

ReinsPIRe: High-Throughput, Low-Communication PIR with Server Preprocessing

Rasoul Akhavan Mahdavi Sarvar Patel Joon Young Seo Kevin Yeo
Google

Abstract

We present ReinsPIRe, a new single-server PIR with higher throughput (smaller computation costs) than all previous schemes that avoid offline communication. The crux of ReinsPIRe is new techniques for leveraging preprocessing in the common reference string (CRS) model to bring down the compute cost significantly. We show that the majority of the server’s FHE operations may completely avoid NTT polynomial encodings while maintaining the ability to efficiently multiply polynomials using matrix operations. As a result, most computation may be performed with moduli that are natively supported by modern hardware substantially improving throughput. Furthermore, our techniques result in more compact cryptographic materials reducing memory bandwidth bottlenecks. To obtain ReinsPIRe, we apply our new techniques to the prior InsPIRe construction [Mahdavi *et al.*, S&P 2026] to obtain a higher throughput single-server PIR while maintaining the same low communication costs. ReinsPIRe obtains 5.9 GB/s throughput per server core that is a 2x improvement.

We apply ReinsPIRe to improve two different applications. First, we consider the in-storage PIR setting where large databases are stored in external memory (such as SSD) that introduces additional storage I/O costs. In this model, ReinsPIRe obtains 3.3x better throughput than prior works due to its smaller cryptographic material. Secondly, we present a verifiable PIR scheme vReinsPIRe building upon our formulation of server computation as matrix operations. vReinsPIRe provides the smallest online communication cost while using nearly 50x smaller long-term client storage compared to prior works with low online communication.

Contents

1	Introduction	4
1.1	Our Contributions	4
2	Preliminaries	5
2.1	Connection between PIR and LHE	6
2.2	LHE and vLHE Syntax	6
2.3	LHE Security	6
3	ReinsPIRe: High-Throughput LHE	7
3.1	Prior LWE-based LHEs with CRS	7
3.2	Revisiting InspiRING	8
3.3	Polynomial Product Sums as a Matrix	8
3.4	ReinspiRING : Efficient Ring Packing	10
3.5	Improving Compilation and Preprocessing	12
3.6	ReinsPIRe Construction	12
3.7	ReinsPIRe Analysis	14
4	vReinsPIRe: Our Verifiable LHE	14
4.1	Revisiting Prior Verifiable LHE	14
4.2	vReinsPIRe Construction	15
4.3	vReinsPIRe Analysis	15
5	Experimental Evaluation	17
5.1	PIR Evaluation	17
5.1.1	PIR with In-Memory Database	18
5.1.2	In-Storage PIR with Sharded Databases	18
5.1.3	PIR for Password Checkup	18
5.2	vPIR evaluation	19
6	Related Works	19
A	LHE and vLHE Preliminaries	24
A.1	LHE and vLHE Syntax	24
A.2	LHE and vLHE Security Definitions	25
B	LHE and VLHE Security Proofs	25
B.1	LHE Security Proof	25
B.2	vLHE Security Proof	26
C	Improving Compile Runtime	27
D	InspiRING Modifications and Improvements	28
D.1	Addressing d^{-1} Existence with Power-of-two q	28
D.2	Optimizing InspiRING.Preprocess	29
D.3	Communication Efficient PIR from ReinsPIRe	30
E	Additional Experiments	31
E.1	Additional Optimizations	31
E.2	Breakdown of ReinspiRING Runtime	31
E.2.1	Breakdown of ReinsPIRe Performance	31
E.3	Improvements on InsPIRe	31
F	More Comparisons with Related Works	31
F.1	Comparisons with Bae et al. [11]	31

G	ReinspiRING with a Single Key-switching Key	32
G.1	InspiRING with a Single Key-switching Key	32
G.2	ReinspiRING with a Single Key-switching Key	33

1 Introduction

Private Information Retrieval (PIR) is an important cryptographic primitive enabling clients to retrieve entries from a server-held database without the server learning the client’s entry of interest. In more detail, the server holds an n -entry database where the client may query the i -th entry such that the client’s index $i \in [n]$ remains private from the server. In recent years, PIR has been shown to be a critical building block for many privacy-preserving systems including block lists [44], communication systems [1, 4], file systems [59] and web search [40] to list some examples. In industry, PIR has been used to enable caller ID lookup [6], content filtering [5], device enrollment [77] and password leak check [46] with strong privacy guarantees. Furthermore, PIR is being proposed for privately reading data in Ethereum [9].

The two main efficiency measures for PIR are the communication costs between the client and the server and the computational costs measured by server throughput. Over the past decade, there are many different PIR constructions offering varying trade-offs between communication and computation costs along with different trust and/or setup assumptions. For example, PIR has been studied with multiple servers that require strong non-colluding trust assumptions between servers. As this requirement is hard to realize in practical settings, we focus on single-server PIR throughout our work.

There exist many single-server PIR using homomorphic encryption that obtain very small communication costs (see [3, 4, 16, 60, 61, 64] and references therein). Unfortunately, these schemes suffer from a couple of drawbacks. First, they are quite computationally expensive since they leverage RLWE ciphertexts for small communication that necessitates performing most modular arithmetic over primes (or powers of prime). Additionally, most of these constructions require the server to store a large public key (on the order of MBs) for each querying client. For applications with many clients, the server’s storage costs would grow impractically. Furthermore, if the cost of uploading this public key is included, the communication cost becomes quite large, particularly when clients only perform a small number of PIR queries.

On the other end of the spectrum, there also exist PIR schemes with small computational costs. In the extreme case, recent works have shown PIR schemes whose computational costs are sublinear in the database size. Client-specific preprocessing schemes (such as [25, 42, 73, 75, 78]) require the client to download the entire database during setup and after every $\tilde{O}(\sqrt{n})$ queries. Furthermore, the client storage must be updated appropriately if any database entry changes. Recent breakthrough work [52] presented doubly efficient PIR schemes without any offline communication, but unfortunately they remain impractical to date [66].

There also exist PIR schemes with high throughput (small computation costs) that operate over the entire database. The recent works of SimplePIR [41] and FrodoPIR [28] present LWE-based constructions where all server arithmetic is performed using 32-bit integers that are natively supported by current hardware. As a result, all these constructions obtain very high throughput that approach the memory bandwidth of the underlying machine. Unfortunately, each of these constructions also suffer significant drawbacks as well. Both SimplePIR and FrodoPIR require large client storage that must be downloaded before any PIR query and this client storage must be updated every time any database entry changes. Furthermore, all these constructions incur significant more communication costs than the earlier RLWE-based schemes (which are computationally inefficient).

A recent line of single-server PIR work (including [30, 40, 48, 56, 58, 59, 62]) has attempted to bridge this gap. Building on top of the LWE-based SimplePIR [41] and FrodoPIR [28], these constructions successfully remove the client storage (and offline communication) requirements while also reducing communication costs by incorporating RLWE-based techniques. However, they require a larger amount of server arithmetic on non-natively supported integer primitives increasing computational costs. These schemes provide interesting trade-offs of smaller communication at the cost of lower throughput.

1.1 Our Contributions

New High-Throughput PIR. As our main contribution, we present a new single-server PIR scheme, ReinsPIRe, that requires no offline communication and only requires server-side preprocessing in the CRS model. ReinsPIRe successfully bridges the gap between the high throughput of LWE-only PIR schemes (such as SimplePIR [41]) and the low communication of recent PIR constructions that incorporate RLWE-based techniques (such as InsPIRe [58]). In more detail, ReinsPIRe obtains the same communication costs as InsPIRe [58], the prior state-of-the-art PIR with no offline communication, while enjoying 2x faster computational costs. In terms of throughput, ReinsPIRe achieves 6.3 GB/s that is 62% of the throughput of SimplePIR [41], all while requiring no offline communication or client storage.

To obtain ReinsPIRe, we present a new ring packing algorithm ReinspiRING where the core idea is to omit any usage of polynomials in the NTT domain. Instead, ReinspiRING performs all modular arithmetic over natively supported 32-bit or 64-bit integers. However, without the NTT domain, we effectively lose the capability of performing efficient polynomial multiplication. Nevertheless, we show that one can still perform efficient plaintext-ciphertext polynomial multiplication using matrix operations by carefully leveraging the fixed random components of RLWE ciphertexts during preprocessing. Furthermore, the preprocessed

cryptographic material of ReinspiRING is also significantly smaller since they are stored in the coefficient domain where the entry norms are bounded as opposed to the NTT domain where entries have arbitrary norms. The smaller cryptographic material reduces the memory bandwidth as a bottleneck. By plugging ReinspiRING into InspiRE [58], we obtain ReinsPIRe with higher throughput while maintaining the same communication as InspiRE.

Application 1: In-Storage and Sharded PIR. We apply ReinsPIRe to the in-storage PIR setting where the database is large and stored in secondary storage (SSD). In this model, the online query throughput also depends on the storage I/O cost required to read the relevant preprocessed material from SSD needed by the server to respond to the query. In-storage PIR is required for applications where the database is very large (such as password leak check with billions of leaked credentials [74]). In such large-scale applications, the database is typically split into shards and PIR is only executed on a single shard to make the cost feasible as done in password leak check by Microsoft [46] and image search by Apple [8] that we denote as sharded PIR¹. ReinsPIRe obtains substantially higher throughput for in-storage and sharded PIR due to smaller preprocessed material reducing storage I/O costs and by storing the database in its native representation.

Application 2: Verifiable PIR. We extend ReinsPIRe to obtain a new verifiable PIR construction, vReinsPIRe. At a high level, prior works [29, 72] provide verifiability by having the client perform linearity checks with a challenge to ensure that the server has performed its computation correctly. By utilizing our new formulation of server computation as matrix operations, vReinsPIRe performs these checks without requiring large client storage or costly homomorphic evaluations.

vReinsPIRe results in the smallest online communication cost compared to prior works. Furthermore, vReinsPIRe uses nearly 50x smaller long-term client storage compared to the closest low communication scheme of VeriSimplePIR [29]. Compared to the recent vPIR [72], vReinsPIRe obtains 10x smaller online communication but requires larger client storage (5.83 MB compared to 256 KB for a 1 GB database).

2 Preliminaries

Notation. Throughout the paper we will use 0-index base, and extend this convention to denote $[n]$ as the set $\{0, \dots, n-1\}$.

We use bold case capital letter (e.g. $\mathbf{M}$) to denote a matrix and bold lower case letter (e.g. $\mathbf{x}$) to denote a vector. When convenient, we will also use $[x_i]_{i \in [n]}$ to denote a vector $[x_0, x_1, \dots, x_{n-1}]$, and if the length is clear from context, omit the subscript and write $[x_i]$. We extend this notation to $[x_{i,j}]_{i \in [m], j \in [n]}$ to denote an $m \times n$ matrix.

We use $a(X)$ to denote a polynomial, and when clear from context, may omit the formal variable X and simply write a .

Common Reference String (CRS) Model. Our constructions work in the common reference string (CRS) model where we assume that there exists a uniformly random string (generated by a trusted third party) accessible to all parties. The CRS model for PIR has been used previously in [41, 48, 58, 62].

SIS. Informally, Short-Integer-Solution (SIS) hardness states that it is computationally hard to find a short vector $\mathbf{s}$ satisfying $\mathbf{A}\mathbf{s} = \mathbf{0}$ for a random $\mathbf{A}$. Following [29, 72], our work also uses SIS hardness in our verifiable PIR construction.

KDRLWE. Key-dependent RLWE (KDRLWE) is a decision RLWE problem with adversary's additional access to oracle encryptions of (functions of) the secret key. KDRLWE hardness assumption is widely used in homomorphic encryption literature to provide additional functionalities (e.g. key-switchings).

Independence Heuristic. We use independence heuristic when deriving noise bounds of our PIR construction, which assumes that the noise components that arise in homomorphic operations are independent subgaussian variables.

LWEs-to-RLWE. We consider LWEs-to-RLWE with preprocessing as a tuple of algorithms (Setup, Preprocess, Pack) :

- $\text{pp} \leftarrow \text{Setup}(1^\lambda)$: A trusted party outputs CRS and other public parameters.
- $\text{sp} \leftarrow \text{Preprocess}(\text{pp}, \hat{\mathbf{H}} \in \mathbb{Z}_q^{d \times d})$: Given pp and d random components of the LWE ciphertexts, outputs preprocessed materials sp .
- $(a_i, b_i) \leftarrow \text{Pack}(\text{pp}, \text{sp}, \mathbf{c} \in \mathbb{Z}_q^d, \text{evk})$: Given d message encoding components of the LWE ciphertexts and evaluation key evk , outputs a RLWE ciphertext (a_i, b_i) encrypting the original LWE plaintexts.

We will use this syntax to refer to the InspiRING LWEs-to-RLWE algorithm that our construction builds on.

¹As a note, it is possible to consider in-storage PIR without sharding and sharded PIR without an in-storage database. However, the most natural setup for large database settings is both in-storage and sharded PIR.

2.1 Connection between PIR and LHE

Informally, linear homomorphic encryption (LHE) [41, 72] is a protocol between a server and a client, where the server holds a plaintext matrix $\mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}$, and the client's goal is to compute $\mathbf{D}\mathbf{p} \bmod p$ without revealing $\mathbf{p}$ to the server. Verifiable LHE (vLHE) provides additional security guarantees against malicious servers.

PIR can be constructed from LHE as follows: the matrix $\mathbf{D}$ is a database, and to retrieve the i -th column of $\mathbf{D}$, the client generates a one-hot vector $\mathbf{p}$ that has the i -th entry as 1 and the rest 0.

For full generality (which will be useful in our verifiable PIR construction), we will present our protocols as LHE and vLHE protocols for the remainder of the paper. We note that optional "folding" steps on $\mathbf{D}\mathbf{p}$ can further reduce the response size to obtain a communication efficient PIR.

2.2 LHE and vLHE Syntax

LHE. We consider single-server Linear Homomorphic Encryption (LHE) with server preprocessing with single roundtrip queries consisting of the tuple of efficient algorithms $\text{LHE} = (\text{Setup}, \text{Preprocess}, \text{Query}, \text{Eval}, \text{Dec})$:

- $\text{pp} \leftarrow \text{Setup}(1^\lambda)$: A trusted party outputs CRS and other public parameters and distributes to the server and the client.
- $\text{sp} \leftarrow \text{Preprocess}(\text{pp}, \mathbf{D})$: The server preprocesses the database and creates server preprocessed materials sp .
- $(\text{st}, \text{qry}) \leftarrow \text{Query}(\text{pp}, \mathbf{p})$: The client receives query vector $\mathbf{p}$ and produces a local state and a query for the server.
- $\text{resp} \leftarrow \text{Eval}(\text{pp}, \text{sp}, \mathbf{D}, \text{qry})$: The server executes the query against the database to produce a response.
- $\text{ans} \leftarrow \text{Dec}(\text{pp}, \text{st}, \text{resp})$: The client decrypts the response to obtain the final output of the LHE protocol.

For correctness, the client should obtain $\text{ans} = \mathbf{D} \cdot \mathbf{p} \bmod p$ with probability at least $1 - \text{negl}(\lambda)$ over the internal randomness of the LHE protocol with negligible error, for honest database $\mathbf{D}$.

In verifiable LHE scheme in Section 4, we extend the syntax to apply to plaintext query matrix $\mathbf{P} \in \mathbb{Z}^{n_0 \times t}$. It is assumed the syntax above is applied to each column of $\mathbf{P}$ individually.

Verifiable LHE. We extend the notion of LHE and consider single-server Verifiable LHE, which consists of tuple of algorithms $\text{vLHE} = (\text{Setup}, \text{Preprocess}, \text{Register} = \{(\text{Register}_j^S, \text{Register}_j^C)\}_{j \in [k]}, \text{Query}, \text{Eval}, \text{VerDec})$. At a high level, vLHE has an additional (possibly multi-round) Register subprotocol between the client and the server which is typically used to bind the malicious server to a database. In the honest execution, the client should obtain $\text{ans} = \mathbf{D} \cdot \mathbf{p} \bmod p$ with probability at least $1 - \text{negl}(\lambda)$ for all honest database $\mathbf{D}$. We defer formal syntax to Appendix A.1.

Table 1: Table of common symbols used throughout the paper.

Symbol	Description
<i>Lattice Parameters</i>	
d	LWE dimension/RLWE degree (power of two)
q	LWE/RLWE ciphertext modulus
R	Cyclotomic ring $\mathbb{Z}[X]/(X^d + 1)$
R_q	Cyclotomic ring $\mathbb{Z}_q[X]/(X^d + 1)$
ℓ	Number of limbs in gadget decomposition
p	Plaintext modulus
Δ	Encryption scaling factor $\lfloor q/p \rfloor$
<i>System Parameters</i>	
n_1	Number of rows of the plaintext DB matrix ($d \mid n_1$ WLOG)
n_0	Number of columns of the plaintext DB matrix

2.3 LHE Security

Our LHE security definitions are largely based on [72] but generalizes to semi-honest LHE. We defer formal security definitions to Appendix A.2, but present the ideal functionality in Figure 1 for motivation. Intuitively, $\mathbf{D}_{\text{init}}$ is the initial reference database

that honest server uses. The registration phase (in the malicious setting) binds the server to *some* database in $\mathcal{D}_a$ (which need not necessarily equal $\mathbf{D}_{\text{init}}$) and we require all subsequent responses to be consistent with this bound database.

For a database class $\mathcal{D}_a \subseteq \mathbb{Z}^{n_1 \times n_0}$, message space $Q = \mathbb{Z}^{n_0}$, plaintext modulus p , and an initial reference database $\mathbf{D}_{\text{init}}$, we define the ideal functionality $\mathcal{F}_{\text{LHE}}$ as the following stateful functionality:

- Implicitly set $\mathbf{D}_{\text{id}} \leftarrow \mathbf{D}_{\text{init}}$ and $\text{status}_{\text{id}} \leftarrow \text{active}$ for each id.
- On register request (id, Register):
 - Send (id, Register) to the adversary.
 - Receive database $\mathbf{D} \in \mathcal{D}_a$ or Abort from the adversary.
 - If the adversary says Abort, set $\text{status}_{\text{id}} \leftarrow \text{abort}$ and return $\perp$ to the client.
 - Set $\mathbf{D}_{\text{id}} \leftarrow \mathbf{D}$
- On query request (id, Query, $\mathbf{p}$), where $\mathbf{p} \in Q$:
 - Send (id, Query, $\perp$) to the adversary.
 - Receive Allow or Abort from the adversary.
 - If $\text{status}_{\text{id}} = \text{abort}$ or the adversary says Abort, return $\perp$ to the client.
 - Return $\mathbf{D}_{\text{id}} \cdot \mathbf{p} \bmod p$ to the client.

Figure 1: Ideal functionality $\mathcal{F}_{\text{LHE}}$

3 ReinsPIRe: High-Throughput LHE

We present ReinsPIRe, our high-throughput single-server LHE protocol only using server-side preprocessing (meaning no offline communication). To obtain high throughput, ReinsPIRe is built on top of InsPIRe and follows the same LWE-based LHE framework introduced in [28, 41]. Before we present our construction, we will first revisit prior LWE-based LHE protocols [40, 48, 58, 62] and explain their shortcomings.

3.1 Prior LWE-based LHEs with CRS

Currently, all high throughput LHE protocols are based on the LWE-based LHE framework of SimplePIR [41] and FrodoPIR [28]. At a high level, the database is modeled as a matrix $\mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}$, and to compute $\mathbf{D} \cdot \mathbf{p}$, the client sends n_0 LWE ciphertexts encrypting $\mathbf{p}$. The n_0 LWE ciphertexts can be represented as a matrix $[\mathbf{A} \mid \mathbf{b}] \in \mathbb{Z}_q^{n_0 \times (d+1)}$, and the entire server processing can be concisely expressed as a matrix multiplication: $\mathbf{D} \cdot [\mathbf{A} \mid \mathbf{b}] = [\mathbf{H} \mid \mathbf{c}] \in \mathbb{Z}_q^{n_1 \times (d+1)}$. The rows of $[\mathbf{H} \mid \mathbf{c}]$ are LWE ciphertexts encrypting the $\mathbf{D} \cdot \mathbf{p}$. SimplePIR’s observation was that with the random matrix $\mathbf{A}$ fixed, the matrix $\mathbf{H}$ can be precomputed and shared with the clients prior to queries. Then, computations involving the query dependent part simply boils down to a matrix-vector multiplication: $\mathbf{D} \cdot \mathbf{b}$. The downside to this, however, is that the matrix $\mathbf{H}$ needs to be transmitted to the clients in the offline phase.

Following Tiptoe [40], subsequent works like HintlessPIR [48], YPIR [62], and InsPIRe [58] employed additional algorithms to eliminate the offline hint matrix $\mathbf{H}$. The general common theme is to somehow compress the space inefficient LWE ciphertexts $[\mathbf{H} \mid \mathbf{c}]$ into more space-efficient RLWE representations.

As introduced and formalized in HintlessPIR, prior works leverage the CRS model to preprocess much of the data for this compression step. At a high level, the online stage of this compression step can be expressed as the summation of polynomial products²: $\sum_i t_i(X) \cdot y_i(X)$ where $t_i(X)$ are the preprocessable polynomials depending only on the random components and the database (e.g. $\mathbf{A}$ and $\mathbf{D}$), and $y_i(X)$ are the query dependent polynomials sent by the client at query time. To make this step as efficient as possible, all these works store and transmit these polynomials in NTT representations to perform polynomial

²We note that this is a slight simplification of what exactly happens in these algorithms. However, it is true that much of the expensive portions in these algorithms essentially boil down to sums of polynomial products between preprocessed polynomials and query polynomials.

multiplication efficiently. However, this has several subtle implications that can negatively impact the practical performance of the protocol:

- **Hardware native modulus:** One of the main drivers of SimplePIR’s high throughput is its use of hardware native ciphertext modulus, like $q = 2^{32}$. However, to convert polynomials into NTT representations, the ciphertext modulus must be composed of NTT-friendly primes. As a result, polynomial operations naturally entail modular reduction algorithms like Barrett’s reduction, which can significantly impact the overall performance. On the other hand, modular reductions are essentially free with hardware native moduli.
- **Memory bandwidth:** In many situations, the preprocessed polynomials $t_i(X)$ are the gadget decomposed polynomials (e.g. used in key-switchings) have bounded norms in coefficient representations. However, this is no longer the case in NTT representations where the coefficients essentially may occupy the entire ciphertext modulus space. Considering that the primary bottleneck of PIR is *memory bandwidth* [41], it is evident that NTT representation is not the most optimal representation for memory bandwidth.

3.2 Revisiting InspiRING

We briefly revisit InspiRING in InsPIRe [58] that we build upon. The main idea behind InspiRING is to use the trace operator where the conjugates are expressed in terms of the two generators, $X \mapsto X^5$ and $X \mapsto X^{-1}$, of the automorphism group (see Lemma 1 in [58]) to transform each LWE ciphertext into an intermediate ciphertext over $R_q^d \times R_q$. For our purpose, one can view this ciphertext as an MLWE ciphertext with rank d and degree d where the first component R_q^d is the random part and the second component R_q is the plaintext encoding. This transformation encodes the original LWE plaintext in the constant term of the plaintext polynomial.

This intermediate ciphertext has the necessary homomorphic properties that enables aggregation of d such ciphertexts into a single ciphertext, where each LWE plaintext is placed in each coefficient of the plaintext polynomial.

The packed ciphertext is over $R_q^d \times R_q$, which is large. The final stage of InspiRING reduces this ciphertext into a standard RLWE ciphertext using a sequence of $d - 1$ key-switchings, where each key-switching reduces the number of the random components by one. The crucial insight is that the secret key components of the transformed intermediate ciphertext is related by the automorphisms in the trace operator (which are generated by the two generators). This enables the entire reduction to be performed using only two key-switching keys mapping $s(X^5) \mapsto s(X)$ and $s(X^{-1}) \mapsto s(X)$.

As formalized in HintlessPIR [48], the components that only depend on the random components can be preprocessed in the offline phase. After preprocessing, the online phase of the algorithm can be succinctly represented as the following sum of polynomial products, assuming number of limbs in gadget decomposition is 1:

$$\left(\sum_{i=0}^{d/2-1} t_i(X) \cdot y(X^{5^i}) + t'_i(X) \cdot y(X^{-5^i}) \right) + t''(X) \cdot y'(X). \quad (1)$$

Note, t_i, t'_i, t'' are the server preprocessed polynomials (that only depend only on the database and the CRS) and y, y' are the message encoding component of the two key-switching keys that the client sends as part of the query.

We stress again that to perform polynomial multiplications efficiently in the online phase, these polynomials are in NTT representations restricting the ciphertext modulus to NTT-friendly primes.

3.3 Polynomial Product Sums as a Matrix

In this subsection, we present a generic compilation algorithm that will be the core building block of our efficient LWEs-to-RLWE ring packing algorithm that’s built on top of InspiRING. In the previous subsection, we’ve established that the online phase of InspiRING can be expressed as a structured summation of products of preprocessed polynomials and query polynomials. To make this summation of polynomial products as efficient as possible, our core idea is to compile these summation of polynomial products into a matrix-vector multiplication. In particular, the preprocessed polynomials are compiled into a matrix in the preprocessing phase, and the online query will be a vector representation of the polynomial coefficients. Then, the online phase of the summation will boil down to a simple matrix-vector multiplication.

To accomplish this, our key idea is to bypass the NTT representations and work in the coefficient representations. Crucially, we will use the following standard and well known matrix formulation of polynomial multiplications:

Lemma 1 (Matrix View of Polynomial Mult). *Let $a(X), b(X) \in R_q$, and define*

$$\text{Neg}(a(X)) := \begin{bmatrix} a_0 & -a_{d-1} & \cdots & -a_1 \\ a_1 & a_0 & \cdots & -a_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{d-1} & a_{d-2} & \cdots & a_0 \end{bmatrix}$$

and $\text{vec}(b(X)) := [b_0, b_1, \dots, b_{n-1}]^\top$. Then

$$a(X) \cdot b(X) \cong \text{Neg}(a(X)) \cdot \text{vec}(b(X))$$

At first, replacing polynomial multiplications with matrix-vector multiplication seems counter intuitive, as this incurs $O(d^2)$ compared to $O(d)$ in NTT. However, as we will show, the specific structure of InspiRING (Equation 1) allows us to amortize this cost and "fuse" these matrices as a single matrix in the preprocessing phase. In the end, the online stage can be expressed as a single matrix-vector multiplication. Meanwhile, the coefficient representation approach allows us to use a power-of-two ciphertext modulus instead of being confined to NTT friendly primes. Additionally, the resulting matrix is much more compact than the NTT represented polynomials due to the fact that coefficient representations typically have small bounded norms because of gadget decomposition. In contrast, their NTT representations do not provide the same small norm guarantees. These two properties lead to significantly faster runtime in practice compared to prior works.

We formulate this idea as the following general lemma, which states that this matrix can be computed efficiently using only the preprocessed polynomials.

Lemma 2 (Compilation Lemma). *Let $R_q = \mathbb{Z}_q[X]/(X^d + 1)$ for a power of two d . Let $[t_i(X)]_{i \in [k]}$ be a sequence of k polynomials $\in R_q$ such that $\|t_i(X)\|_\infty \leq B$ for some B . Let $y(X) \in R_q$ and let $[\tau_i]_{i \in [k]}$ be a sequence of k automorphisms. Then there exists a matrix $\mathbf{M} \in \mathbb{Z}_q^{d \times d}$ that is only a function of t_i and τ_i , such that $\|\mathbf{M}\|_\infty \leq kB$ and the following holds:*

$$\sum_{i=0}^{k-1} t_i(X) \cdot \tau_i(y(X)) \cong \mathbf{M} \cdot \text{vec}(y(X))$$

where vec is the vector representation of the coefficients of $y(X)$. Furthermore, $\mathbf{M}$ can be computed in $O(kd^2)$ time.

Proof. We transform this summation of polynomial products into matrix operations using Lemma 1. Specifically, for $t_i(X) \cdot \tau_i(y(X))$, we make the following transformations:

- $t_i(X) \mapsto \text{Neg}(t_i(X))$
- $\tau_i(y(X)) \mapsto \text{vec}(\tau_i(y(X)))$

where $\text{Neg}(t_i(X))$ is the negacyclic matrix of the coefficients of $t_i(X)$ and $\text{vec}(\tau_i(y(X)))$ is the column vector representation of the coefficients of $\tau_i(y(X))$. Then:

$$t_i(X) \cdot \tau_i(y(X)) \cong \text{Neg}(t_i(X)) \cdot \text{vec}(\tau_i(y(X))).$$

On the other hand, we observe that automorphisms in the cyclotomic ring $\mathbb{Z}_q[X]/(X^d + 1)$ simply permutes the coefficients (with possible negations). In other words:

$$\text{vec}(\tau_i(y(X))) = \mathbf{P}_{\tau_i} \cdot \text{vec}(y(X))$$

where $\mathbf{P}_{\tau_i} \in \mathbb{Z}_q^{d \times d}$ is a signed permutation matrix corresponding to the automorphism τ_i . In particular, we obtain:

$$t_i(X) \cdot \tau_i(y(X)) \cong \text{Neg}(t_i(X)) \cdot \mathbf{P}_{\tau_i} \cdot \text{vec}(y(X)).$$

Plugging this into the summation:

$$\begin{aligned} \sum_{i=0}^{k-1} t_i(X) \tau_i(y(X)) &\cong \left(\sum_{i=0}^{k-1} \text{Neg}(t_i(X)) \cdot \mathbf{P}_{\tau_i} \right) \cdot \text{vec}(y(X)) \\ &= \mathbf{M} \cdot \text{vec}(y(X)). \end{aligned}$$

It is easy to see that $\mathbf{M}$ can be computed in $O(kd^2)$. It remains to show that $\|\mathbf{M}\|_\infty \leq kB$. Negacyclic matrix transformation preserves norm, so $\|\text{Neg}(t_i(X))\|_\infty \leq B$. Since $\mathbf{P}_{\tau_i}$ are signed permutation matrices, right multiplication by $\mathbf{P}_{\tau_i}$ also preserves the norm. Summing these matrix terms k times, the final fused matrix has norm bounded by kB . $\square$

We present the transformation of Lemma 2 in Algorithm 1.

Algorithm 1 Compile

Input: $[t_i(X)]_{i \in [k]}$: A sequence of k polynomials $\in R_q$ with bounded norm B .

Input: $[\tau_i]_{i \in [k]}$: A sequence of k automorphisms.

```

procedure Compile( $[t_i]$ ,  $[\tau_i]$ )
  Initialize zero  $\mathbf{M} \in \mathbb{Z}_q^{d \times d}$ 
  for  $i = 0$  to  $k - 1$  do
     $\mathbf{M} \leftarrow \mathbf{M} + \text{Neg}(t_i) \cdot \mathbf{P}_{\tau_i}$ 
  return  $\mathbf{M}$ 

```

3.4 ReinspiRING: Efficient Ring Packing

In this subsection, we present ReinspiRING, our high-throughput ring packing algorithm built on top of InspiRING. ReinspiRING uses Lemma 2 and Algorithm 1 as a core building block to improve the online phase.

Corollary 1 (Compilation of InspiRING). *In Equation 1, suppose that $\|t_i(X)\|_\infty \leq B$ and $\|t'_i(X)\|_\infty \leq B$. Then there exists a matrix $\mathbf{H}' \in \mathbb{Z}_q^{d \times d}$, $\|\mathbf{H}'\|_\infty \leq dB$ such that the following holds:*

$$\begin{aligned}
 & \left(\sum_{i=0}^{d/2-2} t_i(X) \cdot y(X^{5^i}) + t'_i(X) \cdot y(X^{-5^i}) \right) + t''(X) \cdot y'(X) \\
 & \cong \mathbf{H}' \cdot \text{vec}(y(X)) + t''(X) \cdot y'(X)
 \end{aligned}$$

Proof. We apply Lemma 2 to the summation term in Equation 1, noting that $X \mapsto X^{5^i}$ and $X \mapsto X^{-5^i}$ are all automorphisms in R_q . □

Remark. Note, the online phase of the original InspiRING requires $\approx d^2$ multiplications and additions in the NTT domain. Using Corollary 1, the online phase also incurs $\approx d^2$ multiplications and additions. However, using the compiled representation is much more efficient, since the ciphertext modulus need not be restricted to NTT-friendly primes, and also $\mathbf{H}'$ (typically in practice) has much smaller norm compared to the NTT domain polynomials $t_i(X)$ and $t'_i(X)$.

Handling the $t''(X) \cdot y'(X)$ term. We could use the same technique to represent the term $t''(X) \cdot y'(X)$ as a matrix-vector multiplication. However, this costs $O(d^2)$ runtime compared to the $O(d)$ point-wise multiplication in the NTT representation. But using a power-of-two modulus prevents us from transforming this term to NTT multiplications.

For this step, we use the following strategy, which was also used in YPIR [62]. The idea is to choose a sufficiently large NTT friendly prime modulus Q (or a product of NTT-friendly moduli) such that the result of the polynomial multiplication (interpreted over R instead of R_q) does not overflow Q ($Q > dq^2$ suffices). The input polynomials are lifted to this larger coefficient space Q and converted to NTT representations. In NTT, polynomial multiplication amounts to component-wise multiplication mod Q . The result is then converted back to the coefficient representation. Since Q was chosen large enough such that the result doesn't overflow Q , taking remainder of each coefficient mod q computes the original polynomial multiplication mod q . This strategy incurs $O(d \log d)$ instead of $O(d^2)$, which is more efficient than the matrix-vector multiplication approach.

However, this raises a question: we went on with the matrix formulation in the first place to get around NTT; why are we going back to NTT here? The main difference is that computing the term $t''(X) \cdot y'(X)$ occupies a very small portion of the entire packing time, less than 10% (see Appendix E). The major bottleneck is computing the summation $\sum_{i=0}^{d/2-2} t_i(X) \cdot y(X^{5^i}) + t'_i(X) \cdot y(X^{-5^i})$, and so representing this summation as a matrix-vector multiplication results in much faster runtime.

Discussions about Noise. We stress that our approach of interpreting polynomial operations as matrix operations computes exactly the same result as InspiRING, and hence does not introduce any additional noise.

Discussion about Matrix Norm. The dB bound of the matrix norm in Corollary 1 is a worst case bound, and is actually quite loose in practice. Assuming that each coefficient of $t_i(X)$ is independently uniform in $[-B, B]$, we can show that the norm is $O(\sqrt{d \log dB})$ with high probability using the standard Hoeffding's bound. Although t_i and t'_i are technically correlated in InspiRING, in practice the actual norm of $\mathbf{H}'$ is closer to this bound.

ReinspiRING construction. Extending on our prior discussions, we now formally present our ReinspiRING construction. Generalizing to gadget decomposition, Equation 1 in Section 3.2 becomes:

$$\sum_{j=0}^{\ell-1} \left(\sum_{i=0}^{d/2-2} t_{i,j}(X) \cdot y_j(X^{5^i}) + t'_{i,j}(X) \cdot y_j(X^{-5^i}) \right) + t''_j(X) \cdot y'_j(X) \quad (2)$$

where j indexes the gadget limbs. Thus, Corollary 1 presented in the previous section can be easily applied to the inner sum terms individually.

We present our algorithm in Algorithm 2, which consists of two algorithms $\text{ReinspiRING} = (\text{Preprocess}, \text{Pack})$. Preprocess is executed in the offline phase while Pack is executed in the online phase.

Algorithm 2 ReinspiRING

Input: $\hat{\mathbf{H}} \in \mathbb{Z}_q^{d \times d}$: d LWE ciphertext masks.

Input: $[w_j(X)]_{j \in [\ell]}$: Mask of the KSK for $s(X^5) \mapsto s(X)$

Input: $[w'_j(X)]_{j \in [\ell]}$: Mask of the KSK for $s(X^{-1}) \mapsto s(X)$

```

procedure Preprocess( $\hat{\mathbf{H}}, [w_j], [w'_j]$ )
  st  $\leftarrow$  InspiRING.Preprocess( $\hat{\mathbf{H}}, [w_j], [w'_j]$ )
  Parse st as  $(\tilde{a}, [t_{i,j}]_{i \in [d/2-1], j \in [\ell]}, [t'_{i,j}]_{i \in [d/2-1], j \in [\ell]}, [t''_j]_{j \in [\ell]})$ 
  // Initialize sequence of automorphisms
  aut  $\leftarrow [X \mapsto X^{5^i}]_{i \in [d/2-1]} + [X \mapsto X^{-5^i}]_{i \in [d/2-1]}$ 
  for  $j = 0$  to  $\ell - 1$  do
     $\mathbf{H}'_j \leftarrow \text{Compile}([t_{i,j}]_{i \in [d/2-1]} + [t'_{i,j}]_{i \in [d/2-1]}, \text{aut})$ 
   $\mathbf{H}' \leftarrow [\mathbf{H}'_0, \dots, \mathbf{H}'_{\ell-1}]$ 
  return  $(\tilde{a}, \mathbf{H}', \{t''_j(X)\}_{j \in [\ell]})$ 

```

Input: $\hat{\mathbf{c}} \in \mathbb{Z}_q^d$: Body of the d LWE ciphertexts.

Input: $[y_j(X)]_{j \in [\ell]}$: Body of the KSK for $s(X^5) \mapsto s(X)$

Input: $[y'_j(X)]_{j \in [\ell]}$: Body of the KSK for $s(X^{-1}) \mapsto s(X)$.

Input: $\tilde{a}, \mathbf{H}', \{t''_j(X)\}_{j \in [\ell]}$: Output from Preprocess

```

procedure Pack( $\hat{\mathbf{c}}, [y_j], [y'_j], \tilde{a}(X), \mathbf{H}'$ )
   $\mathbf{y} \in \mathbb{Z}_q^{\ell d} \leftarrow$  Coefficient vector of  $[y_j]$  concatenated.
   $\mathbf{z} \leftarrow \text{vec} \left( \sum_{j=0}^{\ell-1} t''_j \cdot y'_j \right)$ 
  return  $(\tilde{a}, \hat{\mathbf{c}} + \mathbf{H}' \cdot \mathbf{y} + \mathbf{z})$ 

```

InspiRING modification. The pseudocode directly uses InspiRING.Preprocess as a subroutine. It turns out that we need to make a small modification to the existing InspiRING algorithm to make it compatible with a power-of-two modulus q . In the algebraic derivation of the intermediate ciphertexts, the original InspiRING involves multiplication by an inverse of d . However, such inverse does not exist when q is even. To address this, we modify the initial algebraic derivation to *approximately divide by d* using modulus switching. This introduces additional additive noise, but it turns out to be practically negligible in the entire packing process. We defer detailed discussion to Appendix D.1.

Lemma 3 (ReinspiRING Runtime). *ReinspiRING.Preprocess runs in time $O(\ell d^3)$ and ReinspiRING.Pack costs $\ell d^2 + O(\ell d \log d)$ multiplications and additions.*

Lemma 4 (ReinspiRING Noise). *Let the errors in the key-switching keys be subgaussian with parameter σ_{ks} , and let z and ℓ be the gadget decomposition base and number of limbs. Under the independence heuristic, the additive noise introduced by ReinspiRING.Pack is $e_{\text{ks}}(X) + e_{\text{div}}(X)$ where the coefficients of e_{ks} are subgaussian with parameter σ_{pack} satisfying $\sigma_{\text{pack}}^2 \leq \ell d^2 z^2 \sigma_{\text{ks}}^2 / 4$ and, when the secret key is ternary, $\|e_{\text{div}}(X)\|_{\infty} \leq d^2$.*

Lemma 5 (Server Storage). *Let z and ℓ be the gadget decomposition base and number of limbs. The packing matrix $\mathbf{H}'$ output by ReinspiRING.Preprocess has size at most $\ell d^2 (\log_2(dz) + 1)$ bits, and $[t''_j]$ has size $\ell d \log_2(q)$ bits.*

Remark on server storage. Lemma 5 above uses the worst case norm bound of the matrix. However, as noted previously, in practice this is much smaller, roughly $\approx (\log_2 d)/2$ bits less. On the other hand, when representing the matrix in memory, in terms of runtime it's usually more advantageous for each entry to take up the entire native int type (e.g. 32 bit integer) than trying to optimize the representation at the bit level. Thus, the practical gain is much more noticeable if the max norm is near this int-type boundary. Note, compared to the NTT counterpart which requires $\ell d^2 \log_2 q$ bits to represent regardless of ℓ , with our representation the entry norm strictly decreases as ℓ increases. For larger ℓ , this may result in up to $\approx 4\times$ difference in server memory.

ReinspiRING with a single key-switching key. As more of a theoretical interest, we provide a version of ReinspiRING that uses single key-switching key in Appendix G. Our key insight is that the reason that InspiRING (and consequently ReinspiRING) requires two key-switching keys is because the underlying automorphism groups in power-of-two cyclotomic rings are generated by two generators. To bypass this limitation, we derive analogous results in prime cyclotomic rings which have cyclic automorphism groups. This allows us to use a single key-switching key corresponding to a generator.

Note, one of the reasons that power-of-two cyclotomic rings are prevalent in FHE is because they admit practically fast NTT. But in the preprocessing model that we work with in this paper, this efficiency is somewhat less important, since essentially almost all such operations that require NTT are preprocessed once in the offline phase.

3.5 Improving Compilation and Preprocessing

In Compile, computing the compiled matrix $\mathbf{M}$ naively incurs $\Theta(kd^2)$ time. While this matrix is computed by the server in the offline preprocessing step, it has to be recomputed for *every database updates*. Such computational overhead can be prohibitive in a system with frequent database updates and tight data freshness requirements.

We propose an algorithm that computes $\mathbf{M}$ in $O(d^2 \log d)$ time, and hence if $k = \omega(\log d)$, our new algorithm outperforms the naive construction (which is the case when used in ReinspiRING). The high level idea is to use the underlying structure of the polynomial summation: the right multiplicands are automorphic images of the same polynomial $y(X)$. With some algebraic manipulations, it turns out that computing each column of $\mathbf{M}$ amounts to evaluating a degree d polynomial $P(Z) \in R_q[Z]$ (where $R_q = \mathbb{Z}_q[X]/(X^d + 1)$), and mapping the resulting coefficients as the column. Furthermore, $P(Z)$ is efficiently computable from the $t_i(X)$. The evaluation points are powers of X^2 , which is a d -th primitive root of unity in $\mathbb{Z}_q[X]/(X^d + 1)$. This allows us to employ ring based FFT [21, 69] to evaluate $P(Z)$ over these d points in $O(d \log d)$ ring operations. Each ring operation incurs $O(d)$ time, so the total running time is therefore $O(d^2 \log d)$.

We also apply the technique in [69] to improve the offline preprocessing stage of InspiRING from $O(d^3 + \ell d^2 \log d)$ to $O(\ell d^2 \log d)$ (Appendix D.2). Thus, we can reduce the entire preprocessing stage of ReinspiRING to $O(\ell d^2 \log d)$ time. Looking ahead, this optimization plays a crucial role in making the preprocessing phase of our verifiable PIR construction practical. We defer details to Appendix C and D.2.

Lemma 6 (Improved ReinspiRING.Preprocess). *There exists an algorithm that computes ReinspiRING.Preprocess in time $O(\ell d^2 \log d)$.*

3.6 ReinsPIRe Construction

We now formally present our ReinsPIRe construction in Algorithm 3. ReinsPIRe follows the same general structure as prior LWE-based LHE protocols, but uses ReinspiRING for the compression of the LWE ciphertexts into RLWE ciphertexts. We note that in ReinsPIRe.Eval, the code "inlines" the ReinspiRING.Pack subroutine and express multiple packings as a single matrix-vector multiplication.

Communication efficient PIR from ReinsPIRe. We note that the original InspiRe has an additional "folding" step to reduce the response further. Our ReinsPIRe can be used as a blackbox to realize this folding to instantiate a more communication efficient PIR. We present this construction in Appendix D.3 as ReinsPIRe⁺. In our experimental evaluation, we use this construction in our semi-honest PIR comparisons.

Algorithm 3 ReinsPIRe

Input: $\mathbf{A} \in \mathbb{Z}_q^{n_0 \times d}$: Mask parts of the LWE ciphertexts.

Input: $[w_j(X)]_{j \in [\ell]}, [w'_j(X)]_{j \in [\ell]}$: KSK mask parts.

Input: $\mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}$: Plaintext database. Assume d divides n_1 .

procedure Preprocess($\mathbf{A}, [w_j], [w'_j], \mathbf{D}$)

$\mathbf{H} \leftarrow \mathbf{D} \cdot \mathbf{A} \in \mathbb{Z}_q^{n_1 \times d}$

return GenPack($\mathbf{H}, [w_j], [w'_j]$)

procedure GenPack($\mathbf{H}, [w_j], [w'_j]$)

for $i = 0$ to $n_1/d - 1$ **do**

$\hat{\mathbf{H}}_i \leftarrow \mathbf{H}[di : d(i+1)][:]$

$(\hat{a}_i, \mathbf{H}'_i, [t''_{i,j}]) \leftarrow \text{ReinspiRING.Preprocess}(\hat{\mathbf{H}}_i, [w_j], [w'_j])$

$\mathbf{H}' \leftarrow [(\mathbf{H}'_0)^\top, \dots, (\mathbf{H}'_{n_1/d-1})^\top]^\top$

return $([a_i], \mathbf{H}', [t''_{i,j}])$

Input: $\mathbf{A} \in \mathbb{Z}_q^{n_0 \times d}$: Mask of the LWE ciphertexts.

Input: $[w_j(X)]_{j \in [\ell]}, [w'_j(X)]_{j \in [\ell]}$: KSK mask parts.

Input: $\mathbf{p} \in \mathbb{Z}_p^{n_0}$: Plaintext query vector.

procedure Query($\mathbf{A}, [w_j], [w'_j], \mathbf{p}$)

$s(X) \leftarrow \text{Sample secret key over } R$

$\mathbf{s} \leftarrow \text{vec}(s(X))$.

$\mathbf{b} \leftarrow -\mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{p} \in \mathbb{Z}_q^{n_0}$

$z \leftarrow \text{gadget digit}$

for $j = 0$ to $\ell - 1$ **do**

$y_j \leftarrow -w_j(X)s(X) + e_j(X) + z^j \cdot s(X^5)$

$y'_j \leftarrow -w'_j(X)s(X) + e'_j(X) + z^j \cdot s(X^{-1})$

$\mathbf{y} \leftarrow [\text{vec}(y_0)^\top, \dots, \text{vec}(y_{\ell-1})^\top]^\top \in \mathbb{Z}_q^{\ell d}$

return $(\mathbf{u} = [\mathbf{b}^\top, \mathbf{y}^\top]^\top, \{y'_j\})$

Input: $([a_i], \mathbf{H}', [t''_{i,j}(X)])_{i \in [n_1/d], j \in [\ell]}$: Output from Preprocess

Input: $\mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}$: Plaintext database. Assume d divides n_1 .

Input: $(\mathbf{u}, [t''_j(X)]_{j \in [\ell]})$: Query to retrieve the i -th column of $\mathbf{D}$.

procedure Eval($([a_i], \mathbf{H}', [t''_{i,j}]), \mathbf{D}, \mathbf{u}, \{y'_j\}$)

for $i = 0$ to $n_1/d - 1$ **do**

$\mathbf{z}_i \leftarrow \text{vec}(\sum_{j=0}^{\ell-1} t''_{i,j} \cdot y'_j)$

$\mathbf{z} \leftarrow [\mathbf{z}_0^\top, \dots, \mathbf{z}_{n_1/d-1}^\top]^\top$

$\hat{\mathbf{x}} \leftarrow [\mathbf{D}, \mathbf{H}'] \cdot \mathbf{u} + \mathbf{z} \in \mathbb{Z}_q^{n_1}$

return $([a_i], \hat{\mathbf{x}})$

Input: $[a_i]$: Mask of the response RLWE ciphertexts.

Input: $\hat{\mathbf{x}} \in \mathbb{Z}_q^{n_1}$: Vector encoding of the i -th column of $\mathbf{D}$.

Input: $s(X) \in R_q$: Secret key

procedure Dec($[a_i], \mathbf{x}, s(X)$)

for $i = 0$ to $n_1/d - 1$ **do**

$\hat{x}_i(X) \leftarrow \text{Interpret each chunk of } \hat{\mathbf{x}}[di : d(i+1)] \text{ as coefficients over } R_q$

Decrypt each $(a_i, \hat{x}_i)$

3.7 ReinsPIRe Analysis

Theorem 1 (ReinsPIRe Runtime). *Using the improved preprocessing algorithm of Lemma 6, ReinsPIRe has the following running times:*

$$\begin{aligned} \text{Preprocess} &: O(n_1 n_0 d + \ell n_1 d \log d) \\ \text{Query} &: O(n_0 d + \ell d \log d) \\ \text{Eval} &: n_1(n_0 + \ell d) + O(n_1 \ell \log d) \\ \text{Dec} &: O(n_1 \log d) \end{aligned}$$

Theorem 2 (ReinsPIRe Noise). *Let the LWE query error be subgaussian with parameter σ_{lwe} and let the key-switching key errors be subgaussian with parameter σ_{ks} . Under the independence heuristic, the error polynomial in each ReinsPIRe response ciphertext is $e_{\text{main}}(X) + e_{\text{div}}(X)$ where the coefficients of e_{main} are subgaussian with parameter σ_{main} satisfying $\sigma_{\text{main}}^2 \leq n_0 p^2 \sigma_{\text{lwe}}^2 + \ell d^2 z^2 \sigma_{\text{ks}}^2 / 4$ and $\|e_{\text{div}}\|_\infty \leq d^2$.*

Theorem 3 (ReinsPIRe Communication). *Excluding the public CRS, ReinsPIRe upload size is $(n_0 + 2\ell d) \log_2 q$ bits and its download size is $2n_1 \log_2 q$ bits.*

Theorem 4 (Informal ReinsPIRe Security). *Let $\mathcal{D}_h = \{\mathbf{D} \mid \mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}, \|\mathbf{D}\|_\infty \leq p\}$. Assuming key-dependent RLWE hardness and ReinsPIRe satisfies the correctness condition, ReinsPIRe securely computes $\mathcal{F}_{\text{LHE}}$ (Figure 1) against a semi-honest server.*

The security proof largely follows the same structure as in InsPIRe [58], adapted to fit the definitions in Section 3. We defer formal proof to Appendix B.1.

4 vReinsPIRe: Our Verifiable LHE

In this section, we present vReinsPIRe, our high throughput verifiable PIR with small communication. vReinsPIRe is built on top of our semi-honest ReinsPIRe and follows the framework of VeriSimplePIR [29] and vPIR [72].

4.1 Revisiting Prior Verifiable LHE

We first recall the reusable verification framework of VeriSimplePIR [29], as presented in vPIR [72]. Let $\mathbf{A}$ be the public random matrix let $\mathbf{H}$ be the SimplePIR hint. VeriSimplePIR and vPIR use an extractable SIS-based commitment protocol as the core building block (see Section 2 for details on SIS). At a high level, this protocol binds the server to a bounded database $\tilde{\mathbf{D}}$ that satisfies $\mathbf{H} = \tilde{\mathbf{D}}\mathbf{A}$, and SIS hardness guarantees that it is computationally infeasible to find another $\tilde{\mathbf{D}}'$ with the same bounded norm guarantee.

The protocol proceeds as follows: the server first starts by sending the hint matrix $\mathbf{H}$ to the client and the client responds with a random challenge matrix $\mathbf{C}$. The server then responds with $\mathbf{Z}$, which in the honest case is computed as $\mathbf{Z} \leftarrow \mathbf{C}\mathbf{D}$. The client then checks that $\mathbf{Z}$ has bounded norm and $\mathbf{Z}\mathbf{A} \stackrel{?}{=} \mathbf{C}\mathbf{H}$. VeriSimplePIR shows that the server that passes this check with non-negligible probability can be rewound to extract a bounded matrix $\tilde{\mathbf{D}}$ such that $\mathbf{H} = \tilde{\mathbf{D}}\mathbf{A}$, and SIS hardness guarantees that this bounded $\tilde{\mathbf{D}}$ is computationally unique. Furthermore, $\mathbf{H} = \tilde{\mathbf{D}}\mathbf{A} \implies \mathbf{Z}\mathbf{A} = \mathbf{C}\tilde{\mathbf{D}}\mathbf{A} \implies (\mathbf{Z} - \mathbf{C}\tilde{\mathbf{D}})\mathbf{A} = \mathbf{0}$, and SIS hardness can be invoked again to show that $\mathbf{Z} = \mathbf{C}\tilde{\mathbf{D}}$.

However, $\mathbf{C}$ in the above protocol is sent in the clear, and hence cannot be reused. To obtain a reusable proof, VeriSimplePIR and vPIR encrypt $\mathbf{C}$ using a homomorphic encryption scheme and let the server homomorphically evaluate $\mathbf{C}\mathbf{D}$.

The online phase uses the binding properties above and employ Freivalds-style check [33] to verify that the server has executed the requested computation: $\tilde{\mathbf{D}}\mathbf{u}$ where $\mathbf{u}$ is the message encoding part of the LWE query ciphertexts. The client receives $\mathbf{x}$ from the server and checks $\mathbf{Z}\mathbf{u} \stackrel{?}{=} \mathbf{C}\mathbf{x}$. As $\mathbf{C}$ was hidden from the server, the client can conclude that $\mathbf{x} = \tilde{\mathbf{D}}\mathbf{u}$.

While VeriSimplePIR requires storing the hint matrix $\mathbf{H}$ for decryption, vPIR eliminates storing the hint matrix. For decryption, vPIR uses an LHE scheme like HintlessPIR [48] to delegate private evaluation of the LWE decryption circuit $\mathbf{H}\mathbf{s}$. To verify that the server has computed $\mathbf{y} \stackrel{?}{=} \mathbf{H}\mathbf{s}$ honestly, vPIR relies on another Freivalds style check: $\mathbf{Z}\mathbf{A}\mathbf{s} \stackrel{?}{=} \mathbf{C}\mathbf{y}$. vPIR proves that if this check passes, then the server has computed $\mathbf{H}\mathbf{s}$ honestly except with negligible probability.

Although vPIR [72] eliminates long term client storage of $\mathbf{H}$, homomorphically computing $\mathbf{H}\mathbf{s}$ can incur large computational and communication cost. Note that $\mathbf{H}\mathbf{s}$ is over the *original* LWE ciphertext modulus q , which is typically at least 32 bits. In particular, q becomes the *plaintext modulus* of the ciphertext encrypting $\mathbf{H}\mathbf{s}$, and naturally its ciphertext modulus and ring degree must be chosen large enough for sufficient noise budget and security. This inherently increases the request and the response

size³. Furthermore, in the HintlessPIR and vPIR instantiation, this homomorphic matrix-vector multiplication [37, 38] uses NTT-friendly RLWE arithmetic, incurring the high costs discussed in Section 3.

4.2 vReinsPIRe Construction

We resolve these limitations in vReinsPIRe. The high level reason why vPIR incurs large communication and computational overhead is because $\mathbf{H}\mathbf{s}$ has to be evaluated *privately* on the secret input $\mathbf{s}$. However, we can bypass this limitation in vReinsPIRe by observing that the linear evaluations that need to be verified are essentially public in ReinsPIRe.

In more detail, the dominant server response component in ReinsPIRe has the form $\mathbf{x} = [\mathbf{D}, \mathbf{H}'] \cdot \mathbf{u}$ which is a public linear function of the transmitted query vector $\mathbf{u}$. Our key observation is that this computation can be verified directly using the check $[\mathbf{Z}, \mathbf{Z}'] \cdot \mathbf{u} \stackrel{?}{=} \mathbf{C}\mathbf{x}$, where $\mathbf{Z} = \mathbf{C}\mathbf{D}$ and $\mathbf{Z}' = \mathbf{C}\mathbf{H}'$ is the $\mathbf{Z}$ analog of the packing matrix $\mathbf{H}'$. Crucially, since the packing matrix $\mathbf{H}'$ only depends on the hint matrix $\mathbf{H}$ and the CRS, the client can locally compute it, and the preprocessing optimization in Section 3.5 makes this practically efficient.

We note that these improvements in the online phase of the protocol do not come for free: the client has to perform additional computations in the preprocessing stage (to compute $\mathbf{H}'$), and we cannot use the compression optimization in vPIR using a power-of-two modulus. However, as we show in Section 5.2, the extra computation is relatively small compared to the entire preprocessing stage, and the increased client storage is still much smaller than storing the entire hint matrix $\mathbf{H}$. Therefore, vReinsPIRe provides substantial runtime and communication improvements in the online phase at a small cost of slightly larger preprocessing time and client storage.

We present vReinsPIRe in Algorithm 4. The general flow follows the framework of VeriSimplePIR [29] and vPIR [72]. However, there are several differences that we explain below:

Offline Phase: In the offline phase of vReinsPIRe, the client additionally locally computes the packing matrix $\mathbf{H}'$, as well as the masks of the packed ciphertexts $[a_i]$. It then computes $\mathbf{Z}' \leftarrow \mathbf{C}\mathbf{H}'$ locally (analogous to $\mathbf{Z} \leftarrow \mathbf{C}\mathbf{D}$) and stores these results for the online phase.

Online Phase: The client's query generation and the server's response computation are almost identical to ReinsPIRe, except the server does not compute the term involving the second key-switching key. Recall that $\mathbf{u}$ consists of the LWE query vector concatenated with the first key-switching key, and multiplying that to $[\mathbf{D}, \mathbf{H}']$ yields the (partial) body of the RLWE ciphertexts encrypting the result. To verify that the server has honestly computed $\mathbf{x} \stackrel{?}{=} [\mathbf{D}, \mathbf{H}'] \cdot \mathbf{u}$, the client verifies the linear relationships between $\mathbf{u}$ and $\mathbf{x}$ using the locally stored $\mathbf{Z}, \mathbf{Z}'$, and $\mathbf{C}$. Finally, the client locally computes the remaining term involving the second key-switching key to form the final ciphertext, and decrypts the result.

Use of LHE in Register. As noted in [72], a generic LHE construction (not necessarily verifiable) can be used to compute $\mathbf{D}^\top \mathbf{C}^\top$ homomorphically in the Register stage. We write with generic LHE syntax (Section 2) in the pseudocode, but use the ReinsPIRe construction in our evaluations.

4.3 vReinsPIRe Analysis

Theorem 5 (vReinsPIRe Client Storage). *The long term client storage cost of vReinsPIRe is at most $2n_1 \log_2(q) + \kappa n_0 \log_2(n_1 p) + \kappa \ell d \log_2(q) + \lambda$ bits.*

Theorem 6 (vReinsPIRe Runtime). *Let $T_{\text{Pre}}^{\text{LHE}}, T_{\text{Qry}}^{\text{LHE}}, T_{\text{Eval}}^{\text{LHE}}, T_{\text{Dec}}^{\text{LHE}}$ be the running time of the underlying LHE scheme. Using the improved ReinspiRING.Preprocess algorithm, vReinsPIRe has the following running time:*

$$\begin{aligned}
\text{Preprocess} &: O(n_1 n_0 d + \ell n_1 d \log d + T_{\text{Pre}}^{\text{LHE}}) \\
\text{Register}^S &: T_{\text{Eval}}^{\text{LHE}} \cdot \kappa \\
\text{Register}^C &: T_{\text{Qry}}^{\text{LHE}} \cdot \kappa + T_{\text{Dec}}^{\text{LHE}} \cdot \kappa \\
&\quad + O(\kappa n_0 d + \kappa n_1 d + \ell n_1 d \log d + \kappa n_1 \ell d) \\
\text{Query} &: O(n_0 d + \ell d \log d) \\
\text{Eval} &: n_1 (n_0 + \ell d) \\
\text{VerDec} &: O(\kappa (n_0 + \ell d + n_1) + n_1 \ell \log d + n_1)
\end{aligned}$$

³The response overhead can be mitigated by applying modulus switching.

Algorithm 4 vReinsPIRe

Setup: $\mathbf{A} \in \mathbb{Z}_q^{n_0 \times d}, [w_j(X)]_{j \in [\ell]}, [w'_j(X)]_{j \in [\ell]}$
 $\text{pp} \leftarrow \text{LHE.Setup}(1^\lambda)$

Server

Client

$\mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}$
 $\mathbf{H} \leftarrow \mathbf{D}\mathbf{A} \in \mathbb{Z}_q^{n_1 \times d}$
 $(\dots, \mathbf{H}', \dots) \leftarrow \text{ReinsPIRe.GenPack}(\mathbf{H}, [w_j], [w'_j])$
 $\text{sp} \leftarrow \text{LHE.Preprocess}(\text{pp}, \mathbf{D}^\top)$

H

$\xrightarrow{\mathbf{H}}$

$\mathbf{C} \leftarrow \{0, 1\}^{k \times n_1}$
 $(\text{qry} = \text{ct}(\mathbf{C}^\top), \text{st}) \leftarrow \text{LHE.Query}(\text{pp}, \mathbf{C}^\top)$

$\xleftarrow{\text{ct}(\mathbf{C}^\top)}$

$\text{ans} = \text{ct}(\mathbf{Z}^\top) \leftarrow \text{LHE.Eval}(\text{pp}, \text{sp}, \mathbf{D}^\top, \text{ct}(\mathbf{C}^\top))$

$\xrightarrow{\text{ct}(\mathbf{Z}^\top)}$

$\mathbf{Z}^\top \leftarrow \text{LHE.Dec}(\text{pp}, \text{st}, \text{ct}(\mathbf{Z}^\top)) \in \mathbb{Z}_q^{n_0 \times k}$
 Check that $\|\mathbf{Z}\|_\infty \leq n_1 p$
 and $\mathbf{Z}\mathbf{A} = \mathbf{C}\mathbf{H}$
 If the check fails, abort.
 $([a_i], \mathbf{H}', [t''_{i,j}]) \leftarrow \text{ReinsPIRe.GenPack}(\mathbf{H}, [w_j], [w'_j])$
 $\mathbf{Z}' \leftarrow \mathbf{C}\mathbf{H}' \in \mathbb{Z}_q^{k \times \ell d}$
 Client stores $[a_i], [t''_{i,j}], \mathbf{C}, \mathbf{Z}, \mathbf{Z}'$

$\mathbf{p} \leftarrow \text{Plaintext query vector} \in \mathbb{Z}_p^{n_0}$
 $(\mathbf{u}, [y'_j]) \leftarrow \text{ReinsPIRe.Query}(\mathbf{A}, [w_j], [w'_j], \mathbf{p})$
 Client stores $[y'_j]$ for decryption

$\xleftarrow{\mathbf{u}}$

$\mathbf{x} \leftarrow [\mathbf{D}, \mathbf{H}'] \cdot \mathbf{u}$

$\xrightarrow{\mathbf{x}}$

Check that $[\mathbf{Z}, \mathbf{Z}'] \cdot \mathbf{u} = \mathbf{C} \cdot \mathbf{x}$
 and abort otherwise.
 $\mathbf{z}_i \leftarrow \text{vec}(\sum_{j=0}^{\ell-1} t''_{i,j} \cdot y'_j)$ for $i \in [n_1/d]$
 $\mathbf{z} \leftarrow [\mathbf{z}_0^\top, \dots, \mathbf{z}_{n_1/d-1}^\top]^\top$
 Output $\text{ReinsPIRe.Dec}(\{a_i\}, \mathbf{x} + \mathbf{z})$

Theorem 7 (vReinsPIRe Noise). vReinsPIRe introduces no additional homomorphic noise beyond ReinsPIRe.

Theorem 8 (vReinsPIRe Communication). Let U^{LHE} and D^{LHE} be the upload and download cost of the underlying LHE scheme for evaluating $\mathbf{D}^\top \mathbf{C}^\top$. Excluding the CRS and public parameters:

- The per-client registration communication is U^{LHE} bits from the client to the server and $n_1 d \log_2 q + D^{\text{LHE}}$ bits from the server to the client.
- The online upload is $(n_0 + \ell d) \log_2 q$ bits and the online download is $n_1 \log_2 q$ bits.

Table 2: PIR performance metrics over a 1 GB database under two different settings for the server: storing the database and preprocessed material in memory or in SSD. Each protocol retrieves the payload size which produces the best performance.

	Metric	HintlessPIR	YPIR	VIA	VIA (blinded)	InsPIRe	ReinsPIRe
	Server Preprocessing Time (s)	211	32	2	2	110	152
	Client Generate Request (ms)	2,480	255	5.3	5.2	92	69
	Client Extract Response (ms)	69	0.455	0.026	0.003	0.476	1.9
	Client Upload (KB)	488	7,630	556	556	392	338
	Client Download (KB)	1,748	12	23	16	12	12
Server Data in memory	Persistent Memory Usage (GB)	1.4	4	8	8	1.9	1.2
	Server Response Time (ms)	755	462	2,099	2,135	331	173
Server Data in SSD	Server Long term Storage (GB)	3.3	4	8	8	2.5	1.2
	Server Response Time (ms)	2,884	468	2,122	2,138	313	175
	- Data Load Time (ms)	6,130	6,835	14,314	14,908	4,983	3,033
	- Compute Time (ms)	9,014	7,305	16,436	17,046	5,299	3,208

Theorem 9 (Informal vReinsPIRe Security). *Let $\mathcal{D}_h = \{\mathbf{D} \mid \mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}, \|\mathbf{D}\|_\infty \leq p\}$ and $\mathcal{D}_a = \{\mathbf{D} \mid \mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}, \|\mathbf{D}\|_\infty \leq 2n_1p\}$. Assuming key-dependent RLWE and SIS hardness, that the underlying LHE is query hiding, and that ReinsPIRe satisfies correctness on $\mathcal{D}_a$, then vReinsPIRe securely computes $\mathcal{F}_{\text{LHE}}$ (Figure 1) against a malicious server with abort.*

We defer formal statement and proof of Theorem 9 to Appendix B.2.

Honest Digest Assumption. At a high level, the *honest digest assumption* states that the digests produced by the server are all generated honestly, i.e. the hint matrix $\mathbf{H} = \mathbf{DA}$ for an honest database $\mathbf{D} \in \mathcal{D}_h$. We follow all prior works [29, 72] that also make the honest digest assumption in their experimental evaluations. Applying this assumption to our setting, since $\mathbf{H} = \mathbf{DA}$, we only require ReinsPIRe to be correct on the honest database class $\mathcal{D}_h$ instead of the admissible database class $\mathcal{D}_a$. This allows us to use more efficient SIS and RLWE parameters when using the building blocks of ReinsPIRe in vReinsPIRe (as done when instantiating prior verifiable works [29, 72]).

5 Experimental Evaluation

We present our evaluation in two sections. In Section 5.1, we compare ReinsPIRe with prior PIR protocols with server-side preprocessing. We perform our experiments in two settings, in-memory database and in-storage database, and show how ReinsPIRe is superior in both cases. Lastly, we demonstrate the use of in-memory PIR and in-storage PIR for password checkup over a sharded database, as proposed in the literature [49]. In Section 5.2, we compare vReinsPIRe with prior verifiable PIR protocols and measure the performance in the online and offline phase.

Implementation & Evaluation Details. We implement ReinsPIRe, ReinsPIRe⁺, and vReinsPIRe in C++ using SHELL [36] for sampling noise in our encryption and Google Highway [35] to support AVX512 SIMD hardware acceleration. We also incorporate additional optimizations in our implementations which are standard in FHE literature, which we discuss in Appendix E.1.

We perform all experimental evaluations on an Intel Xeon CPU @ 2.6 GHz machine running Debian, with support for AVX512, where we run all experiments in single-threaded mode for fair comparison with related work. Our implementation of our protocols is available on Github ⁴.

We augment each protocol with which we compare to report additional metrics that are discussed in this paper.

5.1 PIR Evaluation

For this section, we use ReinsPIRe with $d = 2048, q = 2^{54}, \sigma = 6.4, \ell = 2, z = 2^{19}$, which provides over 128-bit security, based on the lattice-estimator [2]. For other parameters such as the database shape and interpolation degree, we use parameters that provide the most competitive performance compared to related work. By adjusting parameters, we observe different

communication/computation tradeoffs, similar to related work, but due to limited space, we provide additional experiments using different parameters in Appendix E.

5.1.1 PIR with In-Memory Database

We first compare our protocol with all state-of-the-art protocols with server-side preprocessing such as HintlessPIR, YPIR, VIA, and InsPIRe. We also provide a more in-depth comparison, specifically with InsPIRe, in Appendix E.3 under identical parameters to highlight the algorithmic improvements. For this section, we use our most optimized variant of ReinsPIRe which incorporates all our techniques and uses the most optimized lattice parameters. In this experiment, we follow the implicit assumption made by prior work in which we store the entire database in memory, in preparation for queries made by clients.

Our results show improvement across online runtime, communication costs, and server storage, compared to all related work in this category. For example, we observe at least 50% improvement in runtime compared to every protocol in the literature. Our communication costs are also less than all those in related work. Lastly, our offline preprocessed material is also at least 50% smaller than all related work in the literature, which translates to less peak RAM usage and less cost. In the next section, we also see the effect that this has on runtime when we read data from external memory.

5.1.2 In-Storage PIR with Sharded Databases

We also perform an experiment while reading the database from external memory (SSD) and include that as part of the total runtime. This experiment simulates sharded PIR, where the specific database to query is determined by the user as part of the query.

While the database size is identical for all protocols, the size of preprocessing material affects the runtime since it is comparable to the size of the database. Moreover, protocols such as VIA store the database in a large NTT format to reduce the response time. Hence, we report the total memory usage of the protocol (stored database size + stored preprocessed material). For the purpose of experiments, we modify all related work and our work as well, to simulate a scenario where the data and preprocessed material is read from SSD. Following the offline preprocessing phase, the protocols serializes the database and preprocessed material, writing them contiguously to a binary file on SSD.

To accurately measure execution behavior and simulate a real-world server environment—where subsequent client queries target different databases—the protocol actively defeats Operating System memory caching. Prior to issuing responds to queries, the host environment explicitly flushes pending writes and issues a cache eviction directive to the Linux kernel (POSIX_FADV_DONTNEED). This guarantees the file pages are physically evicted from the OS page cache (DRAM). During the online query phase, the protocol pre-allocates an aligned memory buffer and sequentially executes a bulk user-space read to ingest the entire target database. By forcing this strict cache eviction policy, the experimental framework guarantees a true "cold-start" environment, measuring the authentic physical latency of streaming block data across the PCIe storage bus rather than artificially fast reads from kernel memory caches.

In this experiment, the server response time consists of two components. First, the time required to load the data into memory and second, the time to perform the computation. We also report the long term storage that is used for each protocol in the experiments, which is the actual physical SSD required for each protocol. Note that the practical storage required for each protocol is usually larger than the theoretical size of the preprocessed material, since data is stored in bit aligned format to accelerate subsequent computations.

Our experiments demonstrate the suitability of ReinsPIRe for in-storage PIR due to much smaller preprocessing material. For server response latency, ReinsPIRe is at least 2x faster than all related work due to smaller preprocessing material.

5.1.3 PIR for Password Checkup

We demonstrate the use of PIR for password checkup for a 4 billion leaked credentials, which is the size of prominent leaked credential datasets. Prior works [3, 49, 74] have proposed protocols which split the database into smaller databases, based on the prefix of the hash of the passwords and only performing checks in one of the shards. As the database is large, we can not store all databases in memory at the same time because it would require very high memory usage. So, we store the databases in SSD and retrieve the specific database on the fly, based on the users query. We generate 4 billion synthetic credentials and experiment with performing leaked credential checks when splitting the database into 1, 2, 4, 8, and 16 shards. We map each credential to one of the rows of one of the databases using its hash prefix. We defer further improvements on the mapping mechanism to future work. Table 3 shows the results of this experiment.

⁴<https://github.com/google/InsPIRe>

5.2 vPIR evaluation

For our experimental evaluation of verifiable PIR, we compare with state-of-the-art protocols, VeriSimplePIR [29] and vPIR [72]. For all works, we experiment under the honest digest assumption, meaning that we assume H (and H') are generated correctly.

We instantiate vReinsPIRe with ReinsPIRe as the LHE protocol for both the online and offline phase with the following parameters:

Offline	$d = 2048$	$q = 2^{54}$	$p = 2^{24}$	$\sigma = 6.4$	$\ell = 5$	$z = 2^{11}$
Online	$d = 2048$	$q = 2^{32}$	$p = 2^8$	$\sigma = 3.2$	$\ell = 3$	$z = 2^{19}$

Table 4 shows a summary of our results. Compared to VeriSimplePIR, we see improvements in metrics such as 2.4x less offline communication, equal online communication, and 50x less client storage. In comparison to vPIR, 3.2x in offline communication, and 15x improvement in online communication.

Overall, vReinsPIRe is the best solution when minimizing communication costs, both offline and online, at a small additional cost in computation.

6 Related Works

Early PIR Schemes. PIR was introduced by Chor *et al.* [23] in the multi-server setting while single-server PIR was introduced by Kushilevitz and Ostrovsky [45]. The majority of early single-server PIR schemes were built using linearly homomorphic encryption schemes based on number-theoretic assumptions (see [17, 27, 34, 45, 67, 68] and references therein).

PIR with Offline Communication. A line of PIR schemes built from RLWE have been presented that require offline communication including [1, 3, 4, 16, 60, 61, 64]. These constructions obtain low communication due to the succinctness of RLWE ciphertexts, but they require large computational costs (and, thus, low throughput) along with the server storing a cryptographic key for each client that must be communicated prior to any query. Another line of work has built PIR from [28, 41] that obtain high throughput (almost at the level of two-server PIR schemes [23]) at the cost of high communication. Additionally, the client must receive and store a database-specific hint to issue any PIR queries.

Another line of work has considered PIR with client-stored database hints that obtain sublinear query times including [25, 26, 42, 44, 70, 75]. Recent works [47, 73, 78] present concretely efficient constructions, but they all require downloading the entire database before any query.

PIR with No Offline Communication. To avoid the challenges of offline communication, recent works have aimed at constructing efficient PIR without offline communication including [40, 48, 53, 58, 62]. At a high level, these constructions build on top of high-throughput LWE-based PIR schemes [28, 41] and apply RLWE techniques to remove offline communication and reduce online query communication.

Additionally, we note a recent theoretical breakthrough work of Lin *et al.* [52] presented a PIR construction with sublinear query time with server-side preprocessing, but the concrete efficiency remains impractical [66].

In-Storage and Sharded PIR. In-storage PIR considers the setting where large databases no longer fit into memory and must be placed into secondary storage. A practical example is password leak check where the database contains billions of entries [74].

Table 3: Password checkup over 4 Billion passwords, using varying number of subdatabases, using In-memory PIR and In-Storage PIR. The total database size is roughly 16 GB.

Num Databases	16	8	4	2	1
Each Database Size	2 GB	3 GB	5 GB	9 GB	16 GB
In-Memory					
Latency	477 ms	603 ms	1,036 ms	1,769 ms	3,119 ms
Peak Memory Usage	19.2 GB	19.2 GB	19.2 GB	19.2 GB	19.2 GB
In-Storage					
Latency	7.7 s	9.5 s	15.2 s	24.2 s	40.4 s
Peak Memory Usage	2.4 GB	3.6 GB	6.0 GB	10.8 GB	19.2 GB

Table 4: Experimental Evaluation of VerifiablePIR. (S) and (C) the runtime of the Server and Client, respectively. (C2S) and (S2C) denotes messages sent from client to server and server to client, respectively.

Metric	VeriSimplePIR	vPIR	vReinsPIRe
Offline Phase			
(S) Global Preprocessing	21.9s	17.6m	8.4m
(S) Proof Preprocessing	4.6s	10.4s	13.7s
(C) Proof Preprocessing	N/A	1.9m	36.2s
(C2S) Proof Preprocessing	7 MB	10 MB	8 MB
(S2C) Hint Download	649 MB	880 MB	256 MB
(S2C) Proof Download	10 MB	10 MB	10 MB
(C) Long term Storage	307 MB	256 KB	6 MB
Online Phase			
(C) Generate Request	112.2 ms	2.1 ms	51.6 ms
(S) Response Time	136.2 ms	209.8 ms	299.7 ms
(C) Verification	3.7 ms	2.4 ms	5.5 ms
(C) Extract response	46.0 ms	0.094 ms	45.2 ms
(C2S) Upload	136 KB	1 MB	160 KB
(S2C) Download	154 KB	2 MB	128 KB

To make PIR queries feasible, these systems split the large database into many shards and execute PIR only on a single shard such as for password leak check [3] deployed by Microsoft [46] and image search deployed by Apple [8].

Recent works including [20, 51] consider hardware-level optimizations for in-storage PIR using computational processing in the secondary storage systems themselves to avoid large bandwidth from storage back into memory.

Verifiable PIR. Verifiable PIR consists of the problem that enables clients to detect improper server behavior and has been studied in prior works including [7, 13, 24, 29, 31, 32, 72].

PIR with Other Features. PIR has also been studied with other features including batch PIR that enables retrieving multiple entries in a single query [4, 14, 39, 55, 65, 76], keyword PIR that enables retrieving from sparse databases represented using keys and values [18, 57, 71] along with symmetric PIR that also protects the server’s database [3, 50].

Ring Packing. Ring packing enables transforming (M)LWE to RLWE ciphertexts. The problem has been studied as a standalone task [12, 15, 19, 22, 43, 63] along as a key component in PIR schemes that avoid offline communication [53, 58, 62]. The matrix transposition algorithm in [69] can generalize to ring packing by considering only the first column.

Encrypted Linear Transformations. A line of work [10, 21, 54, 69] proposes encrypted linear transformation algorithms that reduce to plain linear algebra. While these works share the same theme as our work, the techniques proposed in these papers do not directly translate to efficient PC-Mv (plaintext matrix ciphertext vector multiplication) in the CRS setting where a fresh secret key is used for each query. [11], which consolidates and extends [10, 69], proposes a generic framework for encrypted linear transformations including PC-Mv. While the online query processing reduces to plain matrix-vector multiplications, it relies on an initial query independent, but client key dependent step that uses the matrix transposition algorithm of [69], which can be expensive in the CRS setting where a fresh secret key is used for each query.

References

- [1] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Addra: Metadata-private voice communication over fully untrusted infrastructure. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 313–329. USENIX Association, July 2021.
- [2] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of Learning with Errors, 2015. Publication info: Published elsewhere. Minor revision. Journal of Mathematical Cryptology.

- [3] Asra Ali, Tancrede Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication-computation trade-offs in PIR. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1811–1828. USENIX Association, August 2021.
- [4] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979. IEEE Computer Society Press, May 2018.
- [5] Apple. Filter content for apple devices. <https://support.apple.com/guide/deployment/filter-content-dep1129ff8d2/web>.
- [6] Apple. Getting up-to-date calling and blocking information for your app. https://developer.apple.com/documentation/sms_and_call_reporting/getting_up-to-date_calling_and_blocking_information_for_your_app, 2024.
- [7] Pranav Shriram Arunachalam and Ling Ren. Single-server stateful pir with verifiability and balanced efficiency. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 1385–1402, 2026.
- [8] Hilal Asi, Fabian Boemer, Nicholas Genise, Muhammad Haris Mughees, Tabitha Ogilvie, Rehan Rishi, Kunal Talwar, Karl Tarbe, Akshay Wadia, Ruiyu Zhu, and Marco Zuliani. Wally: Batched private nearest neighbor search at scale, 2026.
- [9] Ali Atiia and Keewoo Lee. Sharded pir design for the ethereum state. <https://ethresear.ch/t/sharded-pir-design-for-the-ethereum-state/>, 2026.
- [10] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. Plaintext-ciphertext matrix multiplication and the bootstrapping: fast and fused. In *Annual International Cryptology Conference*, pages 387–421. Springer, 2024.
- [11] Youngjin Bae, Jung Hee Cheon, Guillaume Hanrot, Jai Hyun Park, and Damien Stehlé. Fast homomorphic linear algebra with blas. *Journal of Cryptology*, 39(3):25, 2026.
- [12] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. HERMES: Efficient ring packing using MLWE ciphertexts and application to transciphering. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 37–69. Springer, Cham, August 2023.
- [13] Shany Ben-David, Yael Tauman Kalai, and Omer Paneth. Verifiable private information retrieval. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part III*, volume 13749 of *LNCS*, pages 3–32. Springer, Cham, November 2022.
- [14] Alexander Bienstock, Sarvar Patel, Joon Young Seo, and Kevin Yeo. Batch PIR and labeled PSI with oblivious ciphertext compression. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.
- [15] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.
- [16] Alexander Burton, Samir Jordan Menon, and David J. Wu. Respire: High-rate PIR for databases with small records. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1463–1477. ACM Press, October 2024.
- [17] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *EUROCRYPT’99*, volume 1592 of *LNCS*, pages 402–414. Springer, Berlin, Heidelberg, May 1999.
- [18] Sofia Celi and Alex Davidson. Call me by my name: Simple, practical private information retrieval for keyword queries. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 4107–4121. ACM Press, October 2024.
- [19] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In Kazuo Sako and Nils Ole Tippenhauer, editors, *Applied Cryptography and Network Security*, volume 12726, pages 460–479. Springer International Publishing, Cham, 2021.
- [20] Zehao Chen, Honghui You, Qian Wei, Hang Lu, Lei Ju, and Zhaoyan Shen. Smartpir: A private information retrieval system using computational storage devices. In *Proceedings of the 58th IEEE/ACM International Symposium on Microarchitecture, MICRO ’25*, page 1749–1762, New York, NY, USA, 2025. Association for Computing Machinery.
- [21] Jung Hee Cheon, Minsik Kang, and Junho Lee. Fast batch matrix multiplication in ciphertexts. *Cryptology ePrint Archive*, 2025.
- [22] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 377–408. Springer, Cham, December 2017.
- [23] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995.
- [24] Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J. Wu, and Bryan Ford. Authenticated private information retrieval. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 3835–3851. USENIX Association, August 2023.
- [25] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 3–33. Springer, Cham, May / June 2022.
- [26] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 44–75. Springer, Cham, May 2020.

- [27] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier’s probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Berlin, Heidelberg, February 2001.
- [28] Alex Davidson, Gonçalo Pestana, and Sofia Celi. FrodoPIR: Simple, scalable, single-server private information retrieval. *PoPETs*, 2023(1):365–383, January 2023.
- [29] Leo de Castro and Keewoo Lee. VeriSimplePIR: Verifiability in SimplePIR at no online cost for honest servers. In Davide Balzarotti and Wenyan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.
- [30] Leo de Castro, Kevin Lewi, and Edward Suh. WhisPIR: Stateless private information retrieval with low communication. *Cryptology ePrint Archive*, Paper 2024/266, 2024.
- [31] Marian Dietz and Stefano Tessaro. Fully malicious authenticated PIR. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IX*, volume 14928 of *LNCS*, pages 113–147. Springer, Cham, August 2024.
- [32] Brett Falk, Pratyush Mishra, and Matan Shtepel. Malicious security for PIR (almost) for free. In *CRYPTO 2025, Part VIII*, *LNCS*, pages 175–210. Springer, Cham, August 2025.
- [33] Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP congress*, volume 839, page 842, 1977.
- [34] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005*, volume 3580 of *LNCS*, pages 803–815. Springer, Berlin, Heidelberg, July 2005.
- [35] Google. Highway: Performance-portable, length-agnostic SIMD with runtime dispatch. <https://github.com/google/highway>. Accessed: August 26, 2026.
- [36] Google. SHELL: Simple homomorphic encryption library with lattices. <https://github.com/google/shell-encryption>. Accessed: February 4, 2025.
- [37] Shai Halevi and Victor Shoup. Algorithms in helib. In *Annual Cryptology Conference*, pages 554–571. Springer, 2014.
- [38] Shai Halevi and Victor Shoup. Faster homomorphic linear transformations in helib. In *Annual International Cryptology Conference*, pages 93–120. Springer, 2018.
- [39] Ryan Henry. Polynomial batch codes for efficient IT-PIR. *PoPETs*, 2016(4):202–218, October 2016.
- [40] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nikolai Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th symposium on operating systems principles*, pages 396–416, 2023.
- [41] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 3889–3905. USENIX Association, August 2023.
- [42] Alexander Hoover, Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Plinko: Single-server PIR with efficient updates via invertible PRFs. *LNCS*, pages 3–33. Springer, Cham, June 2025.
- [43] Wenjie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. PEGASUS: Bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy*, pages 1057–1073. IEEE Computer Society Press, May 2021.
- [44] Dmitry Kogan and Henry Corrigan-Gibbs. Private blacklist lookups with checklist. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 875–892. USENIX Association, August 2021.
- [45] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997.
- [46] Kristin Lauter, Sreekanth Kannepalli, Kim Laine, and Radames Cruz Moreno. Password Monitor: Safeguarding passwords in Microsoft Edge. <https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/>, 2021.
- [47] Arthur Lazzaretti and Charalampos Papamanthou. Single pass client-preprocessing private information retrieval. In Davide Balzarotti and Wenyan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.
- [48] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz. Hintless single-server private information retrieval. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part IX*, volume 14928 of *LNCS*, pages 183–217. Springer, Cham, August 2024.
- [49] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. Protocols for checking compromised credentials. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1387–1403. ACM Press, November 2019.
- [50] Chengyu Lin, Zeyu Liu, and Tal Malkin. XSPIR: Efficient symmetrically private information retrieval from ring-LWE. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part I*, volume 13554 of *LNCS*, pages 217–236. Springer, Cham, September 2022.

- [51] Jilan Lin, Ling Liang, Zheng Qu, Ishtiyaque Ahmad, Liu Liu, Fengbin Tu, Trinabh Gupta, Yufei Ding, and Yuan Xie. Inspire: in-storage private information retrieval via protocol and architecture co-design. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ISCA '22, page 102–115, New York, NY, USA, 2022. Association for Computing Machinery.
- [52] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In Barna Saha and Rocco A. Servedio, editors, *55th ACM STOC*, pages 595–608. ACM Press, June 2023.
- [53] Chenyang Liu, Xukun Wang, and Zhifang Zhang. VIA: Communication-Efficient Single-Server Private Information Retrieval. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 1503–1521, Los Alamitos, CA, USA, May 2026. IEEE Computer Society.
- [54] Jing Liu and Liang Feng Zhang. Privacy-preserving and publicly verifiable matrix multiplication. *IEEE Transactions on Services Computing*, 16(3):2059–2071, 2022.
- [55] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015*, volume 8975 of *LNCS*, pages 168–186. Springer, Berlin, Heidelberg, January 2015.
- [56] Ming Luo, Feng-Hao Liu, and Han Wang. Faster FHE-based single-server private information retrieval. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1405–1419. ACM Press, October 2024.
- [57] Rasoul Akhavan Mahdavi and Florian Kerschbaum. Constant-weight PIR: Single-round keyword PIR via constant-weight equality operators. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 1723–1740. USENIX Association, August 2022.
- [58] Rasoul Akhavan Mahdavi, Sarvar Patel, Joon Young Seo, and Kevin Yeo. InsPIRe: Communication-efficient PIR with server-side preprocessing. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 1463–1482. IEEE Computer Society, 2026.
- [59] Miti Mazmudar, Shannon Veitch, and Rasoul Akhavan Mahdavi. Peer2PIR: Private queries for IPFS. In *2025 IEEE Symposium on Security and Privacy*, pages 4438–4456. IEEE Computer Society Press, May 2025.
- [60] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. Xpir: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, pages 155–174, 2016.
- [61] Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy*, pages 930–947. IEEE Computer Society Press, May 2022.
- [62] Samir Jordan Menon and David J. Wu. YPIR: High-throughput single-server PIR with silent preprocessing. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.
- [63] Daniele Micciancio and Jessica Sorrell. Ring packing and amortized FHEW bootstrapping. In Ioannis Chatzigiannakis, Christos Kaklamanis, Daniel Marx, and Donald Sannella, editors, *ICALP 2018*, volume 107 of *LIPICs*, pages 100:1–100:14. Schloss Dagstuhl, July 2018.
- [64] Muhammad Haris Mughees, Hao Chen, and Ling Ren. OnionPIR: Response efficient single-server PIR. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2292–2306. ACM Press, November 2021.
- [65] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *2023 IEEE Symposium on Security and Privacy*, pages 437–452. IEEE Computer Society Press, May 2023.
- [66] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. Towards practical doubly-efficient private information retrieval. *LNCS*, pages 264–282. Springer, Cham, March 2024.
- [67] Rafail Ostrovsky and William E. Skeith, III. A survey of single-database private information retrieval: Techniques and applications (invited talk). In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 393–411. Springer, Berlin, Heidelberg, April 2007.
- [68] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Berlin, Heidelberg, May 1999.
- [69] Jai Hyun Park. Ciphertext-ciphertext matrix multiplication: Fast for large matrices. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 153–180. Springer, 2025.
- [70] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1002–1019. ACM Press, October 2018.
- [71] Sarvar Patel, Joon Young Seo, and Kevin Yeo. Don't be dense: Efficient keyword PIR for sparse databases. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 3853–3870. USENIX Association, August 2023.
- [72] Mayank Rathee, Keewoo Lee, and Raluca Ada Popa. Verifiable PIR with Small Client Storage. In *2026 IEEE Symposium on Security and Privacy (SP)*, pages 1423–1442, Los Alamitos, CA, USA, May 2026. IEEE Computer Society.
- [73] Ling Ren, Muhammad Haris Mughees, and I Sun. Simple and practical amortized sublinear private information retrieval using dummy subsets. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 1420–1433. ACM Press, October 2024.

- [74] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019*, pages 1556–1571. USENIX Association, August 2019.
- [75] Zhikun Wang and Ling Ren. Single-server client preprocessing PIR with tight space-time trade-off. *LNCS*, pages 94–122. Springer, Cham, June 2025.
- [76] Kevin Yeo. Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 197–230. Springer, Cham, August 2023.
- [77] Kevin Yeo and Sarvar Patel. Protecting your device information with private set membership. <https://security.googleblog.com/2021/10/protecting-your-device-information-with.html>, 2021.
- [78] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. Piano: Extremely simple, single-server PIR with sublinear server computation. In *2024 IEEE Symposium on Security and Privacy*, pages 4296–4314. IEEE Computer Society Press, May 2024.

A LHE and vLHE Preliminaries

A.1 LHE and vLHE Syntax

Database and Query Classes. Fix n_1 and n_0 . For our definitions of LHE and vLHE below, we will consider two classes of databases $\mathcal{D}_h$ and $\mathcal{D}_a$ where $\mathcal{D}_h \subseteq \mathcal{D}_a \subseteq \mathbb{Z}^{n_1 \times n_0}$. $\mathcal{D}_h$ consists of honest databases (used in the honest execution) and $\mathcal{D}_a$ consists of admissible databases that malicious servers are bound to (in the malicious execution). Finally, $Q \subseteq \mathbb{Z}^{n_0}$ is the message space. The goal of LHE is to securely compute $\mathbf{D} \cdot \mathbf{p} \bmod p$ for $\mathbf{D} \in \mathcal{D}_h$ (or $\mathcal{D}_a$ in the malicious setting) and $\mathbf{p} \in Q$.

LHE. We consider single-server Linear Homomorphic Encryption (LHE) with server preprocessing with single roundtrip queries consisting of the tuple of efficient algorithms $\text{LHE} = (\text{Setup}, \text{Preprocess}, \text{Query}, \text{Eval}, \text{Dec})$:

- $\text{pp} \leftarrow \text{Setup}(1^\lambda)$: A trusted party outputs CRS and other public parameters and distributes to the server and the client.
- $\text{sp} \leftarrow \text{Preprocess}(\text{pp}, \mathbf{D})$: The server preprocesses the database and creates server preprocessed materials sp .
- $(\text{st}, \text{qry}) \leftarrow \text{Query}(\text{pp}, \mathbf{p})$: The client receives query vector $\mathbf{p}$ and produces a local state and a query for the server.
- $\text{resp} \leftarrow \text{Eval}(\text{pp}, \text{sp}, \mathbf{D}, \text{qry})$: The server executes the query against the database to produce a response.
- $\text{ans} \leftarrow \text{Dec}(\text{pp}, \text{st}, \text{resp})$: The client decrypts the response to obtain the final output of the LHE protocol.

For correctness, the client should obtain $\text{ans} = \mathbf{D} \cdot \mathbf{p} \bmod p$ with probability at least $1 - \text{negl}(\lambda)$ over the internal randomness of the LHE protocol with negligible error, for all $\mathbf{D} \in \mathcal{D}_h$.

In verifiable LHE scheme in Section 4, we extend the syntax to apply to plaintext query matrix $\mathbf{P} \in \mathbb{Z}^{n_0 \times t}$. It is assumed the syntax above is applied to each column of $\mathbf{P}$ individually.

Verifiable LHE. We extend the notion of LHE and consider single-server Verifiable LHE, which consists of tuple of algorithms $\text{vLHE} = (\text{Setup}, \text{Preprocess}, \text{Register} = \{(\text{Register}_j^S, \text{Register}_j^C)\}_{j \in [k]}, \text{Query}, \text{Eval}, \text{VerDec})$:

- $\text{pp} \leftarrow \text{Setup}(1^\lambda)$: A trusted party outputs CRS and other public parameters and distributes to the server and the client.
- $\text{sp} \leftarrow \text{Preprocess}(\text{pp}, \mathbf{D})$: The server preprocesses the database and creates server preprocessed materials sp .
- Register is a k -round stateful protocol (with client aborts) between the client and the server, starting with the server:
 - $(\text{msg}_i^S, \text{st}_i^S) \leftarrow \text{Register}_i^S(\text{pp}, \text{sp}, \mathbf{D}, \text{msg}_{i-1}^C, \text{st}_{i-1}^S)$
 - $(\text{msg}_i^C, \text{st}_i^C) / \text{Abort} \leftarrow \text{Register}_i^C(\text{pp}, \text{msg}_i^S, \text{st}_{i-1}^C)$

where $\text{msg}_{-1}^C = \text{st}_{-1}^S = \text{st}_{-1}^C = \perp$.

- $(\text{st}, \text{qry}) \leftarrow \text{Query}(\text{pp}, \mathbf{p}, \text{st}_{k-1}^C)$: The client receives query vector $\mathbf{p}$ and produces a local state and a query for the server using the last client state from Register.
- $\text{resp} \leftarrow \text{Eval}(\text{pp}, \text{sp}, \mathbf{D}, \text{qry})$: The server executes the query against the database to produce a response.
- $\text{ans} / \text{Abort} \leftarrow \text{VerDec}(\text{pp}, \text{st}, \text{resp}, \text{st}_{k-1}^C)$: The client verifies and decrypts the response to obtain the final output of the LHE protocol.

In the honest execution, the client should obtain $\text{ans} = \mathbf{D} \cdot \mathbf{p} \bmod p$ with probability at least $1 - \text{negl}(\lambda)$ for all $\mathbf{D} \in \mathcal{D}_h$.

A.2 LHE and vLHE Security Definitions

We expand on the security definitions in [72] to generalize to semi-honest LHE.

Definition 1 (Input Transcript Family [72], modified). *Given message space $Q \subseteq \mathbb{Z}^{n_0}$ and $\lambda \in \mathbb{N}$, define $I_{Q,\lambda}$ as the family of sequences with k instructions per sequence, where each instruction is one of the following two types:*

- Register : Register instruction is of the form $(\text{id}, \text{Register})$, denoting the id of the client who registers for the first time.
- Query : Query instruction is of the form $(\text{id}, \text{Query}, \mathbf{p})$ denoting the client's id who queries with input $\mathbf{p} \in Q$.

where $I_{Q,\lambda} = \bigcup_{k \in \mathbb{Z}_{\text{poly}(\lambda)}} I_{Q,\lambda,k}$. Finally, we define $I_{Q,\lambda}^{\text{sh}}$ as a subset of $I_{Q,\lambda}$ without Register command. $I_{Q,\lambda}^v$ is a subset such that for each id that appears in the sequence, $(\text{id}, \text{Register})$ appears exactly once before any $(\text{id}, \text{Query}, \mathbf{p})$.

Definition 2 (Real and ideal executions [72]). *Given a functionality $\mathcal{F}$, a protocol π , non-uniform polynomial time machine $\mathcal{A}$ and S , input X , auxiliary input z and security parameter λ :*

- The ideal execution of $\mathcal{F}$ with adversary S , denoted by $\text{IDEAL}_{\mathcal{F},S(z)}(X,\lambda)$, is defined as the set of outputs returned by $\mathcal{F}$ along with outputs of S .
- The real execution of π with the adversary $\mathcal{A}$ and an implicit set of honest clients $\mathcal{C}$, denoted by $\text{REAL}_{\pi,\mathcal{A}(z)}(X,\lambda)$, is defined as the set of outputs returned by the protocol π (to honest parties) along with the outputs of $\mathcal{A}$.

Definition 3 (LHE and vLHE). *Let $X = (\mathbf{D}_{\text{init}}, T)$ where $\mathbf{D}_{\text{init}} \in \mathcal{D}_h$ is an initial database and T is the input transcript, and $\mathcal{F}_{\text{LHE}}$ be the ideal functionality in Figure 1.*

- We say that LHE = (Setup, Preprocess, Query, Eval, Dec) securely computes $\mathcal{F}_{\text{LHE}}$ in the presence of honest clients and semi-honest server if for all non-uniform probabilistic polynomial time semi-honest adversary $\mathcal{A}$, there exists a non-uniform polynomial time S such that the following holds:

$$\begin{aligned} \{\text{IDEAL}_{\mathcal{F},S(\mathbf{D}_{\text{init}};z)}(X,\lambda)\}_{X \in \mathcal{D}_h \times I_{Q,\lambda}^{\text{sh}}, z \in \{0,1\}^*, \lambda \in \mathbb{N}} &= \lambda \\ \{\text{REAL}_{\text{LHE},\mathcal{A}(\mathbf{D}_{\text{init}};z)}(X,\lambda)\}_{X \in \mathcal{D}_h \times I_{Q,\lambda}^{\text{sh}}, z \in \{0,1\}^*, \lambda \in \mathbb{N}} & \end{aligned}$$

- We say that vLHE = (Setup, Preprocess, Register, Query, Eval, VerDec) securely computes $\mathcal{F}_{\text{LHE}}$ (with abort) in the presence of honest clients and malicious server if the above condition holds for protocol vLHE, malicious $\mathcal{A}$, and input transcript family $I_{Q,\lambda}^v$.

Finally, we define a protocol that securely computes $\mathcal{F}_{\text{LHE}}$ in each of the two settings above as LHE and vLHE scheme, respectively.

Role of $\mathbf{D}_{\text{init}}$. Intuitively, $\mathbf{D}_{\text{init}}$ is the initial reference database that honest server uses. Correctness/completeness is defined with respect to $\mathbf{D}_{\text{init}}$, but the malicious security does not require $\mathbf{D}_{\text{init}}$ to be binding. In particular, the adversary may send any $\mathbf{D} \in \mathcal{D}_a$ to $\mathcal{F}_{\text{LHE}}$ in Register, and the only requirement is that all subsequent responses are consistent with the same $\mathbf{D}$.

B LHE and VLHE Security Proofs

B.1 LHE Security Proof

Definition 4 (Key-dependent RLWE Hardness). *$(d, q, \chi_s, \chi_e, k_s, k_o, \mathcal{F})$ -KDRLWE is a decision RLWE problem with degree d , modulus q , secret key distribution χ_s , and error distribution χ_e , with the number of standard RLWE samples k_s . Additionally, the adversary has access to k_o oracle queries for encryptions of $f \in \mathcal{F}$. Informally, we say that $(d, q, \chi_s, \chi_e, k_s, k_o, \mathcal{F})$ -KDRLWE is hard if the adversary's advantage in distinguishing between random samples (resp. random oracle queries) and RLWE samples (resp. oracle queries for encryptions of $\mathcal{F}$) is at most $\text{negl}(\lambda)$.*

Remark. We note that standard RLWE hardness is not enough in our work since the protocol requires encrypting its own secret keys (for key switching keys). However, these additional assumptions are generally common in lattice based cryptography literature.

Theorem 10 (Security of ReinsPIRe). *Let λ be the security parameter. Let $\mathcal{D}_h = \{\mathbf{D} \mid \mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}, \|\mathbf{D}\|_\infty \leq p\}$, and suppose that $(d, q, \chi_s, \chi_e, k_s, k_o, \mathcal{F}_{\text{auto}})$ -KDRLWE is hard for $k_s = n_0$, $k_o = 2\ell$ and $\mathcal{F}_{\text{auto}} = \{z \cdot \tau(\tilde{s})\}$. Suppose further that ReinsPIRe is correct except with probability $\text{negl}(\lambda)$ for every $\mathbf{D} \in \mathcal{D}_h$. Then ReinsPIRe securely computes $\mathcal{F}_{\text{LHE}}$ against a semi-honest server according to Definition 3.*

Proof sketch. The proof largely follows the same structure as [58], but ported to simulator based proof. The simulator in this case is simple: upon query, the simulator interacts with the adversary $\mathcal{A}$ using a dummy query $\mathbf{p}$ (say 0 vector). For simplicity, suppose that the length of the query transcript $I_{Q,\lambda}^{\text{sh}}$ is 1. We can define a sequence of hybrids replacing real queries with queries encrypting 0, using the KDRLWE hardness assumptions above. This establishes that the real and ideal experiments are indistinguishable except with probability $\text{negl}(\lambda)$. For general $I_{Q,\lambda}^{\text{sh}}$, since the length is $\text{poly}(\lambda)$, we can use another hybrid argument to conclude that the experiment is indistinguishable except with probability $\text{negl}(\lambda)$. $\square$

B.2 vLHE Security Proof

Definition 5 (SIS Hardness). *Given a random $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, (m, n, q, β) -SIS is a problem of finding a vector $\mathbf{s} \in \mathbb{Z}^n$ such that $0 < \|\mathbf{s}\|_\infty \leq \beta$ and $\mathbf{A}\mathbf{s} \bmod q = \mathbf{0}$ holds. Informally, we say that (m, n, q, β) -SIS is hard if the adversary's advantage in finding such $\mathbf{s}$ is at most $\text{negl}(\lambda)$.*

Lemma 7 (Null space lemma [29]). *Let $m, n > 0$ and $q > 0$, and fix any non-zero vector $\mathbf{x} \in \mathbb{Z}_q^n$. Then the following holds:*

$$\Pr_{\mathbf{C} \leftarrow \{0,1\}^{k \times n}} [\mathbf{C} \cdot \mathbf{x} = \mathbf{0}] \leq 2^{-k}$$

Lemma 8 (Main lemma). *Suppose that the registration LHE is query-hiding against PPT servers. If the server passes the registration phase with non-negligible probability and all checks in Algorithm 4 pass, then assuming $(d, n_0, q, (2n_1 + 1)n_1p)$ -SIS is hard, the following hold except with negligible probability:*

- There exists an efficiently extractable $\tilde{\mathbf{D}} \in \mathbb{Z}^{n_1 \times n_0}$ such that $\|\tilde{\mathbf{D}}\|_\infty \leq 2n_1p$ and $\mathbf{H} = \tilde{\mathbf{D}}\mathbf{A}$.
- $\mathbf{x} = [\tilde{\mathbf{D}}, \mathbf{H}']\mathbf{u} \bmod q$

Proof. By the knowledge-soundness property of the extractable SIS-based commitment scheme recalled in Section 4, a server that passes registration with non-negligible probability can be rewound to extract a matrix $\tilde{\mathbf{D}}$ such that $\|\tilde{\mathbf{D}}\|_\infty \leq 2n_1p$ and $\mathbf{H} = \tilde{\mathbf{D}}\mathbf{A}$. Furthermore, the extractor is run using independent challenges from $\mathbf{C}$ and so $\tilde{\mathbf{D}}$ is independent of $\mathbf{C}$.

Because the registration check passes, $\mathbf{Z}\mathbf{A} = \mathbf{C}\mathbf{H} = \mathbf{C}\tilde{\mathbf{D}}\mathbf{A}$, and hence $(\mathbf{Z} - \mathbf{C}\tilde{\mathbf{D}})\mathbf{A} = \mathbf{0}$. We also have $\|\mathbf{Z}\|_\infty \leq n_1p$ since the check passed. Furthermore, $\|\mathbf{C}\tilde{\mathbf{D}}\|_\infty \leq n_1\|\tilde{\mathbf{D}}\|_\infty \leq 2n_1^2p$ and hence $\|\mathbf{Z} - \mathbf{C}\tilde{\mathbf{D}}\|_\infty \leq (2n_1 + 1)n_1p$. If $\mathbf{Z} \neq \mathbf{C}\tilde{\mathbf{D}}$, then a non-zero row of $\mathbf{Z} - \mathbf{C}\tilde{\mathbf{D}}$ gives a solution to the stated SIS problem, which implies that $\mathbf{Z} = \mathbf{C}\tilde{\mathbf{D}}$ except with negligible probability.

The client computes $\mathbf{H}'$ locally from $\mathbf{H}$ and the CRS, and then computes $\mathbf{Z}' \leftarrow \mathbf{C}\mathbf{H}'$. Let the server's online response be $\mathbf{x} = [\tilde{\mathbf{D}}, \mathbf{H}']\mathbf{u} + \delta$ for some $\delta \in \mathbb{Z}_q^{n_1}$. If the online verification accepts, then $[\mathbf{Z}, \mathbf{Z}']\mathbf{u} = \mathbf{C}\mathbf{x}$. This implies that $\mathbf{C}[\tilde{\mathbf{D}}, \mathbf{H}']\mathbf{u} = \mathbf{C}\mathbf{x}$, and substituting $\mathbf{x}$ gives $\mathbf{C}\delta = \mathbf{0}$.

It remains to show that this implies $\delta = \mathbf{0}$ except with negligible probability. The server does not receive $\mathbf{C}$, $\mathbf{Z}$, or $\mathbf{Z}'$, and its only challenge-dependent input is the query-hiding LHE encryption of $\mathbf{C}^\top$ used in registration. We can use standard hybrid argument of replacing $\mathbf{C}^\top$ with random values in the encryption, which implies the server's choice of δ is independent of $\mathbf{C}$. Applying Lemma 7 completes the proof. $\square$

Theorem 11 (Security of vReinsPIRe). *Let λ be the security parameter and choose κ such that $2^{-\kappa} = \text{negl}(\lambda)$. Let $\mathcal{D}_h = \{\mathbf{D} \mid \mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}, \|\mathbf{D}\|_\infty \leq p\}$, and $\mathcal{D}_a = \{\mathbf{D} \mid \mathbf{D} \in \mathbb{Z}^{n_1 \times n_0}, \|\mathbf{D}\|_\infty \leq 2n_1p\}$. Suppose that:*

- The KDRLWE assumption of Theorem 10 holds
- $(d, n_0, q, (2n_1 + 1)n_1p)$ -SIS is hard
- The LHE used during registration is query-hiding against PPT servers and correct for honest evaluation
- ReinsPIRe is correct for every database in $\mathcal{D}_a$

Then vReinsPIRe securely computes $\mathcal{F}_{\text{LHE}}$ with abort against a malicious server according to Definition 3.

Proof. Let $\mathcal{A}$ be a non-uniform PPT malicious server. We construct a stateful simulator $\mathcal{S}$ as follows:

- On message (id, Register) from $\mathcal{F}_{\text{LHE}}$:
 - $\mathcal{S}$ receives $\mathbf{H}$ from $\mathcal{A}$ and invokes the extractor by rewinding $\mathcal{A}$.
 - If extraction fails or if the registration protocol does not produce an accepting transcript, it sends Abort to $\mathcal{F}_{\text{LHE}}$ and marks the client as aborted.
 - Otherwise, the extractor outputs $\tilde{\mathbf{D}}_{\text{id}} \in \mathcal{D}_a$ that satisfies $\mathbf{H} = \tilde{\mathbf{D}}_{\text{id}} \mathbf{A}$.
 - $\mathcal{S}$ sends $\tilde{\mathbf{D}}_{\text{id}}$ to $\mathcal{F}_{\text{LHE}}$, and stores the accepted client state $([a_i], [t''_{i,j}], \mathbf{C}, \mathbf{Z}, \mathbf{Z}')$ under id.
- On message (id, Query, $\perp$) from $\mathcal{F}_{\text{LHE}}$:
 - $\mathcal{S}$ first checks whether the corresponding registration aborted, and if so, sends Abort to $\mathcal{F}_{\text{LHE}}$.
 - Otherwise, it generates a valid ReinsPIRe query for the zero vector using a fresh secret key: $(\text{st}, \text{qry}) \leftarrow \text{ReinsPIRe.Query}(\text{pp}, \mathbf{0})$.
 - It then sends the $\mathbf{u}$ component of qry to $\mathcal{A}$, receives a response $\mathbf{x}$, and performs the same verification as an honest client using the stored $(\mathbf{C}, \mathbf{Z}, \mathbf{Z}')$, sending Abort if the verification fails and Allow otherwise.

We prove indistinguishability using the following sequence of hybrids:

- *Hybrid 0*: This is the ideal world execution with $\mathcal{S}$ described above.
- *Hybrid 1*: The ideal functionality reveals the real query vector $\mathbf{p}$ to the simulator, and the simulator generates a valid ReinsPIRe query for $\mathbf{p}$.
- *Hybrid 2*: Instead of directly using $\mathcal{F}_{\text{LHE}}$'s answer, the client runs the verification procedure of Algorithm 4 and uses its resulting output.
- *Hybrid 3*: This is the real world execution.

Hybrid 0 $\sim$ Hybrid 1: The KDRLWE assumption implies that the distribution of $\mathbf{u}$ of a ReinsPIRe query for $\mathbf{p}$ is indistinguishable from the zero vector. For a polynomial number of queries, we can use a standard hybrid argument replacing one fresh key at a time.

Hybrid 1 $\sim$ Hybrid 2: In this case, the server's view is unchanged, and only the honest client's output is constructed differently. If verification fails, both hybrids output Abort, so suppose verification accepts.

First, $\mathbf{H}'$, $[a_i]$, and $[t''_{i,j}]$ were all generated locally from $\mathbf{H}$ and the CRS. Since $\mathbf{H} = \tilde{\mathbf{D}}_{\text{id}} \mathbf{A}$, these are exactly the preprocessing values obtained by applying ReinsPIRe.GenPack to the hint matrix of $\tilde{\mathbf{D}}_{\text{id}}$.

By Lemma 8, the client receives $\mathbf{x} = [\tilde{\mathbf{D}}_{\text{id}}, \mathbf{H}'] \mathbf{u}$. The client then locally computes $\mathbf{z} = \text{vec}(\sum_{j=0}^{\ell-1} t''_{i,j}(X) y'_j(X))$ and decrypts $\mathbf{x} + \mathbf{z}$. Therefore, the ciphertext reconstructed by the client is exactly the output of an honest ReinsPIRe.Eval on the database $\tilde{\mathbf{D}}_{\text{id}}$ and query $\mathbf{p}$. Because $\tilde{\mathbf{D}}_{\text{id}} \in \mathcal{D}_a$, the ReinsPIRe correctness guarantee gives $\tilde{\mathbf{D}}_{\text{id}} \mathbf{p} \bmod p$ except with negligible probability, which is the same result from $\mathcal{F}_{\text{LHE}}$.

Hybrid 2 $\sim$ Hybrid 3: Hybrid 2 is identical to the real protocol.

□

C Improving Compile Runtime

In this section, we show an $O(d^2 \log d)$ algorithm for Compile that was presented in Section 3.3. Lemma 2 established that there exists a matrix $\mathbf{M}$ such that $\mathbf{M} \cdot \text{vec}(y(X)) \cong \sum_{i=0}^{k-1} t_i(X) \tau_i(y(X))$.

Recall that the automorphism group consists of $\{X \mapsto X^{2^{j+1}}\}_{j \in [d]}$. With some relabelings of $t_i(X)$ and adding dummy zero t_i , we can re-express this identity as follows:

$$\mathbf{M} \cdot \text{vec}(y(X)) \cong \sum_{i=0}^{d-1} \hat{t}_i(X) \cdot y(X^{2^{i+1}})$$

where $\hat{t}_i(X)$ corresponds to either $t_i(X)$ or zero.

Next, we notice that the j -th column of $\mathbf{M}$ corresponds to when $y(X) = X^j$. That is,

$$\begin{aligned}\mathbf{M}[:,j] &\cong \sum_{i=0}^{d-1} \hat{t}_i(X) \cdot X^{j \cdot 2i+1} \\ &= \sum_{i=0}^{d-1} \hat{t}_i(X) \cdot X^{2ij+j} \\ &= X^j \cdot \sum_{i=0}^{d-1} \hat{t}_i(X) \cdot (X^{2j})^i\end{aligned}$$

Let $P(Z) = \sum_{i=0}^{d-1} \hat{t}_i(X) \cdot Z^i \in R_q[Z]$ be a polynomial over R_q with variable Z . Then, we can view computing $\mathbf{M}[:,j]$ as evaluating $X^j P(X^{2j})$.

In the ring R_q , $\omega = X^2$ is a d -th primitive root of unity. In particular, $P(\omega^0), \dots, P(\omega^{d-1})$ can be evaluated efficiently using FFT with $O(d \log d)$ ring operations. We note that each ring operation takes $O(d)$ time, since in Cooley-Tukey butterfly the only multiplications involved are multiplications by powers of ω , which translates to cyclic rotations of the polynomial coefficients (in particular, we do not require q to be NTT-friendly). Thus, we can evaluate the entire d evaluation points in $O(d^2 \log d)$ time. The only remaining steps after these evaluations are multiplying X^j to the j -th evaluation, which can easily be done in $O(d^2)$ total. We summarize this discussion as the following lemma.

Lemma 9 (Improved Compile). *There exists an algorithm that computes Compile in $O(d^2 \log d)$ time.*

D InspiRING Modifications and Improvements

D.1 Addressing d^{-1} Existence with Power-of-two q

We briefly recall the transformation in InspiRING that converts LWE embedded RLWE ciphertext into an intermediate ciphertext to clear the "garbage" coefficients of the plaintext polynomial. InspiRING begins with the RLWE interpretation of the LWE ciphertext $(\mathbf{a}, b = -\langle \mathbf{a}, \mathbf{s} \rangle + e + \Delta m)$:

$$\begin{aligned}\tilde{a}(X) &= \sum_i \mathbf{a}[i] \cdot X^{-i} \\ \tilde{b}(X) &= -\tilde{a}(X)\tilde{s}(X) + \tilde{e}(X) + \Delta \cdot \tilde{m}(X) = b\end{aligned}$$

where $\tilde{s}(X) = \sum_i \mathbf{s}[i] \cdot X^i$. With this interpretation, $\tilde{m}(X)$ (resp. $\tilde{e}(X)$) contains m (resp. e) in its constant term, while the other coefficients contain "garbage" values. For packing, these garbage coefficients must be cleared out. InspiRING applies the trace operator (see Lemma 1 in [58]) on $\tilde{b}(X)$ lifted to R :

$$\begin{aligned}d \cdot \tilde{b}(X) &= \left(\sum_{i=0}^{d/2-1} -\tilde{a}(X^{5^i})\tilde{s}(X^{5^i}) - \tilde{a}(X^{-5^i})\tilde{s}(X^{-5^i}) \right) \\ &\quad d \cdot e + d\Delta \cdot m + dq \cdot \tilde{x}(X)\end{aligned}\tag{3}$$

where $q \cdot \tilde{x}(X)$ is the extra term introduced from lifting to R . The message space now only consists of the constant term m . InspiRING then converts this back to R_q and divides both side by d . Finally, it interprets $\tilde{b}(X)$ as the message encoding component of the intermediate ciphertext in $R_q^d \times R_q$, with random components $\{d^{-1}\tilde{a}(X^{\pm 5^i})\}_{i \in [d]}$ and secret key components $\{\tilde{s}(X^{\pm 5^i})\}_{i \in [d]}$.

In particular, InspiRING implicitly required the ciphertext modulus q to be odd for d^{-1} to exist. However, this creates an issue for a power-of-two modulus q used in this work.

To get around this, we make the following modification: from Equation 3, we interpret $(\{a(X^{\pm 5^i})\}_{i \in [d]}, d \cdot \tilde{b}(X))$ as a ciphertext over $R_{dq}^d \times R_{dq}$. Note that this is a valid ciphertext that encrypts m (with scaling factor $d\Delta$). The d such ciphertexts are then aggregated in the $R_{dq}^d \times R_{dq}$ representations, where each coefficient of the plaintext polynomial contains the LWE message.

We then perform modulus switching to convert the ciphertext down to $R_q^d \times R_q$. We can show that the result is a valid intermediate ciphertext encrypting the aggregated message, with a small additional noise from modulus switching. It turns out that this additional noise is practically negligible in the entire packing process.

Lemma 10. Suppose the secret key is uniform ternary. The transformation above incurs an additive noise $e'(X)$ where $\|e'(X)\|_\infty \leq d^2$.

Proof. From Equation 3, divide both sides by d to yield:

$$\begin{aligned}\tilde{b}(X) &= \left(\sum_{i=0}^{d/2-1} -\frac{\tilde{a}(X^{5^i})}{d} \tilde{s}(X^{5^i}) - \frac{\tilde{a}(X^{-5^i})}{d} \tilde{s}(X^{-5^i}) \right) \\ &\quad + e + \Delta m(X) + q \cdot \tilde{x}(X) \\ &= \left(\sum_{i=0}^{d/2-1} -\lfloor \frac{\tilde{a}(X^{5^i})}{d} \rfloor \tilde{s}(X^{5^i}) - \lfloor \frac{\tilde{a}(X^{-5^i})}{d} \rfloor \tilde{s}(X^{-5^i}) \right) \\ &\quad + \left(\sum_{i=0}^{d/2-1} \epsilon_i(X) \tilde{s}(X^{5^i}) + \epsilon'_i(X) \tilde{s}(X^{-5^i}) \right) \\ &\quad + e + \Delta m(X) + q \cdot \tilde{x}(X)\end{aligned}$$

where $\|\epsilon_i(X)\|_\infty, \|\epsilon'_i(X)\|_\infty < 1$. We will treat $\sum_{i=0}^{d/2-1} \epsilon_i(X) \tilde{s}(X^{5^i}) + \epsilon'_i(X) \tilde{s}(X^{-5^i})$ term as the additional noise incurred from the transformation. Since $\tilde{s}$ is ternary and automorphism preserves norm, $\|\epsilon_i(X) \cdot \tilde{s}(X^{5^i})\|_\infty, \|\epsilon'_i(X) \cdot \tilde{s}(X^{-5^i})\|_\infty \leq d$. Summing d such terms completes the proof. $\square$

Remark. Usually this worst case noise bound analysis is too conservative in practice, and using the independence heuristic yields better (and more realistic) bound.

D.2 Optimizing InspiRING.Preprocess

We observe that the algebra behind the fast transpose computation in [69] can be applied to improve the offline preprocessing stage of InspiRING. Most of the steps below are derived from [69] and written in full for completeness.

Given LWE random components $\{\mathbf{a}_i\}_{i \in [d]}$, InspiRING.Preprocess interprets these as $\tilde{a}_i(X) = \sum_j \mathbf{a}_i[j] \cdot X^{-j}$, and computes

$$\hat{a}_i(X) = \sum_j \tilde{a}_j(X^{k_i}) \cdot X^j$$

where k_i is of the form $\pm 5^i$ corresponding to the automorphism. Computing this in a straightforward way takes $O(d^2)$ for each $\hat{a}_i(X)$, and hence in total $O(d^3)$.

Just as in Appendix C, we can employ FFT to improve this step. Let $\tilde{a}_j(X) = \sum_{k=0}^{d-1} c_{j,k} \cdot X^k$. By relabeling $\hat{a}_i(X)$ to $\hat{a}'_i(X)$, we have:

$$\begin{aligned}\hat{a}'_i(X) &= \sum_j \tilde{a}_j(X^{2i+1}) \cdot X^j \\ &= \sum_j \sum_k c_{j,k} \cdot X^{(2i+1)k} \cdot X^j \\ &= \sum_k \sum_j c_{j,k} \cdot X^{(2i+1)k} \cdot X^j \\ &= \sum_k X^{2ik} \sum_j c_{j,k} \cdot X^{j+k} \\ &= \sum_k \tilde{a}'_k(X) X^{2ik}\end{aligned}$$

where in the last equality we have set $\tilde{a}'_k(X) = \sum_j c_{j,k} \cdot X^{j+k}$. After this step, the rest are almost identical to Appendix C. We express computing $\hat{a}'_i(X)$ as a polynomial evaluation over $P(Z) = \sum_k \tilde{a}'_k(X) \cdot Z^k$ at X^{2i} , and these d evaluations can be done efficiently using ring based FFT in $O(d^2 \log d)$ times.

After this initial aggregation step, the remaining steps in InspiRING.Preprocess takes $O(\ell d^2 \log d)$ time. We summarize this result as the following lemma.

Lemma 11 (Improved InspiRING.Preprocess). *There exists an algorithm that computes InspiRING.Preprocess in $O(\ell d^2 \log d)$ time.*

D.3 Communication Efficient PIR from ReinsPIRe

Algorithm 5 ReinsPIRe⁺

Input: $\mathbf{A} \in \mathbb{Z}_q^{n_0 \times d}$: Random components of the LWE ciphertexts.

Input: $[w_j(X)]_{j \in [\ell]}$: Random component of the KS key for switching $s(X^5) \mapsto s(X)$

Input: $[w'_j(X)]_{j \in [\ell]}$: Random component of the KS key for switching $s(X^{-1}) \mapsto s(X)$

Input: $\mathbf{D} \in \mathbb{Z}^{td \times n_0}$: Plaintext database.

```

procedure Preprocess( $\mathbf{A}, [w_j(X)]_{j \in [\ell]}, [w'_j(X)]_{j \in [\ell]}, \mathbf{D}$ )
   $\mathbf{D}' \leftarrow \text{EncodeDB}(\mathbf{D})$ 
   $\text{sp} \leftarrow \text{ReinsPIRe.Preprocess}(\mathbf{A}, [w_j(X)], [w'_j(X)], \mathbf{D}')$ 
  return ( $\mathbf{D}', \text{sp}$ )

```

```

procedure EncodeDB( $\mathbf{D}$ )
   $\mathbf{D}' \leftarrow \text{Perform Cooley-Tukey on each column of } \mathbf{D}, \text{ interpreted as vector of } R_p.$ 
  return  $\mathbf{D}'$ 

```

Input: $\mathbf{A} \in \mathbb{Z}_q^{n_0 \times d}$: Random components of the LWE ciphertexts.

Input: $[w_j(X)]_{j \in [\ell]}$: Random component of the KS key for switching $s(X^5) \mapsto s(X)$

Input: $[w'_j(X)]_{j \in [\ell]}$: Random component of the KS key for switching $s(X^{-1}) \mapsto s(X)$

Input: idx : Index of the entry to retrieve.

```

procedure Query( $\mathbf{A}, [w_j(X)]_{j \in [\ell]}, [w'_j(X)]_{j \in [\ell]}, \text{idx}$ )
  Parse  $\text{idx}$  as  $(r, c)$ 
   $\mathbf{p} \leftarrow \text{One hot vector with } c\text{-th bit on.}$ 
   $\text{qry}' \leftarrow \text{ReinsPIRe.Query}(\mathbf{A}, [w_j(X)], [w'_j(X)], \mathbf{p})$ 
   $\text{qry}'' \leftarrow \text{RGSW.Enc}(X^{r \cdot 2d/t})$ 
  return  $\text{qry} = (\text{qry}', \text{qry}'')$ 

```

Input: ($\mathbf{D}', \text{sp}$) Output from Preprocess

Input: qry : Query to retrieve idx -th entry.

```

procedure Eval( $(\mathbf{D}', \text{sp}), \text{qry}$ )
  Parse  $\text{qry}$  as  $(\text{qry}', \text{qry}'')$ 
   $\text{resp} \leftarrow \text{ReinsPIRe.Eval}(\text{sp}, \mathbf{D}', \text{qry}')$ 
  Parse  $\text{resp}$  as  $t$  RLWE ciphertexts  $\text{RLWE}(c_0), \dots, \text{RLWE}(c_{t-1})$ 
  return  $\text{EvalPoly}([ \text{RLWE}(c_i) ]_{i \in [t]}, \text{qry}'')$ 

```

```

procedure EvalPoly( $[ \text{RLWE}(c_i) ]_{i \in [t]}, \text{qry}''$ )
  Parse  $\text{qry}''$  as  $\text{RGSW}(X^{r \cdot 2d/t})$ 
   $\text{ct} \leftarrow \text{RLWE}(c_{t-1})$ 
  for  $i = t - 2$  to  $0$  do
     $\text{ct} \leftarrow \text{ct} \cdot \text{RGSW}(X^{r \cdot 2d/t}) + \text{RLWE}(c_i)$ 
  return  $\text{ct}$ 

```

Input: resp : Response RLWE ciphertext encrypting idx -th entry

Input: $s(X) \in R_q$: Secret key

```

procedure Dec( $\text{resp}, s(X)$ )
  Decrypt  $\text{resp}$  using  $s(X)$ 

```

We present the communication efficient PIR instantiation of ReinsPIRe using the additional "folding" step with polynomial evaluation in Algorithm 5. This was originally proposed in InsPIRe to reduce down the response further, and we show that our ReinsPIRe LHE can be used as a blackbox to realize this instantiation.

While we do not provide formal security guarantees here, we note that the security proof in [58] works almost verbatim.

E Additional Experiments

E.1 Additional Optimizations

On top of our algorithmic advancements, we also include additional optimizations in our implementation of ReinsPIRe and vReinsPIRe, which are standard techniques in the FHE literature. We use approximate gadget decomposition for keyswitching which reduces the size of keyswitching keys and preprocessing material, at the cost of a minor increase in homomorphic noise. Through precise choice of parameters, we can ensure that the noise term introduced by key switching remains the non-dominant term. We also perform signed gadget decomposition, which reduces the noise growth during keyswitching. Lastly, we choose optimized parameters for the cryptosystem, which is chosen dynamically based on the database size.

E.2 Breakdown of ReinspiRING Runtime

Table 5 shows a breakdown of the runtime of ReinspiRING including the preprocessing and the different parts of the online packing.

Table 5: Breakdown of runtime of Packing for ReinspiRING

Parameters	ReinspiRING.Preprocess	ReinspiRING.Pack	
		$\mathbf{H}' \cdot \mathbf{y}$	$\sum_{j=0}^{\ell-1} t_j''(X) \cdot y_j'(X)$
$\ell = 2, z = 2^{19}$	3.9 s	13.6 ms	1.2 ms
$\ell = 3, z = 2^{19}$	6.0 s	20.6 ms	1.8 ms

E.2.1 Breakdown of ReinsPIRe Performance

Table 6 gives the breakdown of the runtime and communication costs of ReinsPIRe for a 1GB database. The runtime of the first multiplication dominates the total runtime.

E.3 Improvements on InsPIRe

In Table 7, we compare InsPIRe and ReinsPIRe under identical parameters set to isolate the advantages of our new protocol. We observe an improvement in server runtime and server storage, with no increase in communication costs, which supports the theoretical analysis.

F More Comparisons with Related Works

F.1 Comparisons with Bae et al. [11]

Recent work [11] gives generic reductions from encrypted matrix-vector products to plain linear algebra and shares a similar theme as our work. However, its RGSW-based PC-Mv reduction requires a matrix component encoding the secret key as part of the message (in more detail, $\mathbf{M}\mathbf{S}^v$ where $\mathbf{M}$ is the plaintext matrix and $\mathbf{S}^v$ is the secret key matrix of the encrypted vector), which must be derived from an RLWE representation using the ciphertext matrix transposition algorithm [69] and key-switchings. As the authors note, "the cost of this conversion is too large for it to be used in the context of a ciphertext to cleartext reduction for Mv." While this initial cost can be amortized when the secret key is reused, this is no longer feasible in the CRS setting where a fresh key must be generated every query.

On the other hand, ReinsPIRe's server side compilation for the cleartext Mv reduction does not rely on client secret keys. The compilation into $\mathbf{H}'$ only involves database and CRS dependent terms. The secret key dependence remains only in the message encoding component of the LWE query vector and the key-switching key vector, which are sent by the client every query.

Table 6: Breakdown of ReinsPIRe runtime and communication costs for PIR over a 1 GB database.

Metric	$n_0 = 2^{16}$	$n_0 = 2^{15}$	$n_0 = 2^{14}$
Server Global Preprocessing Time (s)	141	152	170
Server Preprocessed Size Ideal (MB)	120	240	480
Client Generate Request (ms)	111	69	47
Server Response Time (ms)	139	173	235
- DB Mult Time (ms)	111	113	112
- Packing Time (ms)	16	31	61
- Poly Eval Time (ms)	12	29	62
Client Extract Response (ms)	1.9	1.9	1.9
Client Upload (KB)	546	338	234
- First Dim Vector (KB)	416	208	104
- RGSW Second Dim (KB)	78	78	78
- Keys (KB)	52	52	52
Client Download (KB)	12	12	12

Table 7: Performance metrics for InspiRe and ReinsPIRe over a 1 GB database under identical parameters ($d = 2048, \log q = 56, k = 3, z = 2^{19}$) which isolates the advantages of our new protocol

Metric	InspiRe [58]	ReinsPIRe
Offline Phase		
Preprocessing	1.7m	1.5m
Storage	895 MB	584 MB
Online Phase		
Client generate request	1.0s	98.5 ms
Server Response Time	304 ms	164 ms
Client extract response	0.1 ms	4.0 ms
Client Upload	392 KB	392 KB
Client Download	12 KB	12 KB

G ReinspiRING with a Single Key-switching Key

In this section, we present a variant of InspiRING (and consequently, ReinspiRING) that only uses a single key-switching key instead of two. Our key observation is that the reason that InspiRING required two key-switching keys is because the underlying automorphism group is generated by two group elements (e.g. $X \mapsto X^5$ and $X \mapsto X^{-1}$). In particular, we could bypass this limitation by choosing a suitable cyclotomic ring with a cyclic automorphism group, e.g. a prime cyclotomic ring.

For the rest of the section, let d be an odd prime and let $\phi_d(X) = 1 + X + \dots + X^{d-1}$. Let $R = \mathbb{Z}[X]/(\phi_d(X))$ and $R_q = \mathbb{Z}_q[X]/(\phi_d(X))$.

G.1 InspiRING with a Single Key-switching Key

Our construction relies on the following algebraic facts:

Lemma 12 (Ring Embedding of LWE Ciphertext). *Let $(\mathbf{a}, b) \in \mathbb{Z}_q^{d-1} \times \mathbb{Z}_q$ be an LWE ciphertext encrypted using a secret key $\mathbf{s} \in \mathbb{Z}^{d-1}$, satisfying $b = -\langle \mathbf{a}, \mathbf{s} \rangle + e + \Delta m$. Define*

$$\tilde{a}(X) := \mathbf{a}[0] - \sum_{j=1}^{d-2} \left(\sum_{i=1}^{d-1-j} \mathbf{a}[i] \right) X^j$$

Then $(\tilde{a}(X), b)$ is an RLWE ciphertext encrypted with $\tilde{s}(X) = \sum_{i=0}^{d-2} \mathbf{s}[i] \cdot X^i$ satisfying:

$$b = -\tilde{a}(X)\tilde{s}(X) + \tilde{e}(X) + \Delta\tilde{m}(X) \pmod{q}$$

where $\tilde{e}_0 = e$ and $\tilde{m}_0 = m$ (the constant terms of $\tilde{e}(X)$ and $\tilde{m}(X)$).

Proof. Direct verification. □

Lemma 13 (Trace). *Let $a(X) = a_0 + \dots + a_{d-2} \cdot X^{d-2} \in R$. Then*

$$\text{Tr}((1-X) \cdot a(X)) = d \cdot a_0$$

Proof. By linearity of Tr , we have $\text{Tr}((1-X) \cdot a(X)) = \sum_{k=0}^{d-2} a_k (\text{Tr}(X^k) - \text{Tr}(X^{k+1}))$, and since $\text{Tr}(1) = d-1$ and $\text{Tr}(X^k) = -1$ for $0 < k < d$, the result follows. □

Lemma 14 (Cyclic Automorphism Group). *For a prime cyclotomic ring R , the automorphism group is cyclic with order $d-1$.*

We can apply the lemmas above directly to the first stage of InspiRING construction, as follows:

Lemma 15 (Intermediate Ciphertext Transformation). *Let τ be a generator of the automorphism group of R . Let $(\mathbf{a}, b) \in \mathbb{Z}_q^{d-1} \times \mathbb{Z}_q$ be an LWE ciphertext encrypted using a secret key $\mathbf{s} \in \mathbb{Z}^{d-1}$, and let $(\tilde{a}(X), b)$ be the ring embedding of this LWE ciphertext as in Lemma 12. Define $\hat{\mathbf{a}} := [d^{-1} \cdot \tau^i((1-X)\tilde{a}(X))]_{i \in [d-1]}$ and $\hat{\mathbf{s}} := [\tau^i(\tilde{s}(X))]_{i \in [d-1]}$.*

Then $(\hat{\mathbf{a}}, b) \in R_q^{d-1} \times R_q$ is a valid intermediate ciphertext satisfying:

$$b = -\langle \hat{\mathbf{a}}, \hat{\mathbf{s}} \rangle + e + \Delta m \pmod{q}$$

Proof. The ring embedding in Lemma 12 satisfies

$$b = -\tilde{a}(X)\tilde{s}(X) + \tilde{e}(X) + \Delta\tilde{m}(X) \pmod{q}$$

where $\tilde{m}(X)$ and $\tilde{e}(X)$ contains the original LWE message and noise in the constant term. Lifting to R and multiplying both sides by $(1-X)$ yields:

$$\begin{aligned} (1-X)b &= -(1-X)\tilde{a}(X)\tilde{s}(X) \\ &\quad + (1-X)\tilde{e}(X) + \Delta(1-X)\tilde{m}(X) + q \cdot (1-X)\tilde{x}(X) \end{aligned}$$

Applying Tr on both sides and invoking Lemma 13, we get:

$$\begin{aligned} db &= -\text{Tr}((1-X)\tilde{a}\tilde{s}) + \text{Tr}((1-X)\tilde{e}) \\ &\quad + \Delta\text{Tr}((1-X)\tilde{m}) + q\text{Tr}((1-X)\tilde{x}) \\ &= -\sum_{i=0}^{d-2} \tau^i((1-X)\tilde{a}) \cdot \tau^i(\tilde{s}) + de + d\Delta m + dq \cdot x \end{aligned}$$

Modding down to R_q and multiplying both sides by d^{-1} (which exists if q and d are relatively prime) completes the proof. □

The rest of the ring packing follows identically to InspiRING, except that we only require one key-switching key corresponding to the generator τ .

Remark. Unlike in the ReinspiRING protocol, this does not involve approximate division by d since for a power of two modulus q , $\gcd(d, q) = 1$.

G.2 ReinspiRING with a Single Key-switching Key

We provide an analogous matrix formulation (Lemma 1) and compilation lemma (Lemma 2) for prime cyclotomic rings.

Lemma 16 (Matrix Formulation of Polynomial Mult). *Let $a(X) = a_0 + \dots + a_{d-2} \cdot X^{d-2} \in R_q$ and $b(X) \in R_q$, and define $\hat{\mathbf{a}} = [a_0, a_1, \dots, a_{d-2}, 0]^\top \in \mathbb{Z}_q^d$. Define $\text{Circ}(\hat{\mathbf{a}})$ as the $d \times d$ circulant matrix representation of $\hat{\mathbf{a}}$. Then*

$$a(X) \cdot b(X) \cong [\mathbf{I}_{d-1}, -\mathbf{1}] \cdot \text{Circ}(\hat{\mathbf{a}}) \cdot \begin{bmatrix} \mathbf{I}_{d-1} \\ \mathbf{0} \end{bmatrix} \cdot \text{vec}(b(X))$$

Lemma 17 (Matrix Formulation of Automorphism). *Let $a(X) \in R_q$ and τ be an automorphism. Then there exists a $d \times d$ permutation matrix $\mathbf{P}_\tau$ such that the following holds:*

$$\tau(a(X)) \cong [\mathbf{I}_{d-1}, -\mathbf{1}] \cdot \mathbf{P}_\tau \cdot \begin{bmatrix} \mathbf{I}_{d-1} \\ \mathbf{0} \end{bmatrix} \cdot \text{vec}(a(X))$$

Lemma 18 (Compilation Lemma for Prime Cyclotomic Rings). *Let $R_q = \mathbb{Z}_q[X]/(\phi_d(X))$ for an odd prime d . Let $[t_i(X)]_{i \in [k]}$ be a sequence of k polynomials $\in R_q$ such that $\|t_i(X)\|_\infty \leq B$ for some B . Let $y(X) \in R_q$ and let $[\tau_i]_{i \in [k]}$ be a sequence of k automorphisms. Then there exists a matrix $\mathbf{M} \in \mathbb{Z}_q^{d-1 \times d-1}$ that is only a function of t_i and τ_i , such that $\|\mathbf{M}\|_\infty \leq 2kB$ and the following holds:*

$$\sum_{i=0}^{k-1} t_i(X) \cdot \tau_i(y(X)) \cong \mathbf{M} \cdot \text{vec}(y(X))$$

where vec is the vector representation of the coefficients of $y(X)$. Furthermore, $\mathbf{M}$ can be computed in $O(kd^2)$ time.

Proof. The proof proceeds almost identically to Lemma 2 by invoking Lemma 16 and Lemma 17. It remains to show that the matrix norm is bounded by $2kB$. Let $\hat{\mathbf{t}}_i = [t_{i,0}, \dots, t_{i,d-2}, 0]^\top$ be the coefficients of $t_i(X)$ with 0 appended as in Lemma 16. It suffices to show that

$$\| [\mathbf{I}_{d-1}, -\mathbf{1}] \cdot \text{Circ}(\hat{\mathbf{t}}_i) \cdot \begin{bmatrix} \mathbf{I}_{d-1} \\ \mathbf{0} \end{bmatrix} \cdot [\mathbf{I}_{d-1}, -\mathbf{1}] \cdot \mathbf{P}_{\tau_i} \cdot \begin{bmatrix} \mathbf{I}_{d-1} \\ \mathbf{0} \end{bmatrix} \|_\infty \leq 2B$$

which can be verified directly. □

Remark. We can obtain an analogous version of Corollary 1 for prime cyclotomic rings, but unlike in the power-of-two cyclotomics, we don't need to handle the residue term $t''(X) \cdot y'(X)$. This makes the matrix formulation of the protocol particularly clean, since the entire packing boils down to a single matrix-vector multiplication.

We leave offline preprocessing optimizations, formal noise/security analysis, and concrete implementations of this variant for future work.