q}{Q} \cdot \left(\frac{Q}{t} m + e + k \cdot Q \right) \\ &= \frac{q}{t} m + \frac{q}{Q} e + k \cdot q. \end{aligned}$$

In the last equation, the factor in front of the error term e explains the scaling in front of σ^2 in Equation 4, but it does not include the effect of rounding. As we assume that $\vec{a}^{(Q)}$ is distributed uniformly, so is the rounding error over $[-\frac{1}{2}, \frac{1}{2}]$. Then, $n+1$ such errors occur; 1 for $b^{(Q)}$ and n for $\vec{a}^{(Q)}$. As the latter are summed up, this leads to an error term following an Irwin-Hall distribution[3] of variance $\frac{n+1}{12}$. However, the errors are multiplied component-wise by the entries of $\vec{s}$, i.e. only occur whenever $\vec{s}_i \neq 0$ and lead to the additive term in Equation 4. □

Theorem A.15. *Let $c = \mathsf{LWE}_{\vec{s}}^{n,Q'}(m) = [\vec{a}, b = \langle \vec{a}, \vec{s} \rangle + m + e]$ such that $m = \tilde{m} \cdot q + \bar{m} \cdot \alpha$ and $q \mid Q'$. Furthermore, let β denote a bound on the absolute error in the output of negacyclic LUT evaluation, followed by key- and modulus-switching. If $|e| \leq \beta \leq \frac{\alpha}{4}$, then Algorithm 17, on input $c, F \in \{\top, \bot\}, (\mathsf{evk}_i)_{i \in [n]}, (\mathsf{ksk}_{i,j})_{(i,j) \in [n] \times [t]}$, returns a LWE sample $c_{\text{out}} = \mathsf{LWE}_{\vec{s}}^{n,Q'}(\tilde{m}, q)$ with noise term bounded by β. Furthermore, if $F = \top$, the remainder $c' = \mathsf{LWE}_{\vec{s}}^{n,Q'}(\bar{m}, \alpha)$ is also returned.*

[3] This distribution converges to a Gaussian for sufficiently large n and we will treat it as such

Algorithm 17 Homomorphic Floor: HomFloor : $\mathsf{LWE} \times \{\top, \bot\} \times \mathsf{RGSW}^n \times \mathsf{LWE}^{n \times \ell} \to \mathsf{LWE}$

Input: LWE sample $c = \mathsf{LWE}_{\vec{s}}^{n,Q'}(m) \pmod{Q'}$ with $m = \tilde{m} \cdot q + \bar{m} \cdot \alpha$, $q | Q'$

Input: Flag $b \in \{\top, \bot\}$

Input: Blind-rotation key $(\mathsf{evk}_i)_{i \in [n]}$ with $\mathsf{evk}_i = \mathsf{RGSW}(\vec{s}_i)$

Input: Key-Switching key $(\mathsf{ksk}_{i,j})_{(i,j) \in [n] \times [l]}$ with $\mathsf{ksk}_{i,j} = \mathsf{LWE}_{\vec{s}}^{n,q}(\mathsf{Vec}[\mathfrak{s}]_i \cdot L^j)$

Input: **(Implicit)** Error bound β

Output: $c_{\text{out}} = \mathsf{LWE}_{\vec{s}}^{n,Q'}(\tilde{m}, q) \pmod{Q'}$

1: Define $f_{\mathsf{MSB}}(x) = [x < q/2] \cdot \frac{Q}{Q'} \cdot \frac{q}{4}$
2: Define $f_{\mathsf{id}}(x) = \frac{Q}{Q'} x$
3: $c' \leftarrow c + [0, \beta] \pmod{q}$
4: $c_{\mathsf{MSB}} \leftarrow \mathsf{EvalNegLUT}(c', \mathsf{evk}, f_{\mathsf{MSB}})$
5: $c_{\mathsf{MSB}} \leftarrow \mathsf{KeySwitch}_{\vec{s} \to \mathsf{Vec}[\mathfrak{s}]}^{\mathsf{LWE}}(c_{\mathsf{MSB}}, \mathsf{ksk})$
6: $c_{\mathsf{MSB}} \leftarrow \mathsf{ModSwitch}_{Q \to Q'}(c_{\mathsf{MSB}})$
7: $c_1 \leftarrow c - c_{\mathsf{MSB}} - [0, \frac{q}{4}] + [0, \beta]$
8: $c_1' \leftarrow c_1 \pmod{q}$
9: $c_{\mathsf{id}} \leftarrow \mathsf{EvalNegLUT}(c_1', \mathsf{evk}, f_{\mathsf{id}})$
10: $c_{\mathsf{id}} \leftarrow \mathsf{KeySwitch}_{\vec{s} \to \mathsf{Vec}[\mathfrak{s}]}^{\mathsf{LWE}}(c_{\mathsf{id}}, \mathsf{ksk})$
11: $c_{\mathsf{id}} \leftarrow \mathsf{ModSwitch}_{Q \to Q'}(c_{\mathsf{MSB}})$
12: $c_{\text{out}} \leftarrow c_1 - c_{\mathsf{id}}$
13: **if** b **then**
14: **return** c_{out}, c'
15: **else**
16: **return** c_{out}

Proof. First, we note that the addition of β in Lines 3 and Line 7 are to ensure that the errors are negative and to avoid underflow effects, i.e. instead of $|e| \leq \beta$ we now have $0 \leq e + \beta \leq 2\beta$. Then, we can observe that c_1 in Line 7 is a LWE sample of $\tilde{m} \cdot q + \bar{m} \cdot \alpha - \frac{q}{2} \bar{m}_{\mathsf{MSB}}$ with an error term e_1 bounded by $e_1 \leq 4\beta$. Hence, c_1' is a LWE sample of $\bar{m} \cdot \alpha - \frac{q}{2} \bar{m}_{\mathsf{MSB}} \pmod{q}$ with the same error term, and we note that the most significant bit of c_1' is 0. Consequently, we are not restricted to negacyclic functions in the next application of Algorithm 13 using c_1' and recall that in Algorithm 13 the function is applied to the phase of the LWE sample, i.e. $m + e$ instead of just m. Therefore, when we then evaluate the identity function, the output corresponds to the message added to *the error* of c_1' and by extension c_1. Intuitively, the purpose of this step was to "rescale" c_1' to the same modulus as c_1. Consequently, c_{out} is an LWE sample of $\tilde{m} \cdot q$, as $\bar{m} \cdot \alpha$ and the original error of c_1 are cleared by subtraction of c_{id} and the remaining error term is bounded by β and the statement follows. □

Using Algorithm 17, it becomes easy to derive a digit-decomposition method which we describe in Algorithm 18 for which we briefly discuss its soundness in Theorem A.16.

Theorem A.16. *Let $c = \mathsf{LWE}_{\vec{s}}^{n,Q'}(m) = [\vec{a}, b = \langle \vec{a}, \vec{s} \rangle + m + e]$ such that $m = \tilde{m} \cdot q + \bar{m} \cdot \alpha$. Furthermore, let β denote a bound on the absolute error in the output of negacyclic LUT Evaluation followed by key- and modulus-switching. If $||\vec{s}||_2^2, \alpha, q$ are chosen such*

Algorithm 18 Large Precision Digit-Decomposition: LPDD : $\mathsf{LWE} \times \mathsf{RGSW}^n \times \mathsf{LWE}^{n\times\ell} \to (\mathsf{LWE})^{\log_{q/\alpha}(Q')}$

Input: LWE sample $c = \mathsf{LWE}_{\vec{s}}^{n,Q'}(m)$

Input: Blind-rotation key $(\mathsf{evk}_i)_{i\in[n]}$ with $\mathsf{evk}_i = \mathsf{RGSW}(\vec{s}_i)$

Input: Key-Switching key $(\mathsf{ksk}_{i,j})_{(i,j)\in[n]\times[\ell]}$ with $\mathsf{ksk}_{i,j} = \mathsf{LWE}_{\vec{s}}^{n,q}(\mathsf{Vec}[\mathfrak{s}]_i \cdot L^j)$

Output: LWE samples $c_{\text{out}}^{(i)} = \mathsf{LWE}_{\vec{s}}^{n,q}(m^{(i)}, \alpha)$ with $m = \sum_i m^{(i)} \cdot (q/\alpha)^i$

1: $i \leftarrow 0$
2: **while** $Q' > q$ **do**
3: $\quad c_{\lfloor\cdot\rfloor}, c_{\text{out}}^{(i)} \leftarrow \mathsf{HomFloor}(c, \top, (\mathsf{evk}_i)_{i\in[n]}, (\mathsf{ksk}_{i,j})_{(i,j)\in[n]\times[\ell]})$
4: $\quad c \leftarrow \mathsf{ModSwitch}_{Q'\to\alpha Q'/q}(c_{\lfloor\cdot\rfloor})$
5: $\quad Q' \leftarrow \alpha Q'/q$
6: $\quad i \leftarrow i+1$
7: **return** $(c_{\text{out}}^{(i)})_{i\in\log_{q/\alpha}(Q')}$

that the modulus switching noise in Line 4 is bounded by β with overwhelming probability and $|e| \le \beta \le \frac{\alpha}{4}$, then Algorithm 18, on input c, $(\mathsf{evk}_i)_{i\in[n]}$, $(\mathsf{ksk}_{i,j})_{(i,j)\in[n]\times[\ell]}$ outputs LWE samples of the digits of m in base q/α, with noise bounded by β.

Proof. By Theorem A.15, the first iteration will return the two ciphertexts, one corresponding to an encryption of $\alpha\cdot m$ rounded to the next multiple of q and another encoding the significant digit i.e. $\alpha\cdot m \pmod q$. Then, through our assumption regarding the modulus switching error, we maintain the loop invariant that the noise term is bounded by β after each modulus switch. Simultaneously, the modulus switch shifts the bits of the message by $\log_2(q/\alpha)$ or, in other words, moves the next digit in front of the α scaling factor: recall that $c_{\lfloor\cdot\rfloor} = \mathsf{LWE}_{\vec{s}}^{n,Q'}(q\cdot\left\lfloor\frac{m\alpha}{q}\right\rceil)$. Then at the end of the while loop, we have that $c = \mathsf{LWE}_{\vec{s}}^{n,\frac{\alpha Q'}{q}}(\frac{\alpha}{q}\cdot q\cdot\left\lfloor\frac{m\alpha}{q}\right\rceil) = \mathsf{LWE}_{\vec{s}}^{n,\frac{\alpha Q'}{q}}(\alpha\cdot\left\lfloor\frac{m\alpha}{q}\right\rceil) \pmod{\frac{\alpha Q'}{q}}$. The claim follows. □

The digit-decomposition procedure requires 2 blind rotations per digit in basis $\frac{q}{\alpha}$. As we desire a bit-decomposition, we aim for a basis $\frac{q}{\alpha} = 2$ implying that we require $2\cdot\log_{q/\alpha}(Q') = 2\cdot\log_2(Q')$; a significantly larger amount than for a basis e.g. equal to 16 or 32. To optimise the bit-decomposition, we will recall that a bit-decomposition for an LWE ciphertext modulo $q = 2N$ in fact requires only a single blind-rotation, which we show in Algorithm 19 with corresponding proof in Theorem A.17. We finally perform the large-precision bit-decomposition by first applying Algorithm 18 using a large basis, and then perform a decomposition using the specialized bit-decomposition procedure on each previously obtained digit. Algorithm 19 gives the bit-decomposition procedure and we prove its correctness in Theorem A.17.

Theorem A.17. *Let $c = \mathsf{LWE}_{\vec{s}}^{n,q}(m,\alpha) = [\vec{a}, b = \langle\vec{a},\vec{s}\rangle + \alpha\cdot m + e]$ for $q = 2N$ and $\vec{s}\in\{0,1\}$. Furthermore, let $(\mathsf{evk}_i)_{i\in[n]}$ be a blind-rotation key i.e. $\mathsf{evk}_i = \mathsf{RGSW}(\vec{s}_i)$, and $(\mathsf{ksk}_{i,j})_{(i,j)\in[N]\times[\ell]}$ a key-switching key $\mathsf{ksk}_{i,j} = \mathsf{LWE}_{\vec{s}}^{N,Q}(\mathsf{Vec}[\mathfrak{s}]_i\cdot L^j)$. If $|e| \le \beta \le \frac{\alpha}{4}$, then*

Algorithm 19 Bit-Decomposition BitDecomp : $\mathsf{LWE}\times\mathsf{RGSW}^n\times\mathsf{LWE}^{n\times\ell}\times\mathbb{Z} \to \mathsf{LWE}^{\log_2 q/\alpha}$

Input: LWE sample $c = \mathsf{LWE}_{\vec{s}}^{n,q}(m,\alpha)$, $\alpha = \frac{q}{t}$

Input: Blind-rotation key $(\mathsf{evk}_i)_{i\in[n]}$ with $\mathsf{evk}_i = \mathsf{RGSW}(\vec{s}_i)$

Input: Key-Switching key $(\mathsf{ksk}_{i,j})_{(i,j)\in[N]\times[\ell]}$ with $\mathsf{ksk}_{i,j} = \mathsf{LWE}_{\vec{s}}^{N,Q}(\mathsf{Vec}[\mathfrak{s}]_i\cdot L^j)$

Input: Output Modulus q'

Output: LWE samples $(c_{\text{out},i})_{i\in[\log_2(q/\alpha)]}$ with $c_{\text{out}}^{(i)} = \mathsf{LWE}_{\vec{s}}^{n,q'}(b_i, \frac{q'}{2})$

1: $c \leftarrow c + [\vec{0}, \frac{q}{2\alpha}]$
2: Set $\mathfrak{p} \leftarrow \sum_{i=0}^{2N/\alpha-1} X^i$
3: $\mathsf{acc} \leftarrow [\mathbf{0}, -\frac{Q}{4}\mathfrak{p}]$
4: $\mathsf{acc} \leftarrow \mathsf{BlindRotate}(\mathsf{acc}, c, \mathsf{evk})$
5: $\mathfrak{b}^{(0)} \leftarrow (\sum_{i=0}^{N-1} X^i)/\mathfrak{p}$
6: $\mathsf{acc}^{(0)} \leftarrow \mathsf{acc}\cdot\mathfrak{b}^{(0)} + [\mathbf{0}, \frac{Q}{4}]$
7: $\mathsf{acc} \leftarrow 2\cdot\mathsf{acc}$
8: **for** $i \leftarrow 1$ **to** $\log_2(q/\alpha) - 1$ **do**
9: $\quad \mathfrak{b}^{(i)} \leftarrow \sum_{j=0}^{i} X^{\frac{N}{2^i}\cdot j}$
10: $\quad \mathsf{acc}^{(i)} \leftarrow \mathsf{acc}\cdot\mathfrak{b}^{(i)}$
11: **for** $i \leftarrow 0$ **to** $\log_2(q/\alpha) - 1$ **do**
12: $\quad c' \leftarrow \mathsf{SampleExtract}(\mathsf{acc}^{(i)}, 0)$
13: $\quad c' \leftarrow \mathsf{KeySwitch}_{\vec{s}\to\mathsf{Vec}[\mathfrak{s}]}^{\mathsf{LWE}}(c', \mathsf{ksk})$
14: $\quad c_{\text{out},i} \leftarrow \mathsf{ModSwitch}_{Q\to q'}(c')$
15: **return** $(c_{\text{out},i})_{i\in[\log_2(q/\alpha)]}$

Algorithm 19, on input c, $(\mathsf{evk}_i)_{i\in[n]}$, $(\mathsf{ksk}_{i,j})_{(i,j)\in[n]\times[\ell]}$ and a target output modulus q', returns LWE samples $c_{\text{out},i} = \mathsf{LWE}_{\vec{s}}^{n,q'}(b_i, \frac{q'}{2})$ such that $m = \sum_{i=0} b_i 2^i$ and with variance

$$\sigma_{bd}^2 \le \left(\frac{Q}{q'}\right)^2\left(N\cdot\sigma_{br}^2 + \sigma_{ks}^2\right) + \sigma_{ms}^2.$$

Proof. We start by showing that $\mathsf{acc}^{(i)} = \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}(\frac{Q}{2}b_i + \mathfrak{r}^{(i)})$ for some homogeneous polynomial $\mathfrak{r}^{(i)}$. We note that after Line 4 it holds that

$$\begin{aligned}\mathsf{acc} &= \mathsf{RLWE}\left(-\frac{Q}{4}\cdot\mathfrak{p}\cdot X^{-\frac{2N}{t}m-(e+\frac{q}{2\alpha})}\right)\\ &= \mathsf{RLWE}\left(-\frac{Q}{4}\cdot\mathfrak{p}\cdot X^{Nb_0-\frac{2N}{t}\tilde{m}-\tilde{e}}\right),\end{aligned}$$

where $\tilde{e} = (e+\frac{q}{2\alpha})$ with $\tilde{e}\ge 0$ by the assumption on e (interpreted over $\mathbb{Z}$). For the case $i = 0$, by definition of $\mathfrak{b}^{(0)}$, it then follows that

$$\begin{aligned}\mathsf{acc}\cdot\mathfrak{b}^{(0)} &= \mathsf{RLWE}_{\mathfrak{s}}^{N,Q}\left(-\frac{Q}{4}\cdot\sum_{i=0}^{N-1}X^i\cdot X^{Nb_0-\frac{2N}{t}\tilde{m}-\tilde{e}}\right)\\ &= \mathsf{RLWE}\left(-\frac{Q}{4}\cdot(\sum_{i=0}^{N-1}X^i)\cdot(-1)^{b_0}\cdot X^{-\frac{2N}{t}\tilde{m}-\tilde{e}}\right).\end{aligned}$$

Since $\tilde{m} < \frac{t}{2}$, it holds that $\frac{2N}{t}\tilde{m} + \tilde{e} < N$ and therefore

$$\begin{aligned}\mathsf{acc}\cdot\mathfrak{b}^{(0)} &= \mathsf{RLWE}\left((-1)^{1-b_0}\cdot\frac{Q}{4} + \mathfrak{r}^{(0)}\right)\\ \implies \mathsf{acc}\cdot\mathfrak{b}^{(0)} + \left[\mathbf{0}, \frac{Q}{4}\right] &= \mathsf{RLWE}\left(b_0\cdot\frac{Q}{2} + \mathfrak{r}^{(0)}\right).\end{aligned}$$

For $i > 0$, from Line 7 it holds that

$$\begin{aligned}\mathsf{acc} &= 2 \cdot \mathsf{RLWE}\left(-\frac{Q}{4} \cdot \mathfrak{p} \cdot X^{-\frac{2N}{t}m-\tilde{e}}\right) \\ &= \mathsf{RLWE}\left(\frac{Q}{2} \cdot \mathfrak{p} \cdot X^{-\frac{2N}{t}\tilde{m}-\tilde{e}}\right).\end{aligned}$$

We note that the negation disappeared. This stems from the fact that $2 \cdot \pm\frac{Q}{4} = \frac{Q}{2} \pmod{Q}$ if Q is even. In the case where Q is odd, it is true that $-\frac{Q}{2}$ and $\frac{Q}{2}$ differ by an "error" of 1 which we move into the error term. Likewise, since the most significant bit of m would induces a sign-flip (c.f. case $i = 0$) we can now simply replace m by $\tilde{m}$ in the exponent. Then, note that $(\mathfrak{b}^{(i)})_j = 1$ iff. the i-th most significant bit of j is 1. Then, for any i we have that

$$\mathsf{acc}^{(i)} = \mathsf{RLWE}\left(\frac{Q}{2} \cdot \mathfrak{b}^{(i)} \cdot \mathfrak{p} \cdot X^{-\frac{2N}{t}\tilde{m}-\tilde{e}}\right).$$

For simplicity, first assume $\tilde{e} = 0, \mathfrak{p} = 1$, then it is easy to see that the constant term of $\frac{Q}{2}\mathfrak{b}^{(i)} \cdot X^{-\frac{2N}{t}\tilde{m}-\tilde{e}}$ if $\frac{Q}{2}b_i$. However, since $\tilde{e} \neq 0$ is general, we must encode values into blocks justifying the choice $\mathfrak{p} = \sum_{i=0}^{2N/\alpha-1} X^i$. Finally, soundness follows from the correctness of sample-extraction, key-switching and modulus switching. Regarding the output variance, we note that for any i, $||\mathfrak{b}^{(i)}||_0 \leq N$ and $||\mathfrak{b}_\infty^{(i)} \leq 1$. The output variance then follows from Theorem A.5, Theorem A.10, Theorem A.6 and Theorem A.14. □

We can now give the large-precision bit-decomposition procedure in Algorithm 20 and corresponding discussion in Theorem A.18.

Algorithm 20 Large Precision Bit-Decomposition: LPBD : $\mathsf{LWE} \times \mathsf{RGSW}^{2\times n} \times \mathsf{LWE}^{2\times n\times \ell} \to (\mathsf{LWE})^{\log_2(Q'/\alpha)}$

Input: LWE sample $c = \mathsf{LWE}^{n,Q'}_{\vec{s}^{(0)}}(m, \alpha)$

Input: Blind-rotation key $(\mathsf{evk}^{(i)}_j)_{j\in[n]}, i \in \{0, 1\}$ with $\mathsf{evk}^{(i)}_j = \mathsf{RGSW}(\vec{s}^{(0)}_i; \mathfrak{s}^{(i)})$

Input: Key-Switching key $(\mathsf{ksk}^{(i)}_{j,k})_{(j,k)\in[N]\times[\ell]}, i \in \{0, 1\}$ with $\mathsf{ksk}^{(i)}_{j,k} = \mathsf{LWE}^{N,Q}_{\vec{s}^{(i)}}(\mathsf{Vec}[\mathfrak{s}^{(i)}]_j \cdot L^k, 1)$

Input: Output modulus q'

Output: LWE samples $c^{(i)}_{\mathsf{out}} = \mathsf{LWE}^{n,q'}_{\vec{s}}(b_i, \frac{q'}{2})$ with $m = \sum_i b_i \cdot 2^i$

1: $(c_i)_{i\in[\log_\alpha(Q')]} \leftarrow \mathsf{LPDD}(c, \mathsf{evk}^{(0)}, \mathsf{ksk}^{(0)})$
2: **for** $i \leftarrow 0$ **to** $\log_\alpha(Q') - 1$ **do**
3: $\quad (c'_{i,j})_{j\in[\log_2(q/\alpha)]} \leftarrow \mathsf{BitDecomp}(c_i, \mathsf{evk}^{(1)}, \mathsf{ksk}^{(1)}, q')$
4: Set $c_{\mathsf{out},2^i+j} = c'_{i,j}$
5: **return** $(c_{\mathsf{out},i})_{i\in[\log_2(Q'/\alpha)]}$

Theorem A.18. *Let $c = \mathsf{LWE}^{n,q}_{\vec{s}}(m \cdot \alpha)$ and $\mathsf{evk}^{(0)}, \mathsf{ksk}^{(0)}$ respectively be blind-rotation and key-switching keys such that*

- *The correctness criteria from Theorem A.16 hold.*
- *For the output modulus $q^{(0)}$ of Algorithm 18 it holds that $q^{(0)} = 2N^{(1)}$ where $N^{(1)}$ is the ring dimension of $\mathsf{evk}^{(1)}$.*
- *The digits returned by Algorithm 18 fulfil the requirements for inputs to Algorithm 19 as outlined in Theorem A.17.*

Furthermore, let $\mathsf{evk}^{(1)}, \mathsf{ksk}^{(1)}$ be blind-rotation and key-switching keys, and q' be an output modulus. Then, on input $c, \mathsf{evk}^{(0)}, \mathsf{evk}^{(1)}, \mathsf{ksk}^{(0)}, \mathsf{ksk}^{(1)}, q'$, Algorithm 20 outputs ciphertexts $c_{\mathsf{out},i} = \mathsf{LWE}^{n,q'}_{\vec{s}^{(1)}}(b_i \cdot \frac{q'}{2})$ with variance

$$\sigma^2_{lpbd} = \sigma^2_{bd}.$$

Proof. We note that under the stated requirements of the theorem, correctness follows immediately as we simply chain Algorithm 18 and Algorithm 19. Furthermore, the output ciphertexts are simply the outputs of Algorithm 19 hence the variance claim follows. □

For the sake of completeness, we note that, although we require different blind-rotation keys in Algorithm 20, is it strictly speaking not necessary but reflects possible optimizations achieved by selecting different parameters.

A.1.6 Selector Construction. We have now defined the procedures necessary to construct the RGSW samples required to perform Phase 2 and Phase 3 in Respire/Pirouette: we first use Algorithm 20 and apply Algorithm 15 on the results. In this part, we discuss how to obtain the selectors required for Phase 1. The algorithm to construct the corresponding selectors is given in Algorithm 21 and we prove its correctness in Theorem A.19

Algorithm 21 Phase 1 Selector Construction: SelCon : $(\mathsf{RGSW})^{\log_2(\nu_1)} \to (\mathsf{RLWE}')^N$

Input: RGSW samples $(\mathsf{C}^{(i)})_{i\in[\nu_1]}$ $\mathsf{C}^{(i)} = \mathsf{RGSW}(b_i)$

Output: $(\mathsf{acc}^{(j)}_{\mathsf{out}})_{j\in[2^{\nu_1}]} = \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}([j = \sum_i b_i 2^i])$

1: $\mathbf{1} \leftarrow \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(1)$ ▷ *Can be constructed without knowing $\mathfrak{s}$*
2: $\mathcal{S}^{(0)} \leftarrow [\mathbf{1}]$
3: **for** $i \leftarrow 0$ **to** $\nu_1 - 1$ **do**
4: $\quad \mathcal{S}^{(i+1)} \leftarrow [\,]$
5: $\quad$ **for** $j \leftarrow 0$ **to** $|\mathcal{S}^{(i)}| - 1$ **do**
6: $\quad\quad c \leftarrow \mathcal{S}^{(i)}[j]$
7: $\quad\quad c^R \leftarrow \mathsf{C}^{(\nu_1-i-1)} \boxtimes c$ ▷ $\boxtimes$ *applied component-wise*
8: $\quad\quad c^L \leftarrow \mathbf{1} - c^R$
9: $\quad\quad \mathcal{S}^{(i+1)} \leftarrow \mathcal{S}^{(i+1)}$ concatenated with $[c^L, c^R]$
10: $(\mathsf{acc}^{(j)}_{\mathsf{out}})_{j\in[N]} \leftarrow \mathcal{S}^{(\nu_1)}[j]$
11: **return** $(\mathsf{acc}^{(j)}_{\mathsf{out}})_{i\in[2^{\nu_1}]}$

Theorem A.19. *Let $\mathsf{C}^{(i)} = \mathsf{RGSW}(b_i), i \in [\nu_1]$ be a set of RGSW samples of bits. Then on input $\mathsf{C}^{(i)}$, Algorithm 21 outputs $\mathsf{acc}^{(j)}_{\mathsf{out}} = \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}([j = \sum_i b_i 2^i])$ with variance*

$$\sigma^2_{selcon} \leq \nu_1 \cdot \sigma^2_{\boxtimes}$$

where $\sigma^2_{\boxtimes}$ is the noise variance induced by the external product (i.e. Algorithm 5)

Proof. We show correctness by proving the loop invariant

$$\mathcal{S}^{(i+1)}[j] = \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}\left(\left[j = \sum_{k=0}^{i} b_k 2^k\right]\right)$$

We proceed inductively. In the case $i = 0$, we have that

$$\begin{aligned}\mathcal{S}^{(1)} &= [c^L, c^R] \\ &= [\mathbf{1} - C^{(\nu_1-1)} \boxtimes c, C^{(\nu_1-1)} \boxtimes c] \\ &= [\mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(1 - b_0 \cdot 1), \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}(b_0 \cdot 1).]\end{aligned}$$

and the claim holds. For $i > 0$, fix any j and note that the inner loop constructs:

$$\begin{aligned}&\mathcal{S}^{(i+1)}[2j], \mathcal{S}^{(i+1)}[2j+1] \\ &= [c^L, c^R] \\ &= [\mathbf{1} - \mathsf{C}^{(\nu_1-1-i)} \boxtimes \mathcal{S}^{(i)}[j], \mathsf{C}^{(\nu_1-1-i)} \boxtimes \mathcal{S}^{(i)}[j]] \\ &= [\mathbf{1} - \mathsf{C}^{(\nu_1-1-i)} \boxtimes \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}\left(\left[j = \sum_{k=0}^{i-1} b_k 2^k\right]\right), \mathsf{C}^{(\nu_1-1-i)} \boxtimes \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}\left(\left[j = \sum_{k=0}^{i-1} b_k 2^k\right]\right)] \\ &= [\mathbf{1} - \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}\left(b_i \cdot \left[j = \sum_{k=0}^{i-1} b_k 2^k\right]\right), \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}\left(b_i \cdot \left[j = \sum_{k=0}^{i-1} b_k 2^k\right]\right)] \\ &= [\mathbf{1} - \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}\left(\left[2j+1 = \sum_{k=0}^{i-1} b_k 2^k\right]\right), \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}\left(\left[2j+1 = \sum_{k=0}^{i} b_k 2^k\right]\right)] \\ &= [\mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}\left(\left[2j = \sum_{k=0}^{} b_k 2^k\right]\right), \mathsf{RLWE}'^{N,Q}_{\mathfrak{s}}\left(\left[2j+1 = \sum_{k=0}^{i} b_k 2^k\right]\right)],\end{aligned}$$

where in the last step we used the fact that the least significant bit of $2j + 1$ is 1, and correctness holds. For the output variance, we note that every leaf is the result of ν_1 external products using RGSW samples of bits and the result follows. □

A.1.7 Pirouette. To complete the section on correctness we write out the Pirouette routine in full. For the sake of completeness, we

Algorithm 22 Pirouette

Input: $c = \mathsf{LWE}^{n,q}_{\vec{s}}(m, \alpha)$

Input: Database $(\mathrm{db}_{i,j})_{(i,j)\in[2_1^{\nu}]\times[2^{\nu_2}]} \in \mathcal{R}, i \in [2^{\nu_1}], j \in [2^{\nu_2}]$

Input: Key material km_{dig} for high-precision bit decomposition.

Input: Key material km_{switch} for LWEtoRGSW

Input: Ring switching key rswk including compression moduli $Q_1 \in \mathbb{Z}$

Output: $\mathsf{acc}_{\mathsf{out}} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{m})$

$(\mathsf{ct}_i)_{i\in[\nu]} \leftarrow \mathsf{LPBD}(c, \mathsf{km}_{dig})$
$(\mathsf{C}_i)_{i\in[\mathcal{N}]} \leftarrow \mathsf{LWEtoRGSW}((\mathsf{ct}_i)_{i\in[nu]}, \mathsf{km}_{switch})$
$(\mathsf{sel}_j)_{j\in[2^{\nu_1}]} \leftarrow \mathsf{SelCon}((C_i)_{i\in[\nu_1]})$
$\forall j : \mathsf{acc}_{-1,j} \leftarrow \sum_i \mathsf{sel}_i \otimes \mathrm{db}_{i,j}$
for $i = 0$ to $i = \nu_2 - 1$ **do**
  for $j = 0$ to $j = 2^{\nu_2-i-1} - 1$ **do**
    $\mathsf{acc}_{i,j} \leftarrow \mathsf{CMUX}(\mathsf{C}^{\nu_1+i-1}, \mathsf{acc}_{i-1,j}, \mathsf{acc}_{i-1,j+2^{\nu_2-i-1}})$
$\mathsf{acc}' \leftarrow \mathsf{acc}_{\nu_2-1,0}$
for $i = 0$ to $i = \nu_3 - 1$ **do**
  $\mathsf{acc}' \leftarrow \mathsf{CMUX}(\mathsf{C}^{\nu_1+\nu_2+i-1}, \mathsf{acc}', \mathsf{acc}' \cdot X^{\nu_3-1-i})$
$\mathsf{acc}'_1 \leftarrow \mathsf{ModSwitch}_{Q\to Q_1}(\mathsf{acc}')$
$\mathsf{acc}_{\mathsf{out},Q_1} \leftarrow \mathsf{RingSwitch}(\mathsf{acc}'_1, (\mathsf{sel}_j)_{j\in[\ell]}$
return $\mathsf{acc}_{\mathsf{out}}$

note that in Respire [27], an additional modulus switch is applied on the right-hand side of the result RLWE sample. The authors refer to it as *split-modulus* switching. This technique has previously been introduced in [24] as *ciphertext shrinking*, utilized in order to compress ciphertexts. As this step, if performed properly, does not affect correctness we refer the reader to the article.

Theorem A.20. *Let $c = \mathsf{LWE}^{n',Q'}_{\vec{s}'}(m, \alpha)$ be an LWE sample of the index. Furthermore, let km_{dig} be the key-material for high-precision bit decomposition, km_{switch} be key material for* LWEtoRGSW, rswk *a ring-switching key and $Q_1 \in \mathbb{Z}$ be two compression moduli. Then, Algorithm 22, on input $c, \mathsf{km}_{dig}, \mathsf{km}_{switch}, \mathsf{rswk}, Q_1$, returns a RLWE sample encoding the record $\mathsf{acc}_{\mathsf{out}} = \mathsf{RLWE}^{N,Q}_{\mathfrak{s}}(\mathfrak{r}_m)$ and with variance*

$$\sigma_{\mathsf{out}} \leq \frac{Q_1^2}{Q^2}\left(2^{\nu_1} l_p L_p^2 \sigma_{sel-con} + 2^{\nu_2+\nu_3}\sigma^2_{switch}\right) + \sigma^2_{ms1} + \sigma^2_{rswitch}$$

where

- *L_p, ℓ_p are the* RLWE′ *selector basis and digits.*
- *$\sigma^2_{sel-con}, \sigma^2_{switch}$ are the variances of selector construction and scheme switching respectively. Furthermore, recall $\sigma^2_{sel-con} \leq \nu_1 \cdot \sigma^2_{switch}$.*
- *σ^2_{ms} is the modulus switching variance.*
- *$\sigma^2_{rswitch}$ is the variance induced by ring-switching.*

Proof. Correctness follows from the correctness of the subroutines and Respire. The output variance follows from the application of Theorem A.1, Theorem A.19, Theorem A.14, Theorem A.9 □

A.2 Security analysis of Pirouette

Circular security assumes that CPA security holds even when encryptions of (functions of) the secret key are provided. This assumption is essential for FHE schemes [23, 26, 35, 47, 48, 53] that support key switching or bootstrapping, as well as for FHE-based applications, including the previous PIR schemes [1, 4, 6, 27, 44, 50, 58, 59, 69, 73, 79, 80, 82, 88]. In [79], circular security is referred to as key-dependent message (KDM) security. We adopt this terminology and present their definition as adapted from [17].

Definition A.21 (KDM Security [17, 79]). Let $\Sigma = (\mathsf{KeyGen}, \mathsf{Encrypt})$ be an encryption scheme with message space $\mathcal{M}$. Let $\mathcal{F}$ be a set of functions from $\mathcal{M}$ to $\mathcal{M}$. Then Σ satisfies $\mathcal{F}$-key-dependent message security ($\mathcal{F}$-KDM security) if for any probabilistic polynomial-time adversary $\mathcal{A}$, it holds that

$$\left|\Pr[\mathcal{A}^{\mathcal{O}_b(\mathsf{sk},\cdot)}(1^\lambda) = b] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda),$$

where $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, $b \xleftarrow{\$} \{0, 1\}$, and on input $f \in \mathcal{F}$, the oracle $\mathcal{O}_b$ samples $r \xleftarrow{\$} \mathcal{M}$ and responds with $\mathsf{Encrypt}(\mathsf{sk}, f(\mathsf{sk}))$ if $b = 0$ and $\mathsf{Encrypt}(\mathsf{sk}, r)$ if $b = 1$.

For Pirouette security, we consider the following families of functions over $\mathbb{Z}_q$ and $\mathcal{R}_Q$.

- Let $\mathcal{F}_{scal} = \{r \mapsto \alpha \cdot r : \alpha \in \mathbb{Z}_q\}$ denote the family of scaling functions over $\mathbb{Z}_q$.
- Let $\mathcal{F}_{quad} = \{r \mapsto \alpha_0 + \alpha_1 r + \alpha_2 r^2 : \alpha_0, \alpha_1, \alpha_2 \in \mathbb{Z}_Q\}$ denote the family of quadratic functions over $\mathcal{R}_Q$.
- Let $\mathcal{F}_{aut} = \{r \mapsto k \cdot \sigma_i(r) : k \in \mathbb{Z}_Q, i \in \mathbb{N}\}$ denote the family of scaled automorphisms over $\mathcal{R}_Q$, where $\sigma_i : m(X) \to m(X^i)$ denotes an automorphism in the Galois group $\mathrm{Gal}(\mathcal{K}/\mathbb{Q})$.

Theorem A.22 (Security of Pirouette). *Suppose LWE encryptions with message space $\mathbb{Z}_q$ are IND-CPA secure and $\mathcal{F}_{scal}$-KDM secure. Suppose RLWE encryptions with message space $\mathcal{R}_q$ are IND-CPA secure and KDM secure with respect to the family of $\mathcal{F}_{quad}$ and $\mathcal{F}_{aut}$. Then our construction in Section 4.1 satisfies query privacy when we model the PRG as a random oracle. Precisely, for any database parameter $\mathsf{pp}_{\mathsf{db}} = \mathsf{pp}_{\mathsf{db}}(\lambda)$ and any probabilistic polynomial-time adversary $\mathcal{A}$, it holds that*

$$\left| Pr[\mathcal{A}^{\mathcal{O}_b(\mathsf{sk},\cdot,\cdot)}(1^\lambda, \mathsf{evk}) = b] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda),$$

where $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{db}})$, $b \xleftarrow{\$} \{0,1\}$, *and the oracle* $\mathcal{O}_b(\mathsf{sk}, \mathsf{idx}_0, \mathsf{idx}_1)$ *outputs* $\mathsf{Query}(\mathsf{sk}, \mathsf{idx}_b)$.

The proof proceeds with a hybrid argument:

- Hyb_0: This is the real query privacy game. The challenger samples $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{db}})$ and gives evk to $\mathcal{A}$. Specifically, the challenger samples sk that consists of LWE secret key $\tilde{s}$ and RLWE secret keys, as well as evaluation keys for FHE procedures including high-precision bit decomposition, LWEtoRGSW conversion, and response compression. The challenger samples a random bit $b \xleftarrow{\$} \{0,1\}$, and replies to the adversary's queries $(\mathsf{idx}_0, \mathsf{idx}_1)$ with $(\tilde{\mathbf{a}}, \tilde{b}) \leftarrow \mathsf{Query}(\mathsf{sk}, \mathsf{idx}_b)$, where the $\tilde{\mathbf{a}}$ part is generated from a uniformly chosen PRG seed and $\tilde{b} = \langle \tilde{\mathbf{a}}, \tilde{\mathbf{s}} \rangle + \Delta \cdot \mathsf{idx}_b + e \bmod q$. Then the challenger sends this PRG seed and the $\tilde{b}$ part to $\mathcal{A}$. At the end of the experiment, the adversary outputs a bit $b' \in \{0,1\}$. The experiment outputs 1 if $b = b'$.
- Hyb_1: Same as Hyb_0 except that the challenger substitutes the evaluation keys evk for values sampled uniformly at random from their domain.
- Hyb_2: Same as Hyb_1 except that the challenger replies to each adversary's query with $(\tilde{\mathbf{a}}, \tilde{b})$, where $\tilde{\mathbf{a}}$ is derived from the PRG and $\tilde{b} \xleftarrow{\$} \mathbb{Z}_q$ is uniformly sampled instead.

Let $\mathsf{Hyb}_i(\mathcal{A})$ denote the output of the experiment Hyb_i with adversary $\mathcal{A}$. By construction, the adversary's view in Hyb_2 is completely independent of the bit b. Therefore, no adversary can do better than guess a random bit, i.e.

$$\left| \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \right| = \frac{1}{2}$$

for all adversaries $\mathcal{A}$.

It remains to show that for $i = 0, 1$, the outputs of $\mathsf{Hyb}_i(\mathcal{A})$ and $\mathsf{Hyb}_{i+1}(\mathcal{A})$ are computationally indistinguishable for any probabilistic polynomial-time adversary $\mathcal{A}$.

Lemma A.23. *Suppose that the LWE instance is secure. Then for any probabilistic polynomial-time adversary $\mathcal{A}$, $\mathsf{Hyb}_1(\mathcal{A})$ and $\mathsf{Hyb}_2(\mathcal{A})$ are computationally indistinguishable if we model the PRG as a random oracle.*

Proof. Suppose there is a probabilistic polynomial-time adversary $\mathcal{A}$ such that $\left| \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] \right| \geq \epsilon$ for some non-negligible ϵ. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ that breaks the LWE security[4]:

[4]LWE security in the random oracle model: the challenger always picks $\mathbf{a}$ using the random oracle.

(1) The LWE security challenger starts by sampling $\mathbf{s} \leftarrow \chi_{key}$.
(2) Algorithm $\mathcal{B}$ samples the evaluation keys evk for values sampled uniformly at random from their domain.
(3) Algorithm $\mathcal{B}$ samples a bit $\beta \xleftarrow{\$} \{0,1\}$. When $\mathcal{A}$ makes a query $(\mathsf{idx}_0, \mathsf{idx}_1)$, algorithm $\mathcal{B}$ queries the encryption oracle on idx_β to obtain the encoded query ct. It replies to $\mathcal{A}$ with ct.
(4) Finally, algorithm $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$. Algorithm $\mathcal{B}$ outputs 1 if $\beta = \beta'$ and 0 otherwise.

Let $b \in \{0,1\}$ be the bit of the LWE challenger samples.

- If $b = 0$, then the challenger replies to $\mathcal{B}$'s queries with the encryption of the queried message. In this case, $\mathcal{B}$ perfectly simulates Hyb_1 if we model the PRG as a random oracle. As such, $\mathcal{B}$ outputs 1 with $\left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$ and outputs $b = 0$ with probability $1 - \left|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]\right|$.
- If $b = 1$, then the challenger replies to $\mathcal{B}$'s queries with a pseudorandom $\mathbf{a}$ and a random $b \in \mathbb{Z}_q$. In this case, $\mathcal{B}$ perfectly simulates Hyb_2 and outputs 1 with $\left|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]\right|$.

Thus, the algorithm $\mathcal{B}$ outputs b with probability

$$\begin{aligned} &\left| \Pr[\mathcal{B}(1^\lambda) = b] - \frac{1}{2} \right| \\ &= \left| \frac{1}{2}(1 - \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]) + \frac{1}{2}\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \frac{1}{2} \right| = \frac{\epsilon}{2}. \end{aligned}$$

□

Lemma A.24. *Suppose the LWE encryption with message space $\mathbb{Z}_q$ is $\mathcal{F}_{scal}$-KDM secure, and RLWE encryption with message space $\mathcal{R}_q$ is KDM secure with respect to the family of $\mathcal{F}_{quad}$ and $\mathcal{F}_{aut}$. Then for any probabilistic polynomial-time adversary $\mathcal{A}$, $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$ are computationally indistinguishable.*

Proof. First, the key-switching keys in `BitDecomp` (Algorithm 3) are LWE encryptions of scaled secret keys. The reduction can simulate these elements using the KDM oracle (by querying on functions in $\mathcal{F}_{scal}$).

Next, the bootstrapping keys in `BitDecomp` (Algorithm 3), together with the bootstrapping and squaring keys in LWEtoRGSW, are RLWE encryptions of linear and quadratic functions of secret keys. The reduction can simulate these elements using the KDM oracle (by querying on functions in $\mathcal{F}_{quad}$).

Finally, the automorphism keys in LWEtoRGSW and in response compression are RLWE encryptions of scaled automorphisms of secret keys. The reduction can simulate these elements using the KDM oracle (by querying on functions in $\mathcal{F}_{aut}$).

Therefore, we conclude that $\mathsf{Hyb}_0(\mathcal{A})$ and $\mathsf{Hyb}_1(\mathcal{A})$ are computationally indistinguishable. □

Theorem A.25 (Security of Pirouette$^{\mathsf{H}}$). *Suppose LWE encryptions with message space $\mathbb{Z}_q$ are IND-CPA secure and $\mathcal{F}_{scal}$-KDM secure. Suppose RLWE encryptions with message space $\mathcal{R}_q$ are IND-CPA secure and KDM secure with respect to the family of $\mathcal{F}_{quad}$ and $\mathcal{F}_{aut}$. Then our construction in Section 4.2 satisfies query privacy when we model the PRG as a random oracle. Precisely, for any database parameter $\mathsf{pp}_{\mathsf{db}} = \mathsf{pp}_{\mathsf{db}}(\lambda)$ and any probabilistic polynomial-time*

adversary $\mathcal{A}$, it holds that

$$\left| Pr[\mathcal{A}^{\mathcal{O}_b(\mathsf{sk},\cdot,\cdot)}(1^\lambda, \mathsf{evk}) = b] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda),$$

where $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{db}})$, $b \xleftarrow{\$} \{0,1\}$, *and the oracle* $\mathcal{O}_b(\mathsf{sk}, \mathsf{idx}_0, \mathsf{idx}_1)$ *outputs* $\mathsf{Query}(\mathsf{sk}, \mathsf{idx}_b)$.

The proof proceeds with a hybrid argument:

- Hyb_0: This is the real query privacy game. The challenger samples $(\mathsf{sk}, \mathsf{evk}) \leftarrow \mathsf{Setup}(1^\lambda, \mathsf{pp}_{\mathsf{db}})$ and gives evk to $\mathcal{A}$. Specifically, the challenger samples sk that consists of LWE secret key $\tilde{s}$ and RLWE secret keys, as well as evaluation keys for FHE procedures including LWEtoRGSW conversion and response compression. The challenger samples a random bit $b \xleftarrow{\$} \{0,1\}$, and replies to the adversary's queries $(\mathsf{idx}_0, \mathsf{idx}_1)$ with $\{(\tilde{\mathbf{a}}_i, \tilde{b}_i)\}_{i\in[0,\log(\mathcal{N})-1]} \leftarrow \mathsf{Query}(\mathsf{sk}, \mathsf{idx}_b)$, where $\{\tilde{\mathbf{a}}_i\}_{i\in[0,\log(\mathcal{N})-1]}$ is generated from a uniformly chosen PRG seed and $\tilde{b}_i = \langle \tilde{\mathbf{a}}_i, \tilde{\mathbf{s}} \rangle + \Delta \cdot \mathsf{idx}_{b,i} + e \bmod q$ where $\mathsf{idx}_b = \sum_{i=0}^{\log(\mathcal{N})-1} \mathsf{idx}_{b,i} \cdot 2^i$. Then the challenger sends this PRG seed and the $\{\tilde{b}_i\}_{i\in[0,\log(\mathcal{N})-1]}$ part to $\mathcal{A}$. At the end of the experiment, the adversary outputs a bit $b' \in \{0,1\}$. The experiment outputs 1 if $b = b'$.
- Hyb_1: Same as Hyb_0 except that the challenger substitutes the evaluation keys evk for values sampled uniformly at random from their domain.
- $\mathsf{Hyb}_2, \ldots, \mathsf{Hyb}_k, \ldots, \mathsf{Hyb}_{\log(\mathcal{N})+1}$: Same as Hyb_1 except that the challenger replies to each adversary's query with $\{(\tilde{\mathbf{a}}_i, \tilde{b}_i)\}_{i\in[0,\log(\mathcal{N})-}$ where $\tilde{b}_i \xleftarrow{\$} \mathbb{Z}_q$ is uniformly sampled for $i \in [\log(\mathcal{N}) + 1 - k, \log(\mathcal{N}) - 1]$.

Let $\mathsf{Hyb}_i(\mathcal{A})$ denote the output of the experiment Hyb_i with adversary $\mathcal{A}$. By construction, the adversary's view in $\mathsf{Hyb}_{\log(\mathcal{N})+1}$ is completely independent of the bit b. Therefore, no adversary can do better than guess a random bit, i.e.

$$\left| \Pr[\mathsf{Hyb}_{\log(\mathcal{N})+1}(\mathcal{A}) = 1] \right| = \frac{1}{2}$$

for all adversaries $\mathcal{A}$.

Similar to Lemma A.23 and A.24, for each $i \in [0, \log(\mathcal{N})]$, the outputs of $\mathsf{Hyb}_i(\mathcal{A})$ and $\mathsf{Hyb}_{i+1}(\mathcal{A})$ are computationally indistinguishable for any probabilistic polynomial-time adversary $\mathcal{A}$. The proof follows analogously and is omitted for brevity.