InsPIRe: Communication-Efficient PIR with Server-side Preprocessing

Rasoul Akhavan Mahdavi
Google and University of Waterloo
rasoul.akhavan.mahdavi@uwaterloo.ca

Sarvar Patel
Google
sarvar@google.com

Joon Young Seo
Google
jyseo@google.com

Kevin Yeo
Google and Columbia University
kwlyeo@google.com

Abstract

We present InsPIRe that is the first private information retrieval (PIR) construction simultaneously obtaining both high-throughput and low query communication while using only server-side preprocessing (meaning no offline communication). Prior PIR schemes with both high-throughput and low query communication required substantial offline communication of either downloading a database hint that is 10-100x larger than the communication cost of a single query (such as SimplePIR and DoublePIR [Henzinger *et al.*, USENIX Security 2023]) or streaming the entire database (such as Piano [Zhou *et al.*, S&P 2024]). In contrast, recent works such as YPIR [Menon and Wu, USENIX Security 2024] avoid offline communication at the cost of increasing the query size by 1.8-2x, up to 1-2 MB per query. Our new PIR protocol, InsPIRe, obtains the best of both worlds by obtaining high-throughput and low communication without requiring any offline communication. Compared to YPIR, InsPIRe requires 5x smaller cryptographic keys, requires up to 50% less online query communication while obtaining up to 25% higher throughput. We show that InsPIRe enables improvements across a wide range of applications and database shapes including the InterPlanetary File System and private device enrollment.

At the core of InsPIRe, we develop a novel ring packing algorithm, InspiRING, for transforming LWE ciphertexts into RLWE ciphertexts. InspiRING is more amenable to the server-side preprocessing setting that allows moving the majority of the necessary operations to offline preprocessing. InspiRING only requires two key-switching matrices whereas prior approaches needed logarithmic key-switching matrices. We also show that InspiRING has smaller noise growth and faster packing times than prior works in the setting when the total key-switching material sizes must be small. To further reduce communication costs in the PIR protocol, InsPIRe performs the second level of PIR using homomorphic polynomial evaluation, which only requires one additional ciphertext from the client.

Contents

1. Introduction

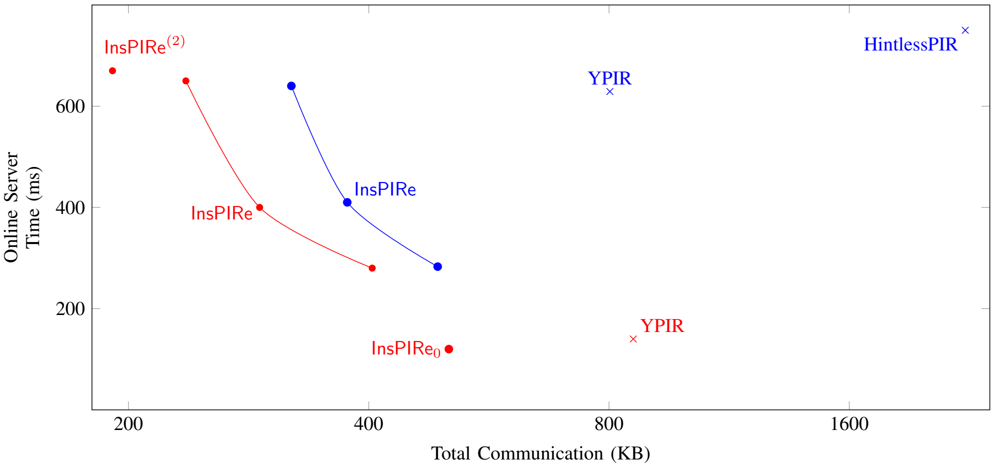

Figure 1: Communication Cost vs. Online Server Runtime for a PIR query over a 1 GB database. Red points retrieve 1-bit entry and blue points retrieve a 32KB entry.

Private information retrieval (PIR) is a powerful cryptographic protocol enabling users to privately query entries from a public database held by a server. In this protocol, the server holds a database with N entries and the client wishes to retrieve the i-th entry. PIR ensures the client's query index i remains hidden from the server. The strong privacy guarantees of PIR have been critically leveraged to design privacy-preserving systems for many applications such as advertising [39], blocklists [47], certificate transparency [43], contact discovery [9], [33], [46], databases [78], file systems [60], media consumption [40], metadata-hiding communication [66], [5], [49], [4], [1], password leak checks [77], [54], [3] and web search [42]. PIR has also seen adoption for real-world deployments in industry by Apple [6], Google [80] and Microsoft [51].

PIR has been studied in both the single-server (such as [48], [71], [70]) and multiple, non-colluding server settings (including [24], [35], [11]). While multi-server PIR schemes are efficient, their privacy relies on stronger, non-collusion trust assumptions that may be difficult to materialize in practice. In our work, we focus exclusively on single-server PIR to avoid the these challenges.

When evaluating the efficiency of PIR protocols, there are two core cost metrics that must be considered. First, the amount of computational cost necessary to serve PIR requests. The computational cost can be measured using throughput, which is the ratio of the database size and the server time (equivalently, the amount of time needed per database entry). The second important metric is the communication (bandwidth) needed for each PIR query. Over the past decade, many works have studied the problem of designing efficient PIR protocols that we survey below.

Server-Stored Client-Specific Keys. Starting from XPIR [61], there has been a long line of work designing PIR protocols based on the RLWE hardness assumption including [4], [3], [67], [1], [62], [16]. These works obtain small online communication, but have two main drawbacks. First, they require substantially larger computational costs during queries compared to other PIR schemes (see below). Even worse, these works requires the server to store client-specific keys of several megabytes in size that must be used during PIR queries, which is very limiting in practice. These client-specific keys must be uploaded to the server (for example, either with the first query or in an offline preprocessing step). So, the total communication cost is substantially large, especially if users only perform a small number of PIR queries. Also, server storage grows with the number of users limiting applicability for systems with very large user bases.

Client-Stored Database Hints. Another line of work has considered the case where the PIR protocol with offline preprocessing where the client must retrieve and store hints about the database. Several recent works including SimplePIR and DoublePIR [43] as well as FrodoPIR [30] build LWE-based PIR schemes in this class. They obtain higher throughput compared to the prior RLWE-based PIR schemes while maintaining slightly larger, but comparable query communication. However, the database hint, that must be downloaded by each user before issuing any PIR queries, is substantially large. Furthermore, hints must be re-downloaded by all users each time the database changes.

Taking this paradigm to an even more extreme, recent work [27] has shown that client-stored database hints can obtain PIR schemes with online query time sublinear in the database size obtaining even higher throughput. Unfortunately, practical instantiations including [81], [76] require every user to stream the entire database during offline preprocessing. Furthermore, the database must be streamed every $\tilde{O}(\sqrt{n})$ PIR queries unlike the prior LWE-based schemes [43], [30]. The very recent work of ThorPIR [37] avoid streaming the entire database by, instead, using heavy cryptographic primitives requiring up to multiple hours of server computation for just one client to retrieve a hint.

Server-side Preprocessing with No Offline Communication. The main drawback of the prior two classes of PIR schemes usually come from the communication required in the offline preprocessing steps. To solve this, recent works including Tiptoe [42], HintlessPIR [53], YPIR [63], RLWEPIR [60], WhisPIR [32], and KSPIR [58] construct PIR protocols in the Common Reference String (CRS) model where there is no offline communication. This solves many of the problems from the prior schemes that require offline communication. Unfortunately, these works end up sacrificing online communication during PIR queries to eliminate offline communication. For example, YPIR [63] requires up to megabytes of online communication more compared to PIR with client-stored hints.

In a recent breakthrough work, Lin *et al.* [56] showed the first PIR scheme with silent preprocessing and sublinear query time. Unfortunately, the concrete cost remains large to date rendering the construction impractical [69].

Our Question. Given the current state of affairs in PIR, we are left to navigate one of two obstacles. If we insist on both high-throughput and low query communication, then we must solve the challenges and costs associated with the offline communication for either server-stored client-specific keys or client-downloaded database hints. In contrast, if we aim to avoid offline communication and require PIR schemes with silent preprocessing, we must stomach substantial additional communication for each PIR query. This leads to the following question:

Can we construct a PIR without offline communication obtaining high-throughput and low query communication?

In this work, we answer in the affirmative with a new class of PIR protocols that obtain both high-throughput and low query communication without offline communication.

1.1. The Case for Server-side Preprocessing with no Offline Communication

PIR with server-side preprocessing is very useful in many applications of PIR. We list four circumstances under which server-side preprocessing is beneficial, compared to other approaches in the literature. This is not an exhaustive list and we leave more applications for PIR with server-side preprocessing to future work.

Singular Query. A common assumption in the PIR literature is that the user makes many queries such that any setup cost may be amortized. Under this assumption, PIR protocols with large initial offline communication costs are justified as the per-query costs become small over time. For example, protocols that require server-stored cryptographic keys [4], [67], [62] assume that these keys are used for many queries. Protocols that use database-dependent client-stored hints assume that the downloaded hint is used across multiple queries over some longer period of time [43], [30]. Mazmudar et al. [60] argued that in many applications (e.g., IPFS), the client may only make one query to a server. In these settings, protocols with large initial costs make the usage of PIR infeasible. It is also not hard to see the PIR applicability challenges extending to even more use cases where clients only perform a few number of PIR queries infrequently over a long period of time. In such applications, PIR with server-side preprocessing that avoid large setup costs are critical.

Cold Start. In some applications, the client may not have enough time to perform offline communication. An example is the use of PIR for device enrollment in Chrome devices [80]. Given that the PIR query is performed upon startup, there has not been any opportunity to exchange information in the offline phase. Within these usages of PIR, the initial setup must be performed online at the same time as the PIR query. As a result, protocols with expensive preprocessing (such as streaming the database or uploading large cryptographic keys) would incur substantial costs to use PIR. In contrast, PIR protocols with server-side preprocessing may be used immediately even if the client has no time to perform offline communication.

User Anonymity. There has been work that suggests using PIR in the context of anonymity networks such as Tor [66]. In this context, the usage of PIR requiring per-client storage on the server, also known as a per-client state, is contradictory to the goal of anonymity. If a per-client state is used, the server can correlate queries made by the client, which reveals metadata. Henzinger et al. [43, Appendix B] also demonstrated the risk of client state recovery attacks by an active adversary when the client reuses a secret key. Once again, PIR with server-side preprocessing has none of these drawbacks and may be used in anonymity networks without these privacy concerns.

Large User Bases. We can also consider setting where PIR is used to serve information to large user bases. For example, one can the consider password leak checks application that is served to all browser users [51], [3]. Once again, we could consider PIR protocols with large preprocessing costs. For protocols with server-stored keys, the PIR service must store a large per-client state for every browser user. The storage costs grow linearly in the user base size, which would be infeasible in this application. Similarly, PIR protocols with client-stored hints would require executing an expensive offline phase with each browser user that is once again challenging to scale. In either of the above types of PIR protocols, if the client loses its storage for any reason, the entire offline phase must be executed again. PIR with server-side preprocessing does not suffer from any of these scalability issues.

Database Changes. We can also consider the natural setting where the database may update on a frequent basis. For PIR with client-held hints, any application with a constantly changing database will require modifying the offline database-dependent hints that have been given to clients. In many cases, this requires either re-executing the offline phase or running some specialized protocol for updating client hints [81], [44]. Nevertheless, the necessity of executing an additional protocol with every client for each database update adds substantial complexity to the system. In contrast, PIR with server-side preprocessing only requires the server to locally compute and update its internal database representation to handle database changes.

1.2. Technical Overview & Contributions

Efficient PIR with Server-side Preprocessing. As our main contribution, we present a new PIR protocol, InsPIRe, that simultaneously obtains high-throughput and low query communication using only server-side preprocessing in the CRS model, without any offline communication. Specifically, we obtain higher throughput and lower communication than all state-of-the-art PIR protocols in the CRS and silent preprocessing model such as YPIR [63] and HintlessPIR [53]. We observe this advantage across various database sizes with different payload sizes. For example, for retrieving a 32 KB payload from a 32 GB database, InsPIRe has 10% higher throughput and 67-90% lower communication costs compared to HintlessPIR [53] and YPIR [63]. Moreover, we can parameterize InsPIRe to optimize for other metrics, e.g., lower communication. InsPIRe can achieve 78-93% reduction in communication whilst still having better throughput than HintlessPIR and YPIR. Figure 1 shows a full comparison with related work.

Ring Packing with Preprocessing. At the core of our techniques, we develop a novel ring packing algorithm InspiRING that translates LWE ciphertexts into a RLWE ciphertext (Section 3) using three steps. In the first step, LWE ciphertexts are transformed to an intermediate representation with the message encoded as a constant term of the plaintext polynomial. This is achieved by carefully applying the trace function, expressed in terms of the two generators of the Galois group. Next, ciphertexts are aggregated to obtain one ciphertext in the intermediate representation that encodes all the original messages. Finally, the result is converted back to the standard RLWE ciphertext format using only two cryptographic keys. In the context of PIR, InspiRING requires substantially less cryptographic material to compress PIR responses while simultaneously obtaining better throughput.

Homomorphic Polynomial Evaluation. We also present a novel technique to further reduce the size of the PIR response by representing database entries using polynomials. We show how to reduce the PIR response size by evaluating polynomials homomorphically in an efficient manner (Section 6).

Applications. We show that our new protocol, InsPIRe, can be applied to improve efficiency in several real-world deployments. To showcase the improvements across a wide-range of realistic database sizes and shapes, we show that InsPIRe can improve the efficiency of two applications that heavily rely upon PIR with silent preprocessing. First, we show that InsPIRe is a better alternative for issuing private queries in the InterPlanetary File System (IPFS), where PIR queries are issued over databases of various sizes with varying payload sizes. In this context, our work can improve communication by up to 51% and computation by up to 86% over prior work [60]. Secondly, we consider the private device enrollment that is deployed by Google [80] where PIR is used to privately check membership of identifiers in a database. We show InsPIRe requires less than 300 KB communication to respond to a request in about 800 ms, for a database of 40M identifiers. In summary, InsPIRe enables improvements across a wide set of applications and databases.

2. Preliminaries

Basic Notation. We use lowercase bold letters to denote vectors and uppercase bold letters to denote matrices. We use $[a_1, \ldots, a_n]$ to denote an $1 \times n$ row matrix and $[a_1, \ldots, a_n]^\top$ to denote an $n \times 1$ column matrix. For a set R, we use $R^n = R^{n\times 1}$ to denote the set of $n \times 1$ column vectors over R while $R^{1\times n}$ to denote the set of $1 \times n$ row vectors over R. We will use R and $R^{1\times 1}$ interchangeably. For a vector $\mathbf{a}$, we use $\mathbf{a}[j]$ to index the j-th element and $\mathbf{a}[i : j]$ to denote a slice in interval $[i, j)$. For a polynomial p, we denote $p[k_1 : k_2]$ as the subset of coefficients of p, with degree in $[k_1, k_2)$. If it is clear from the context, we will treat a row vector as a tuple and vice versa and interchange them freely without explicit casting.

For a discrete probability distribution D defined over the set S, we use $x \leftarrow D(S)$ to denote sampling an element from S according to the distribution D. We use $U(S)$ to denote a uniform distribution of a set S.

We use λ to denote the security parameter.

Gaussian and Subgaussian Variables. A random variable X is subgaussian with parameter σ if $\Pr[|X| > t] \leq 2\exp(-\pi t^2/\sigma^2)$ for all $t \geq 0$. A discrete Gaussian variable with width σ is subgaussian with parameter σ. If X is subgaussian with σ, cX is subgaussian with parameter $|c|\sigma$. The sum $X_1 + \ldots + X_k$ of independent subgaussian variables (with parameters $\sigma_1, \ldots, \sigma_k$) is subgaussian with parameter $\sqrt{\sum_i \sigma_i^2}$. See [74] for more details.

Cyclotomic Rings. We use cyclotomic rings $R = \mathbb{Z}[X]/(X^d + 1)$ and $R_q = \mathbb{Z}_q[X]/(X^d + 1)$, where d is a power of two. Ring elements $p(X) \in R$ may be denoted as p if clear from context.

LWE, RLWE, and MLWE. In our work, we will use both LWE- and RLWE-based private key encryption. We use $\chi(\mathbb{Z})$ to denote an error distribution over $\mathbb{Z}$ and $\chi(R)$ to denote an error distribution over the coefficients of R. We naturally extend to $\chi(\mathbb{Z}^n)$ for an error distribution over $\mathbb{Z}^n$ and $\chi(R^n)$ for an error distribution over R^n.

An LWE encryption of a message $m \in \mathbb{Z}_p$ is the pair $(\mathbf{a}, b) \in \mathbb{Z}_q^d \times \mathbb{Z}_q$ such that $b = -\langle \mathbf{a}, \mathbf{s} \rangle + e + \Delta \cdot m$ where $\mathbf{s} \leftarrow \chi(\mathbb{Z}^d)$ is a secret key, $e \leftarrow \chi(\mathbb{Z})$ is a small error value, and Δ is a scaling factor that is typically $\Delta = \lfloor q/p \rfloor$. Note, the secret key $\mathbf{s}$ enables decryption message m by computing $b + \langle \mathbf{a}, \mathbf{s} \rangle = e + \Delta \cdot m \in \mathbb{Z}_q$. When e is small, this allows retrieving $m \in \mathbb{Z}_p$ by rounding.

An RLWE [36] encryption of a message $m \in R_p$ consists of the pair $(a, b) \in R_q \times R_q$ such that $b = -as + e + \Delta \cdot m$ where $s \leftarrow \chi(R)$ is a secret key, $e \leftarrow \chi(R)$ is an error polynomial and Δ is a scaling factor that is typically $\Delta = \lfloor q/p \rfloor$. Given the secret key s, it is possible to decrypt the message m by computing $b + as = e + \Delta \cdot m \in R_q$ such that m can be retrieved by rounding as long as e is small.

While we don't directly use it in our work, we draw some useful analogies to MLWE encryption schemes [13], [50] in our packing algorithm in Section 3. We refer to Section 2.3 in [13] for details.

Throughout the paper, we use d to denote both the dimension of LWE samples and the degree of RLWE samples. In both encryption schemes, we will use the version where the public random components of the ciphertexts are fixed, which remain secure as long as the secret keys are re-sampled (see [75]).

Gadget Matrices. [64] Given a modulus $q \in \mathbb{N}$ and a decomposition base $z \in \mathbb{N}$, we use $\mathbf{g_z} = [1, z, \ldots, z^{\ell-1}]^\top \in \mathbb{Z}_q^\ell$ where $\ell = \lceil \log q / \log z \rceil$ to denote a one-dimensional gadget matrix. We use $\mathbf{g_z^{-1}} : \mathbb{Z}_q \to \mathbb{Z}^{1\times\ell}$ to denote the base-z digit decomposition operator on an integer in $\mathbb{Z}_q$, where each component of the output vector is an integer in $[-z/2, z/2)$. We extend $\mathbf{g_z^{-1}} : \mathbb{Z}_q \to \mathbb{Z}^{1\times\ell}$ to $\mathbf{g_z^{-1}} : R_q \to R^{1\times\ell}$ to operate over the ring R_q where the digit decomposition is applied to each coefficient of the element in R_q independently.

RLWE-RGSW External Product [23]. For $\mathbf{g_z} \in \mathbb{Z}_q^\ell$ and identity matrix $\mathbf{I}_2$, let $\mathbf{G}_{2,z} = \mathbf{I}_2 \otimes \mathbf{g}_z \in R_q^{2\ell\times 2}$. An RGSW encryption of message $m \in R_p$ is $[\mathbf{a}', \mathbf{b}'] \in R_q^{2\ell\times 2}$:

$$[\mathbf{a}', \mathbf{b}'] := [\mathbf{a}, -s\mathbf{a} + \mathbf{e}] + m \cdot \mathbf{G}_{2,z}$$

where $\mathbf{a} \leftarrow U(R_q^{2\ell})$ is the random component vector and $\mathbf{e} \leftarrow \chi(R_q^{2\ell})$ is the noise vector. For a RLWE ciphertext $\mathsf{RLWE}(m_0) = (a, b)$ and a RGSW ciphertext $\mathsf{RGSW}(m_1) = [\mathbf{a}', \mathbf{b}']$, the external product is defined as:

$$\begin{aligned} \boxdot :&\mathsf{RLWE}(m_0) \times \mathsf{RGSW}(m_1) \to \mathsf{RLWE}(m_0 m_1) \\ &(a'', b'') := [\mathbf{g_z^{-1}}(a), \mathbf{g_z^{-1}}(b)] \cdot [\mathbf{a}', \mathbf{b}']. \end{aligned}$$

The result is a RLWE ciphertext of the product $m_0 m_1$.

RLWE Key-Switching. Given a RLWE ciphertext $(a, b) = (a, -as + m + e)$ encrypted under a secret key s, the goal of the key-switching procedure is to transform the ciphertext (a, b) to be encrypted under a different key s'. Let z and ℓ be the gadget parameters. The key-switching procedure consists of a pair of algorithms $\mathsf{KS} = (\mathsf{Setup}, \mathsf{Switch})$ defined as:

- $\mathsf{KS.Setup}(s, s')$: On input source secret $s \in R_q$, target secret $s' \in R_q$
  1) Sample $\mathbf{w} \leftarrow U(R_q^\ell)$ and $\mathbf{e} \leftarrow \chi(R^\ell)$.
  2) Compute $\mathbf{y} \leftarrow -s' \cdot \mathbf{w} + s \cdot \mathbf{g}_z + \mathbf{e}$
  3) Output $\mathbf{K} = [\mathbf{w}, \mathbf{y}] \in R^{\ell\times 2}$ as the key-switching matrix.
- $\mathsf{KS.Switch}((a, b), \mathbf{K})$: On input a RLWE ciphertext $(a, b) = (a, -as + m + e)$, a key-switching matrix $\mathbf{K} \in R^{\ell\times 2}$ that switches from s to s'
  1) Output $(a', b') \leftarrow (0, b) + \mathbf{g_z^{-1}}(a) \cdot \mathbf{K}$.

We refer to Section 2.4 in [18] for more details.

Galois Group. For the cyclotomic ring R that we consider in this work, the Galois group is a group of ring automorphisms, with each element:

$$\begin{aligned} \tau_g &: R \to R \\ \tau_g(p) &= p(X^g) \end{aligned}$$

for $g \in \{1, 3, \ldots, 2d - 1\}$. The Galois group is isomorphic to the additive group $\mathbb{Z}_{d/2} \times \mathbb{Z}_2$ and can be generated using two generators [34].

For a non-negative integer i, we use τ_g^i (resp. τ_g^{-i}) to denote the Galois group element corresponding to i applications of τ_g (resp. τ_g^{-1}).

For a matrix $\mathbf{M} \in R^{n\times m}$, we extend the use of τ_g and denote $\tau_g(\mathbf{M})$ to mean that τ_g is applied to each entry of $\mathbf{M}$. We extend this notion naturally to τ_g^i as well.

Independence Heuristic. Following prior works such as [4], [67], [62], [53], we assume the independence heuristic where the error terms of all intermediate computations are independent. As a result, we analyze the variance of the noise vector as opposed to the worst case magnitude. In particular, we leverage the fact that variance is additive for independent subgaussian variables to analyze noise.

PIR. We consider single-server PIR schemes with single roundtrip queries consisting of the tuple of four algorithms $(\mathsf{Setup}, \mathsf{Query}, \mathsf{Respond}, \mathsf{Extract})$:

- $\mathsf{Setup}(1^\lambda, \mathbf{D}) \rightarrow (\mathsf{pp}, \mathbf{D}')$: Executed by the server, this algorithm receives security parameter 1^λ and database $\mathbf{D}$, and outputs public parameters pp and encoded database $\mathbf{D}'$. Output pp is shared with the client.
- $\mathsf{Query}(\mathsf{pp}, \mathsf{idx}) \rightarrow (\mathsf{st}, \mathsf{qry})$: Executed by the client, this algorithm receives public parameters pp and the DB index idx, and outputs client state st and query qry.
- $\mathsf{Respond}(\mathsf{pp}, \mathbf{D}', \mathsf{qry}) \rightarrow \mathsf{resp}$: Executed by the server, this algorithm receives public parameters pp, encoded database $\mathbf{D}'$, and query qry, and outputs response resp.
- $\mathsf{Extract}(\mathsf{pp}, \mathsf{st}, \mathsf{resp}) \rightarrow \mathsf{entry}$: Executed by the client, this algorithm receives public parameters pp, client state st, and server's response resp and outputs the entry.

Our work follows standard security and correctness definitions from prior works (such as [43], [63]). We defer formal descriptions to Appendix A.

2.1. The Common Reference String (CRS) Model

In this model, we assume that all parties have access to a global string, generated by a trusted entity. Under this assumption, we fix the random components of the LWE/RLWE ciphertexts, and this scheme remains secure as long as fresh secret keys are generated for each query (with multiplicative factor security loss proportional to the number of queries). Prior work such as SimplePIR [43], HintlessPIR [53], and YPIR [63] also work in this model.

2.2. Convention of our Pseudocode Presentation

In the CRS model, the message-independent components of the ciphertexts are fixed before the online phase, enabling precomputation of protocol parts dependent solely on these message-independent components. Typically, the offline and the online phases of a protocol are described using separate sets of pseudocode. Instead, we combine both phases into a single, integrated pseudocode description.

To clearly distinguish these phases within a unified structure, operations performable during offline preprocessiong are highlighted. This convention requires a two-pass interpretation of our protocol:

1) **Offline Preprocessing Pass:** Only highlighted portions are executed and generated data is cached. Non-highlighted parts are disregarded.
2) **Online Execution Pass:** Non-highlighted portions are executed using cached values from highlighted portions.

We believe this integrated style offers a more intuitive grasp of the interplay between precomputed and online operations without having to jump back and forth between them.

3. Ring Packing with Preprocessing

In this section, we present $\mathsf{InspiRING}$, our ring packing construction transforming LWEs into a RLWE ciphertext. $\mathsf{InspiRING}$ is specifically designed for the CRS model where the random components of ciphertexts are fixed and can be preprocessed. $\mathsf{InspiRING}$ will be the core building block of our low query communication and high-throughput PIR protocol. Prior LWE-based PIR schemes [42], [53] essentially used existing packing/bootstrapping algorithms in a blackbox manner, which mainly target a more general setting where the random parts of the ciphertexts are not necessarily known ahead and can be preprocessed [1]. Our observation is that, if we design algorithms where offline preprocessing utilize fixed random components of the ciphertexts, then we can obtain a more efficient ring packing algorithm. In comparison to the CDKS ring packing [18] used in YPIR [63], our construction is practically more efficient and has better analytical and concrete noise growth while only requiring *two key-switching matrices*. In contrast, CDKS [18] requires $\lg d$ key-switching matrices, increasing PIR communication.

Notational Convention. For the presentation in this section, we adopt a notational convention that omits the message scaling factor Δ and the error term e and assume they are part of the plaintext message. For example, the LWE pseudorandom component $b = -\langle \mathbf{a}, \mathbf{s} \rangle + e + \Delta m$ will be written as $b = -\langle \mathbf{a}, \mathbf{s} \rangle + m'$ where $m' = e + \Delta m$. For this section, we emphasize that this notational convenience does not impact the correctness or security of the described algorithm. On the other hand, the noise introduced during the packing will be written and analyzed explicitly.

[1] For PIR, these schemes did take advantage of preprocessing the random components of the ciphertexts to improve the online computation, but only by "simulating" the underlying algorithm in the offline phase.

3.1. Revisiting CDKS [18] Packing

Before we describe our construction, we will first re-examine the CDKS packing algorithm [18] which is also used in YPIR [63]. While certain insights from the CDKS algorithm will be directly incorporated into our construction, our primary focus here is to understand why CDKS requires many key-switching matrices. In this discussion, d represents both the dimension of the initial LWE ciphertexts and the degree of the target ring.

The CDKS algorithm begins by interpreting the input LWE ciphertexts denoted as $(\mathbf{a}, b = -\langle \mathbf{a}, \mathbf{s} \rangle + m) \in \mathbb{Z}_q^d \times \mathbb{Z}_q$ as elements of $R_q \times R_q$ where $R_q = \mathbb{Z}_q[X]/(X^d + 1)$. This embedding process treats the LWE ciphertext as an RLWE ciphertext, $(\tilde{a}, \tilde{b}) \in R_q \times R_q$. In this interpretation, $\tilde{b}$ corresponds to the original b (treated as a constant term polynomial), and $\tilde{a}$ is $\sum_{i=0}^{d-1} \mathbf{a}[i] \cdot X^{-i}$. Consequently, $\tilde{b}$ can be formulated as:

$$\tilde{b} = -\tilde{a}\tilde{s} + \tilde{m} \pmod{q}. \tag{1}$$

In this equation, $\tilde{s} = \sum_{i=0}^{d-1} \mathbf{s}[i] \cdot X^i$ represents the LWE secret $\mathbf{s}$ as a polynomial. A key outcome of this transformation is that the constant term of $\tilde{m}$ holds the original message m. However, the other coefficients of $\tilde{m}$ are filled with arbitrary values due to the embedding. If these other coefficients were zero, packing multiple messages would be simple: multiplying the RLWE ciphertext by X^i cyclically shifts its coefficients by i positions, and thus, by applying suitiable X^i multiplications and summing the resulting ciphertexts, a single RLWE ciphertext could embed all the LWE plaintexts.

For ring packing, it's necessary to zero out specific coefficient slots. This creates space for the plaintext messages from other ciphertexts to be merged into a single, consolidated RLWE ciphertext.

CDKS addresses this challenge by incrementally merging the input ciphertexts through a recursive process. This packing procedure can be viewed as a complete binary tree with depth of $\lg(d)$. The leaves of this tree are the RLWE-embedded LWE ciphertexts (Eq. 1). At each internal node, two RLWE ciphertexts are combined into a single, valid RLWE ciphertext. This merging step employs an automorphism to zero out the necessary coefficient slots of the plaintext message. Specifically, these automorphisms preserve certain coefficients (for already packed messages) while negating others. When a ciphertext is added to its automorphic image, the negated coefficients cancel each other, therby freeing up slots for the merging process. A subesequent key-switching operation is then performed to eliminate the extraneous term from the automorphism.

A critical aspect of this method is that the specific automorphism applied depends on the number of messages already packed. Consequently, each level in the recursion tree utilizes a distinct automorphism. This necessitates a unique key-switching matrix at each level to remove the extraneous term generated by the automorphism. As a result, this packing strategy inherently requires $\lg(d)$ key-switching matrices, which results in large cryptographic material.

It is important to note that the CDKS algorithm was not initially developed with the CRS model in mind. In the CRS model, the random components of ciphertexts are known and can be preprocessed. This raises a crucial question: can we leverage the CRS model to devise a more efficient ring packing algorithm to reduce the substantial cryptographic overhead of the existing methods such as CDKS?

3.2. Packing with Two Key-switching Matrices

We affirm the above question by presenting our packing algorithm that can efficiently translate d input LWE ciphertexts into a single RLWE ciphertext using only *two key-switching matrices*. Here, d is the dimension of the input LWE ciphertexts and also the degree of the target ring. Our construction is divided into three stages as follows:

1) *LWE to Intermediate Ciphertexts:* The goal of this initial stage is to reinterpret each LWE ciphertext as a *larger* intermediate representation (reminiscent of a Module-LWE ciphertext [13], [50]). Crucially, this intermediate ciphertext encrypts the original LWE message as a *constant term polynomial* and retains necessary homomorphic properties for packing. This transformation is important because a) the constant term message enables straightforward aggregation of multiple such ciphertexts (Stage 2) and b) a carefully designed "correlated structure" in these intermediate ciphertexts enables our construction to ultimately rely on just two key-switching matrices (Stage 3).
2) *Aggregation of Intermediate Ciphertexts:* Stage 2 leverages homomorphic properties of the transformed ciphertexts to combine multiple such ciphertexts into one consolidated ciphertext. Since each transformed ciphertext embeds the original LWE message as a constant term polynomial, the aggregation becomes straightforward. The aggregated ciphertext's plaintext polynomial embeds the input LWE messages into distinct coefficients.
3) *Conversion to RLWE Ciphertext:* The final stage converts the intermediate ciphertext (which now holds all packed messages) into a standard, compact RLWE ciphertext. This is achieved via an iterative key-switching procedure that "collapses" the components, efficiently utilizing the correlated structure and Galois automorphic images of two elementary key-switching matrices.

At first, the transformation of LWE ciphertexts into significantly larger intermediate ciphertext in the initial stage might appear counterintuitive to achieving efficiency. However, the key to this approach lies in the nature of these expanded structures. The bulk of the data within these intermediate forms is determined solely by the fixed random components of the original LWE ciphertexts and the key-switching matrices. In the CRS model where these random elements are fixed and known beforehand, the computationally intensive operations involved in constructing and processing these larger structures can be mainly moved to an offline preprocessing phase. This idea of investing more in offline computation is a recurring principle across all the three stages of our packing algorithm. It allows a substantial reduction in online computation time and minimizes overhead associated with cryptographic materials during the online packing process.

Stage 1: LWE to Intermediate Ciphertext. The first stage transforms each LWE input ciphertext into a specially structured ciphertext that resembles a Module-LWE (MLWE) ciphertext [13], [50]. Specifically, we will construct an intermediate ciphertext where the random component and the secret key are *modules* over the ring $R_q = \mathbb{Z}_q[X]/(X^d + 1)$. The pseudorandom component is an element in R_q and will embed the original LWE message as a constant term polynomial, masked by an inner product between the random component and the secret key. However, the crucial difference from a typical MLWE ciphertext is the carefully designed correlations among the secret key components. Looking ahead, this correlated structure of the secret key components is essential for enabling the use of two key-switching matrices in Stage 3.

The core idea begins with embedding the input LWE ciphertext, $(\mathbf{a}, b = -\langle \mathbf{a}, \mathbf{s}\rangle + m)$, as an RLWE ciphertext $(\tilde{a}, \tilde{b} = -\tilde{a}\tilde{s} + \tilde{m})$ as in CDKS [18] (see Eq. 1). However, as discussed previously, the non-zero coefficients of the embedded plaintext message need to be zeroed out to enable packing. CDKS handles this by incrementally merging the ciphertexts in a recursive fashion, only freeing up necessary coefficient slots required for that stage of merging.

Instead, our insight is to transform the RLWE interpretation $(\tilde{a}, \tilde{b})$ into a carefully designed intermediate representation *upfront* that is more amenable to packing. Specifically, we aim to construct an intermediate ciphertext that embeds the original LWE message as a constant term polynomial, e.g. message of the form $\hat{m}(X) = m$. Looking ahead, this "sparsity" of the message allows multiple such ciphertexts to be aggregated without interference between messages.

To achieve this, we leverage the following formulation of the trace function (defined as the sum of Galois conjugates [34]) expressed in terms of two generators of the Galois group. We recall that for the cyclotomic ring R in this work, the Galois group is isomorphic to the additive group $\mathbb{Z}_{d/2} \times \mathbb{Z}_2$ [34]. τ_g and τ_h in Lemma 1 corresponds to the two generators $(1, 0)$ and $(0, 1)$ in $\mathbb{Z}_{d/2} \times \mathbb{Z}_2$. We leave the proof to Appendix D.

Lemma 1. *Let $p(X) \in \mathbb{Z}[X]/(X^d + 1)$ such that $p(X) = \sum_{i=0}^{d-1} c_i X^i$ where d is a power of two. Let $g = 5$ and $h = 2d-1$, and define Tr : $R \to R$ as*

$$\mathit{Tr}(p) := \sum_{j=0}^{d/2-1} \tau_g^j(p) + \tau_h \circ \tau_g^j(p).$$

Then $\mathit{Tr}(p) = d \cdot c_0$.

The key insight to forming the intermediate ciphertext lies in how its components are constructed from the initial RLWE-like interpretation $(\tilde{a}, \tilde{b})$. The first (random) component of the new intermediate ciphertext is a vector of d polynomials $\hat{\mathbf{a}} \in R_q^d$. The first half of $\hat{\mathbf{a}}$ is defined as $\hat{\mathbf{a}}[j] = d^{-1} \cdot \tau_g^j(\tilde{a})$ for $j < d/2$. The second half is the automorphic image of the first half by τ_h, e.g. $\hat{\mathbf{a}}[d/2 : d] = \tau_h(\hat{\mathbf{a}}[0 : d/2])$.

The original $\tilde{b}$ is re-interpreted to serve as the pseudorandom component of this new intermediate ciphertext. Specifically, suppose we define the corresponding intermediate secret key vector (module) as $\hat{\mathbf{s}} \in R_q^d$ where the first half is $\hat{\mathbf{s}}[j] = \tau_g^j(\tilde{s})$ and the second half is the automorphic image of the first half by τ_h, e.g. $\hat{\mathbf{s}}[d/2 : d] = \tau_h(\hat{\mathbf{s}}[0 : d/2])$. Then, the component $\tilde{b}$ satisfies:

$$\tilde{b} = -\langle \hat{\mathbf{a}}, \hat{\mathbf{s}} \rangle + \hat{m} \pmod q \tag{2}$$

We note that this equation is derived from the application of the operator Tr from Lemma 1 appropriately on Eq. 1 (detailed in Appendix B). In particular, the pair $(\hat{\mathbf{a}}, \tilde{b})$ forms an MLWE-like ciphertext encrypting message $\hat{m}(X) = m$.

Importantly, the structure $(\hat{\mathbf{a}}, \tilde{b})$ maintains necessary homomorphic properties (akin to MLWE) for packing. It supports plaintext absorption (multiplication to each component) and homomorphic addition (component-wise addition), which is crucial for Stage 2.

In summary, Stage 1 converts an LWE ciphertext $(\mathbf{a}, b)$ (encrypting m) into an intermediate ciphertext encrypting $\hat{m}(X) = m$, which we denote as $\mathsf{IRCtx}(\hat{m})$ (stands for Intermediate Representation CipherTeXt).

We present the formal pseudocode of this transformation in Algorithm 1's subroutine TRANSFORM. See Appendix B for further detailed mathematical derivation of $\mathsf{IRCtx}(\hat{m})$.

Stage 2: Aggregation of Intermediate Ciphertexts. Stage 2 combines d IRCtx ciphertexts (from Stage 1) into a single IRCtx, packing their messages as distinct coefficients.

Consider d input LWE ciphertexts: $(\mathbf{a}_0, b_0), \ldots, (\mathbf{a}_{d-1}, b_{d-1})$, where the k-th ciphertext encrypts m_k. Applying Stage 1's TRANSFORM to each $(\mathbf{a}_k, b_k)$ yields: $\mathsf{IRCtx}(\hat{m}_k) \leftarrow \mathrm{TRANSFORM}(\mathbf{a}_k, b_k)$, where $\hat{m}_k(X) = m_k$ is a constant term.

We then multiply each $\mathsf{IRCtx}(\hat{m}_k)$ by X^k and sum the results. Leveraging the homomorphic properties of IRCtx, we obtain the following:

$$\sum_{k=0}^{d-1} \mathsf{IRCtx}(\hat{m}_k) \cdot X^k = \mathsf{IRCtx}(\sum_{k=0}^{d-1} \hat{m}_k X^k).$$

We denote this aggregated plaintext message as $\hat{m}_{agg} := \sum_{k=0}^{d-1} \hat{m}_k X^k$ and the corresponding ciphertext encrypting it $\mathsf{IRCtx}(\hat{m}_{agg}) = (\hat{\mathbf{a}}_{agg}, \tilde{b}_{agg})$. The above operation positions the original plaintext messages in the LWE ciphertexts, m_k, into the coefficients of $\hat{m}_{agg}$. However, the intermediate ciphertext produced at this stage, $\mathsf{IRCtx}(\hat{m}_{agg})$, is considerably large, comprising $d+1$ elements from the ring R_q. The formal pseudocode of this stage is provided in Algorithm 1's PACK procedure, Line 4.

Stage 3: Conversion to RLWE Ciphertext. The aggregated intermediate ciphertext $\mathsf{IRCtx}(\hat{m}_{agg}) = (\hat{\mathbf{a}}_{\mathbf{agg}}, \tilde{b}_{agg}) \in R_q^d \times R_q$ is currently represented by $d+1$ ring elements in R_q and is effectively encrypted under a vector of secret key shares $\hat{\mathbf{s}}$ with $\hat{\mathbf{s}}[j] = \tau_g^j(\tilde{s})$ and $\hat{\mathbf{s}}[j+d/2] = \tau_h \circ \tau_g^j(\tilde{s})$ for $j < d/2$.

As a reminder, $\tilde{b}_{agg}$ can be expressed as (from Eq. 2):

$$\begin{aligned}\tilde{b}_{agg} &= -\langle \hat{\mathbf{a}}_{agg}, \hat{\mathbf{s}} \rangle + \hat{m}_{agg} \pmod q \\ &= -\left(\sum_{j=0}^{d/2-1} \hat{\mathbf{a}}_{agg}[j] \cdot \tau_g^j(\tilde{s})\right) \pmod q \\ &\quad -\left(\sum_{j=0}^{d/2-1} \hat{\mathbf{a}}_{agg}[j+d/2] \cdot \left(\tau_h \circ \tau_g^j(\tilde{s})\right)\right) + \hat{m}_{agg} \pmod q.\end{aligned}$$

To obtain a standard two-component RLWE ciphertext encrypted under a single base secret key $\tilde{s}$, we must reduce the number of these components by re-expressing the masking part $\langle \hat{\mathbf{a}}_{\mathbf{agg}}, \hat{\mathbf{s}} \rangle$ with respect to a single base secret key $\tilde{s}$.

At a high level, this reduction is achieved through an iterative process that systematically "collapses" the components. In each step, a RLWE key-switching procedure (see Section 2 for full description) is applied, which re-expresses the pseudorandom component originally masked by one secret key share $\tau_g^k(\tilde{s})$ (resp. $\tau_h \circ \tau_g^k(\tilde{s})$) so that its masking now relies on a different share $\tau_g^{k-1}(\tilde{s})$ (resp. $\tau_h \circ \tau_g^{k-1}(\tilde{s})$), effectively reducing the number of distinct key shares in the overall ciphertext masking. This iterative process eventually consolidates all parts under two key shares $\tilde{s}$ and $\tau_h(\tilde{s})$.

This process relies on a set of key-switching matrices, but all necessary key-switching matrices are automorphic images of a single elementary key-switching matrix $\mathbf{K}_g = \mathsf{KS.Setup}(\tau_g(\tilde{s}), \tilde{s})$. This is because if $\mathbf{K}_g$ transforms a ciphertext component from being encrypted under $\tau_g(\tilde{s})$ to $\tilde{s}$, then its automorphic image $\tau_g^k(\mathbf{K}_g)$ (resp. $\tau_h \circ \tau_g^k(\mathbf{K}_g)$) will transform a component from $\tau_g^{k+1}(\tilde{s})$ to $\tau_g^k(\tilde{s})$ (resp. $\tau_h \circ \tau_g^{k+1}(\tilde{s})$ to $\tau_h \circ \tau_g^k(\tilde{s})$).

This creates a "telescoping" effect where each step in the reduction uses a key related to $\mathbf{K}_g$ by an appropriate Galois automorphism, allowing the component reduction to be performed efficiently using variants of just one base key-switching matrix. After $d-2$ such iterations, the number of random components of $\mathsf{IRCtx}(\hat{m}_{agg})$ is reduced to two, with the resultant ciphertext encrypted under two secret key shares $\tilde{s}$ and $\tau_h(\tilde{s})$. To eliminate the key share $\tau_h(\tilde{s})$, we perform the final key-switching using $\mathbf{K}_h = \mathsf{KS.Setup}(\tau_h(\tilde{s}), \tilde{s})$. The final result is a standard two-component RLWE ciphertext (a_{fin}, b_{fin}), where $b_{fin} = -a_{fin} \cdot \tilde{s} + \hat{m}_{agg} + e_{tot}$ and e_{tot} is the total accumulated noise from the key-switchings.

The formal pseudocode of this stage is presented in Algorithm 1's subroutine COLLAPSE. The full details of this iterative component reduction algorithm, including the specific key-switching steps, are provided in Appendix C.

Putting It Together. The discussions in the preceding sections can be summarized as the unified algorithm InspiRING.Pack. The pseudocode is in Algorithm 1. We remind the readers that the highlighted parts of the algorithm correspond to the preprocessable components in the CRS model (Section 2.2. We denote $\tilde{s}$ is the natural polynomial interpretation of the LWE secret $\mathbf{s}$.

Preprocessing in the CRS Model. A key strength of InspiRING algorithm is its inherent suitability for preprocessing in the CRS model where the random components of the ciphertexts are fixed. As detailed in the preceding stages, a substantial portion of the data manipulated during the packing process depends solely on the random components of the initial LWE ciphertexts and the key-switching matrices $\mathbf{K}_g$ and $\mathbf{K}_h$. This characteristic allows for significant portions of the computation to be performed offline, making the online packing phase very efficient.

The benefits of this approach are evident in the first two stages. In Stage 1, the random components of the resultant $\mathsf{IRCtx}(\hat{m}_k)$ ciphertexts are entirely determined by the random vectors of the input LWE ciphertexts. Similarly, in Stage 2, the bulk of operations involves combining these preprocessable random components. The online aspects of these stages are minimal: the re-interpretation of the LWE pseudorandom components b_k into their ring element counterparts $\tilde{b}_k$ requires no

Algorithm 1 InspiRING algorithm

```
procedure Pack($[\mathbf{A}, \mathbf{b}] \in \mathbb{Z}_q^{d\times(d+1)}, \mathbf{K}_g = [\mathbf{w}_g, \mathbf{y}_g], \mathbf{K}_h = [\mathbf{w}_h, \mathbf{y}_h] \in R_q^{\ell\times 2}$)
    for $k = 0 \dots d-1$ do
        $(\hat{\mathbf{a}}_k, \tilde{b}_k) \leftarrow$ Transform($\mathbf{A}[k], \mathbf{b}[k]$)
    $(\hat{\mathbf{a}}_{\mathbf{agg}}, \tilde{b}_{agg}) \leftarrow \sum_{k=0}^{d-1}(\hat{\mathbf{a}}_k, \tilde{b}_k) \cdot X^k$
    $(a_{fin}, b_{fin}) \leftarrow$ Collapse($(\hat{\mathbf{a}}_{agg}, \tilde{b}_{agg}), [\mathbf{w}_g, \mathbf{y}_g], [\mathbf{w}_h, \mathbf{y}_h]$)
    return $(a_{fin}, b_{fin}) \in R_q \times R_q$
```

— Subroutines —

```
procedure Transform($(\mathbf{a}, b) \in \mathbb{Z}_q^d \times \mathbb{Z}_q$)
    $\tilde{a} \leftarrow d^{-1}\sum_{i=0}^{d-1}\mathbf{a}[i]X^{-i}$                    ▷ $d \cdot d^{-1} = 1 \mod q$
    $\tilde{b} \leftarrow b$
    Initialize $\hat{\mathbf{a}} \in R_q^d$
    for $j = 0 \dots d/2-1$ do
        $\hat{\mathbf{a}}[j] \leftarrow \tau_g^j(\tilde{a})$
        $\hat{\mathbf{a}}[j+d/2] \leftarrow \tau_h \circ \tau_g^j(\tilde{a})$
    return $(\hat{\mathbf{a}}, \tilde{b}) \in R_q^d \times R_q$
procedure Collapse($(\hat{\mathbf{a}}_{agg}, \tilde{b}_{agg}) \in R_q^d \times R_q, \mathbf{K}_g = [\mathbf{w}_g, \mathbf{y}_g] \in R_q^{\ell\times 2}, \mathbf{K}_h = [\mathbf{w}_h, \mathbf{y}_h] \in R_q^{\ell\times 2}$)
    $\hat{\mathbf{a}}_1 \leftarrow \hat{\mathbf{a}}_{agg}[0 : d/2]$
    $\hat{\mathbf{a}}_2 \leftarrow \hat{\mathbf{a}}_{agg}[d/2 : d]$
    $(a_1, b_1) \leftarrow$ CollapseHalf($(\hat{\mathbf{a}}_1, \tilde{b}_{agg}, [\mathbf{w}_g, \mathbf{y}_g], I$))
    $(a_2, b_2) \leftarrow$ CollapseHalf($(\hat{\mathbf{a}}_2, b_1, [\mathbf{w}_g, \mathbf{y}_g], \tau_h$))
    $(a, b) \leftarrow$ CollapseOne($([a_1, a_2], b_2), [\mathbf{w}_h, \mathbf{y}_h]$)
    return $(a, b) \in R_q \times R_q$
procedure CollapseHalf($(\hat{\mathbf{a}}_{half}, \tilde{b}_{half}) \in R_q^{d/2} \times R_q, \mathbf{K}_g = [\mathbf{w}_g, \mathbf{y}_g] \in R_q^{\ell\times 2}, \rho \in \{I, \tau_h\}$)
    Rename $(\hat{\mathbf{a}}_{half}, \tilde{b}_{half})$ as $(\mathbf{a}^{(d/2-1)}, b^{(d/2-1)})$
    for $k = d/2-1 \dots 0$ do
        $\mathbf{K}_g^{k-1} \leftarrow \rho \circ \tau_g^{k-1}(\mathbf{K}_g)$
        $(\mathbf{a}^{(k-1)}, b^{(k-1)}) \leftarrow$ CollapseOne($(\mathbf{a}^{(k)}, b^{(k)}), \mathbf{K}_g^{k-1}$)
    return $(\mathbf{a}^{(0)}, b^{(0)}) \in R_q \times R_q$
procedure CollapseOne($(\mathbf{a}, b) \in R_q^k \times R_q, \mathbf{K} = [\mathbf{w}, \mathbf{y}] \in R_q^{\ell\times 2}$)
    $(a', b') \leftarrow$ KS.Switch($(\mathbf{a}[k-1], b), [\mathbf{w}, \mathbf{y}]$)
    $\mathbf{a}' \leftarrow [\mathbf{a}[0], \dots, \mathbf{a}[k-3], \mathbf{a}[k-2] + a']^\top \in R_q^{k-1}$
    return $(\mathbf{a}', b') \in R_q^{k-1} \times R_q$
```

actual computation, and their aggregation into IRCtx$(\hat{m}_{agg})$ is a straightforward process of assigning the constant $\tilde{b}_k$ values as polynomial coefficients.

The advantages extend to Stage 3, though perhaps less immediately obvious. This stage involves an iterative reduction of the intermediate ciphertext's components using key-switchings. After Stage 2, the IRCtx$(\hat{m}_{agg})$ consists of d random components (which, as established, can be preprocessed offline) and a single pseudorandom polynomial component $\tilde{b}_{agg}$ (which must be processed online). The iterative key-switching procedure is designed to maintain an important invariant: throughout the reduction, the evolving random components remain dependent only on the initial, preprocessable random components of the input LWE ciphertexts and the key-switching matrices $\mathbf{K}_g$ and $\mathbf{K}_h$. This crucial property ensures that most data necessary to update the pseudorandom component during each step of the reduction can also be precomputed. Consequently, the online operations in Stage 3 are significantly efficient, contributing to the overall efficiency of the online packing algorithm.

Theorem 1. InspiRING *in the CRS model can pack d LWE ciphertexts in $O(d^3 + \ell d^2 \lg d)$ offline time and $O(\ell \cdot d^2)$ online time where ℓ is the dimension of the key-switching matrix.*

Furthermore, our algorithm also results in smaller noise growth (both asymptotically and practically).

Theorem 2. *Let the error distribution χ be subgaussian with parameter σ_χ. Let ℓ be the dimension of the key-switching matrix and z be the decomposition base. Under the independence heuristic,* InspiRING *incurs an additive noise $e_{pack} \in R_q$, which has subgaussian coefficients with parameter σ_{pack} and $\sigma_{pack}^2 \leq \ell d^2 z^2 \sigma_\chi^2/4$.*

Our packing scheme has the same asymptotic running time, but much faster concrete running time as well as smaller noise growth compared to CDKS [18]. We corroborate our analysis with experimental evaluation in Appendix 7.4.

Security. Beyond RLWE hardness, our packing scheme relies on the standard *circular security* assumption, as key-switching matrices encrypt (scaled) automorphic images of the secret key. This assumption is common in lattice based FHE literature [38], [14], [13], [12], [18], [45], [7] and PIR [67], [62], [16], [63], [53], [4].

3.3. Partial Packing with One Key-switching Matrix

InspiRING shows how to pack d LWE ciphertexts into one RLWE ciphertext. This algorithm can be simplified to pack $\gamma \leq d/2$ LWE ciphertexts as well. There are two advantages to partial packing compared to full packing. First, we show how to achieve this using only one key switching matrix instead of two, which reduces the required key material by half. The second advantage is that since only γ LWEs are packed within each RLWE ciphertext, we keep the first γ coefficients of b_{fin} and discard the remaining coefficients. This is useful as means to reduce communication when transmitting ciphertexts over the network and is also used in one of our PIR constructions. Algorithm 2 shows the algorithm for partial packing which we denote as PartialInspiRING.

Algorithm 2 PartialInspiRING algorithm for packing γ LWE ciphertexts

```
procedure PartialPack($[\mathbf{A}, \mathbf{b}] \in \mathbb{Z}_q^{\gamma \times (d+1)}, \mathbf{K}_g = [\mathbf{w}_g, \mathbf{y}_g] \in R_q^{\ell \times 2}$)
    for $k = 0 \ldots \gamma - 1$ do
        $(\hat{\mathbf{a}}_k, \tilde{b}_k) \leftarrow$ TransformPartial($\mathbf{A}[k], \mathbf{b}[k]$)
    $(\hat{\mathbf{a}}_{\mathbf{agg}}, \tilde{b}_{agg}) \leftarrow \sum_{k=0}^{\gamma-1} (\hat{\mathbf{a}}_k, \tilde{b}_k) \cdot X^k$
    $(a_{fin}, b_{fin}) \leftarrow$ CollapsePartial($\gamma, (\hat{\mathbf{a}}_{agg}, \tilde{b}_{agg}), [\mathbf{w}_g, \mathbf{y}_g]$)
    return $(a_{fin}, b_{fin}) \in R_q \times R_q$

— Subroutines —

procedure TransformPartial($\gamma$, $(\mathbf{a}, b) \in \mathbb{Z}_q^d \times \mathbb{Z}_q$)
    $\tilde{a} \leftarrow \gamma^{-1} \sum_{i=0}^{d-1} \mathbf{a}[i] X^{-i}$                ▷ $\gamma \cdot \gamma^{-1} = 1 \mod q$
    $\tilde{b} \leftarrow b$
    Initialize $\hat{\mathbf{a}} \in R_q^\gamma$
    for $j = 0 \ldots \gamma - 1$ do
        $\hat{\mathbf{a}}[j] \leftarrow \tau_g^j(\tilde{a})$
    return $(\hat{\mathbf{a}}, \tilde{b}) \in R_q^\gamma \times R_q$
procedure CollapsePartial($\gamma$, $(\hat{\mathbf{a}}_{partial}, \tilde{b}_{partial}) \in R_q^\gamma \times R_q, \mathbf{K}_g = [\mathbf{w}_g, \mathbf{y}_g] \in R_q^{\ell \times 2}$)
    Rename $(\hat{\mathbf{a}}_{partial}, \tilde{b}_{partial})$ as $(\mathbf{a}^{(\gamma-1)}, b^{(\gamma-1)})$
    for $k = \gamma - 1 \ldots 0$ do
        $\mathbf{K}_g^{k-1} \leftarrow \tau_g^{k-1}(\mathbf{K}_g)$
        $(\mathbf{a}^{(k-1)}, b^{(k-1)}) \leftarrow$ CollapseOne($(\mathbf{a}^{(k)}, b^{(k)}), \mathbf{K}_g^{k-1}$)
    return $(\mathbf{a}^{(0)}, b^{(0)}) \in R_q \times R_q$
```

Theorem 3. PartialInspiRING *in the CRS model can pack $\gamma \leq d/2$ LWE ciphertexts in $O(\gamma^2 d + \ell\gamma d \lg d)$ offline time and $O(\ell\gamma d)$ online time where ℓ is the dimension of the key-switching matrix.*

Theorem 4. *Let the error distribution χ be subgaussian with parameter σ_χ. Let ℓ be the dimension of the key-switching matrix and z be the decomposition base. Under the independence heuristic,* PartialInspiRING *incurs an additive noise $e_{pack} \in R_q$, such that the first γ coefficients of e_{pack} are subgaussian with parameter σ_{pack} and $\sigma_{pack}^2 \leq \ell\gamma d z^2 \sigma_\chi^2 / 4$.*

4. InsPIRe$_0$: PIR from Ring Packing

A direct application of InspiRING to PIR results in our first PIR construction InsPIRe$_0$. InsPIRe$_0$ is instantiated on top of DoublePIR by using InspiRING or PartialInspiRING to pack the result of the DoublePIR responses. This protocol is useful when the entry size is small (e.g. 1 bit) and when optimizing for specific metrics such as runtime, as we show in the evaluation.

5. InsPIRe$^{(2)}$: PIR from Double Ring Packing

Our first construction based on our new ring packing algorithm is denoted as InsPIRe$^{(2)}$. This protocol takes advantage of two layers of ring packing, specifically partial ring packing. The algorithm is designed such that most of the work is done in an offline phase. At a high level, InsPIRe$^{(2)}$ consists of two levels of packing as follows:

- We perform a first level of PIR using LWE and perform partial packing on the resulting LWE ciphertexts using γ_0 as the packing parameter. The packed RLWE ciphertexts are modulus switched and decomposed into plaintexts, which are used as the database in the second layer of PIR.
- In the second level, we start again with PIR using LWE ciphertexts and pack the resulting LWE ciphertexts again. However, the packing in this level is done in two parts, using two different packing parameters, γ_1 and γ_2.

We present our detailed PIR protocol $\mathsf{InsPIRe}^{(2)} = (\mathsf{Setup}, \mathsf{Query}, \mathsf{Respond}, \mathsf{Extract})$ using PartialInspiRING in Algorithms 3, 4, 5, 6. The database consists of N records, each of which is an element of $\mathbb{Z}_p^{\gamma_0}$. The database is, however, restructured into a matrix $\mathbf{D} \in \mathbb{Z}_p^{t\gamma_0 \times N/t}$ For convenience, we assume that the public parameters pp are globally available to the subroutines of the PIR algorithms and do not explicitly specify them as inputs.

Algorithm 3 $\mathsf{InsPIRe}^{(2)}.\mathsf{Setup}$

```
procedure SETUP($1^\lambda, \mathbf{D} \in \mathbb{Z}_p^{t\gamma_0 \times N/t}$)
    $\mathsf{rp} \leftarrow (d(\lambda), q(\lambda), \tilde{q}, \chi(\lambda), p)$ as the RLWE params
    $\mathsf{gp}_{ks} \leftarrow (z_{ks}, \ell_{ks})$ as the gadget params for the KS matrices
    $\mathsf{qry}_{\mathsf{off}} \leftarrow$ GENFIXEDQUERYPARTS()
    $\tau \leftarrow \lceil \tilde{q}/p \rceil$
    $\mathsf{pp} \leftarrow (N, t, \gamma_0, \gamma_1, \gamma_2, \mathsf{rp}, \tilde{q}, \tau, \mathsf{gp}_{ks}, \mathsf{qry}_{\mathsf{off}})$
    return $(\mathsf{pp}, \mathbf{D})$

procedure GENFIXEDQUERYPARTS()
    $\mathbf{A_0} \leftarrow U(\mathbb{Z}_q^{N/t \times d})$
    $\mathbf{A_1} \leftarrow U(\mathbb{Z}_q^{t \times d})$
    $\{\mathbf{w}_g^{(\gamma_i)}\}_{i \in [3]} \leftarrow U(R_q^{\ell_{ks}})$
    return $\mathsf{qry}_{\mathsf{off}} = (\mathbf{A}, \{\mathbf{w}_g^{(\gamma_i)}\}_{i \in [3]})$
```

Algorithm 4 $\mathsf{InsPIRe}^{(2)}.\mathsf{Query}$

```
procedure QUERY(pp, idx)
    Parse pp as $(N, t, \gamma_0, \gamma_1, \gamma_2, \mathsf{rp}, \tilde{q}, \tau, \mathsf{gp}_{ks}, \mathsf{qry}_{\mathsf{off}})$
    Parse $\mathsf{qry}_{\mathsf{off}}$ as $(\mathbf{A_0}, \mathbf{A_1}, \{\mathbf{w}_g^{(\gamma_i)}\}_{i \in [3]})$
    Parse idx as $(i_0, i_1)$
    $\mathbf{s} \leftarrow \chi(\mathbb{Z}^d)$
    $\tilde{s} \leftarrow \sum_{k=0}^{d-1} \mathbf{s}[k] \cdot X^k$
    $\mathbf{b_0} \leftarrow$ GENQUERY($\mathbf{A_0}, \mathbf{s}, N/t, i_0$)
    $\mathbf{b_1} \leftarrow$ GENQUERY($\mathbf{A_1}, \mathbf{s}, t, i_1$)
    for $i \in \{0, 1, 2\}$ do
        $\mathbf{y}_g^{(\gamma_i)} \leftarrow$ GENKSMATY($\tilde{s}, \mathbf{w}_g^{(\gamma_i)}, \tau_g$)
    $\mathsf{qry} \leftarrow (\mathbf{b_0}, \mathbf{b_1}, \{\mathbf{y}_g^{(\gamma_i)}\}_{i \in [3]})$
    return $(\mathsf{st} = \tilde{s}, \mathsf{qry})$

procedure GENQUERY($\mathbf{A}, \mathbf{s}, n, i$)
    Initialize $\mathbf{b} \in \mathbb{Z}_q^n$
    for $k = [n]$ do
        $m_k \leftarrow k = i$
        $e_k \leftarrow \chi(\mathbb{Z})$
        $\mathbf{b}[k] \leftarrow -\langle \mathbf{A}[k], \mathbf{s} \rangle + e_k + \Delta m_k$        ▷ $\Delta = \lfloor q/p \rceil$
    return $\mathbf{b}$

procedure GENKSMATY($\tilde{s}, \mathbf{w}, \rho \in \{\tau_g\}$)
    $\mathbf{e} \leftarrow \chi(R_q^{\ell_{ks}})$
    $\mathbf{y} \leftarrow \tilde{s}\mathbf{w} + \mathbf{e} + \rho(\tilde{s}) \cdot \mathbf{g}_{z_{ks}}$
    return $\mathbf{y}$
```

Theorem 5. *Let ℓ_{ks} be the gadget parameter of the key-switching matrices. In the CRS model,* $\mathsf{InsPIRe}^{(2)}$ *runs in offline time*

$$O(N\gamma_0 d + t(\gamma_0^2 d + \gamma_0 \ell_{ks} d \lg d) + \tau t d^2 + \tau d(\gamma_1 d + \ell_{ks} d \lg d))$$

and online time

$$O(N\gamma_0 + t\gamma_0 d\ell_{ks} + t\tau\gamma_0 + \tau d^2 \ell_{ks} + \tau\gamma_0 d(\gamma_2 + \ell_{ks} \lg d)).$$

Theorem 6. *Let $\alpha = |\{\gamma_0, \gamma_1, \gamma_2\}|$, which denotes the number of distinct values for the packing parameter. The total communication cost of* $\mathsf{InsPIRe}^{(2)}$ *is equal to*

$$\alpha \cdot d\ell_{ks} \log_2 q + (N/t) \log_2 q + t \log_2 q + (\tau d/\gamma_1)(d + \gamma_1) \log_2 \tilde{q} + (\tau\gamma_0/\gamma_2)(d + \gamma_2) \log_2 \tilde{q}$$

Proof. The total communication consists of three components, the packing keys, the query, and the response. We require one packing key for each distinct value of γ_i, each of which is size $d\ell_{ks} \log_2 q$. The two indicator vector of length N/t and

Algorithm 5 $\mathsf{InsPIRe}^{(2)}.\mathsf{Respond}$

```
procedure RESPOND(pp, $\mathbf{D} \in \mathbb{Z}_p^{t\gamma_0 \times N/t}$, qry)
    Parse pp as $(N, \gamma_0, \gamma_1, \gamma_2, \mathsf{rp}, \tilde{q}, \tau, \mathsf{gp}_{ks}, \mathsf{gp}_{gsw}, \mathsf{qry}_{\mathsf{off}})$
    Parse $\mathsf{qry}_{\mathsf{off}}$ as $(\mathbf{A_0}, \mathbf{A_1}, \{\mathbf{w}_g^{(\gamma_i)}\}_{i\in[3]})$
    Parse qry as $(\mathbf{b_0}, \mathbf{b_1}, \{\mathbf{y}_g^{(\gamma_i)}\}_{i\in[3]}))$
    for $i \in \{0,1,2\}$ do
        $\mathbf{K}_g^{(\gamma_i)} \leftarrow [\mathbf{w}_g^{(\gamma_i)}, \mathbf{y}_g^{(\gamma_i)}]$
    $[\mathbf{H_0}, \mathbf{b}'_\mathbf{0}] \leftarrow \mathbf{D} \cdot [\mathbf{A_0}, \mathbf{b_0}]$
    $\mathsf{resp}_0 \leftarrow$ InspiRING.PARTIALPACKBATCH$(t, \gamma_0, [\mathbf{H_0}, \mathbf{b}'_\mathbf{0}], \mathbf{K}_g^{(\gamma_0)})$        ▷ $\mathsf{resp}_0 \in \mathbb{Z}_{\tilde{q}}^{\,t\times(d+\gamma_0)}$
    $\mathbf{D_1} = [\,\overline{\mathbf{D}}_\mathbf{1} \mid \underline{\mathbf{D}}_\mathbf{1}\,] \leftarrow$ DECOMPOSE$(\mathsf{resp}_0, \tilde{q}, p)$        ▷ $\mathbf{D_1} \in \mathbb{Z}_p^{\,t\times\tau(d+\gamma_0)}$
    $\mathbf{H_1} = \begin{bmatrix} \overline{\mathbf{H}}_\mathbf{1} & |\overline{\mathbf{b}}'_\mathbf{1} \\ \underline{\mathbf{H}}_\mathbf{1} & |\underline{\mathbf{b}}'_\mathbf{1} \end{bmatrix} \leftarrow [\,\overline{\mathbf{D}}_\mathbf{1} \mid \underline{\mathbf{D}}_\mathbf{1}\,]^T \cdot [\,\mathbf{A_1} | \mathbf{b_1}]$        ▷ $\mathbf{H_1} \in \mathbb{Z}_q^{\tau(d+\gamma_0)\times(d+1)}$
    $\mathsf{resp}_1 \leftarrow$ InspiRING.PARTIALPACKBATCH$(\left\lceil \frac{\tau d}{\gamma_1} \right\rceil, \gamma_1, [\overline{\mathbf{H}}_\mathbf{1}, \overline{\mathbf{b}}'_\mathbf{1}], \mathbf{K}_g^{(\gamma_1)})$
    $\mathsf{resp}_2 \leftarrow$ InspiRING.PARTIALPACKBATCH$(\left\lceil \frac{\tau\gamma_0}{\gamma_2} \right\rceil, \gamma_2, [\underline{\mathbf{H}}_\mathbf{1}, \underline{\mathbf{b}}'_\mathbf{1}], \mathbf{K}_g^{(\gamma_2)})$
    return resp $= \{\mathsf{resp}_1, \mathsf{resp}_2\}$

procedure INSPIRING.PARTIALPACKBATCH$(\ell, \gamma, [\mathbf{H}, \mathbf{b}'], \mathbf{K}_g)$
    for $k \in [\ell]$ do
        $\mathbf{H_k} \leftarrow \mathbf{H}[k\gamma : k\gamma + \gamma][:]$
        $\mathbf{b}'_\mathbf{k} \leftarrow \mathbf{b}'[k\gamma : k\gamma + \gamma]$
        $(a_k, b_k) \leftarrow$ InspiRING.PARTIALPACK$(\gamma, [\mathbf{H_k}, \mathbf{b}'_\mathbf{k}], \mathbf{K}_g)$
        $(\tilde{a}_k, \tilde{b}_k) \leftarrow (\lfloor \tilde{q} a_k / q \rceil, \lfloor \tilde{q} b_k / q \rceil)$
    return $\{(\tilde{a}_k[:], \tilde{b}_k[: \gamma])\}_{k\in[\ell]}$

procedure DECOMPOSE$(\mathbf{C}, \tilde{q}, p, \tau)$
    for $i \in [\tau]$ do
        $\mathbf{C}_i = \lfloor p^i \mathbf{C} / \tilde{q} \rceil \mod p$
    $\mathbf{D} = [\mathbf{C}_0 | \mathbf{C}_1 | \cdots | \mathbf{C}_{\tau-1}]$
    return $\mathbf{D}$
```

Algorithm 6 $\mathsf{InsPIRe}^{(2)}.\mathsf{Extract}$

```
procedure EXTRACT(pp, st, resp)
    Extract $\tilde{s}$ from st
    Parse resp as $\{\mathsf{resp}_1, \mathsf{resp}_2\}$
    $a \leftarrow$ BATCHPARTIALDECRYPT$(\tilde{s}, \mathsf{resp}_1, \gamma_1)$        ▷ $a \in \mathbb{Z}_q^{1\times\tau d}$
    $b \leftarrow$ BATCHPARTIALDECRYPT$(\tilde{s}, \mathsf{resp}_2, \gamma_2)$        ▷ $b \in \mathbb{Z}_q^{1\times\tau\gamma_0}$
    ct $\leftarrow$ COMPOSE$([a|b], \tilde{q}, p, \tau)$
    res $\leftarrow$ BATCHPARTIALDECRYPT$(\tilde{s}, \mathsf{ct}, \gamma_0)$
    return res

procedure COMPOSE$(\mathbf{D}, \tilde{q}, p, \tau)$
    Parse $\mathbf{D}$ as $[\mathbf{C}_0 | \mathbf{C}_1 | \cdots | \mathbf{C}_{\tau-1}]$
    $\mathbf{C} = \sum_{i\in[\tau]} p^{\tau-1-i} \mathbf{C}_i$
    return $\mathbf{C}$

procedure BATCHPARTIALDECRYPT$(\tilde{s}, \mathsf{resp}, \gamma)$
    Parse resp as $\{ct_i\}_{i\in[k]}$
    for $i \in [k]$ do
        $p_i \leftarrow$ RLWE.Dec$(\tilde{s}, ct_i)$
        $pt_i \leftarrow p_i[: \gamma]$
    return $pt = [pt_0 | pt_1 | \cdots | pt_{k-1}]$        ▷ $pt \in \mathbb{Z}_q^{1\times k\gamma}$
```

t, for the first and second layer, respectively, which amounts to size $(N/t)\log_2 q + t \log_2 q$. Lastly, the response consists of resp_1 and resp_2, which are of size $(\tau d/\gamma_1)(d+\gamma_1)\log_2 \tilde{q}$ and $(\tau\gamma_0/\gamma_2)(d+\gamma_2)\log_2 \tilde{q}$, respectively. □

Theorem 7. *Let the error distribution χ be subgaussian with parameter σ_χ. For $i \in \{0,1,2\}$, let e_i denote the error term of an RLWE ciphertext in $\mathsf{resp_i}$, defined as in Algorithm 5. Moreover, let $e[:\gamma]$ denote a polynomial which only consists of the first γ coefficients of e. Then for $i \in \{0,1,2\}$, under the independence heuristic, $\tilde{e}_i[:\gamma_i] = \tilde{e}_i^{(1)} + \tilde{e}_i^{(2)}$ such that*

$$||e_i^{(1)}||_\infty \leq f(q, \tilde{q}, p) = \frac{1}{2}(2 + (\tilde{q} \mod p) + \frac{\tilde{q}}{q}(q \mod p))$$

and $\tilde{e}_i^{(2)}$ has subgaussian coefficients with parameter $\tilde{\sigma}_i$ where

$$\tilde{\sigma}_0^2 \leq \Sigma_0 = d\sigma_\chi^2/4 + (\tilde{q}/q)^2((N/t)p^2\sigma_\chi^2 + \ell_{ks}\gamma_0 d z_{ks}^2 \sigma_\chi^2/4) \quad (3)$$

$$\tilde{\sigma}_1^2 \leq \Sigma_1 = d\sigma_\chi^2/4 + (\tilde{q}/q)^2(tp^2\sigma_\chi^2 + \ell_{ks}\gamma_1 d z_{ks}^2 \sigma_\chi^2/4) \quad (4)$$

$$\tilde{\sigma}_2^2 \leq \Sigma_2 = d\sigma_\chi^2/4 + (\tilde{q}/q)^2(tp^2\sigma_\chi^2 + \ell_{ks}\gamma_2 d z_{ks}^2 \sigma_\chi^2/4) \quad (5)$$

Theorem 8. *For $f(q, \tilde{q}, p)$ and Σ_i defined as in Theorem 7, then if*

$$\delta_0 \geq 2t\gamma_0 \exp\left(-\pi(\tilde{q}/2p - f(q, \tilde{q}, p))^2/\Sigma_0^2\right)$$
$$\delta_1 \geq 2\tau d \exp\left(-\pi(\tilde{q}/2p - f(q, \tilde{q}, p))^2/\Sigma_1^2\right)$$
$$\delta_2 \geq 2\tau\gamma_0 \exp\left(-\pi(\tilde{q}/2p - f(q, \tilde{q}, p))^2/\Sigma_2^2\right)$$

then $\mathsf{InsPIRe}^{(2)}$ *is $(1 - \delta_0 - \delta_1 - \delta_2)$-correct.*

6. InsPIRe: PIR from Ring Packing and Homomorphic Polynomial Evaluation

In this section, we present InsPIRe, our low query communication and high-throughput PIR protocol that uses our InspiRING ring packing with preprocessing algorithm as a building block. One can follow the prior PIR frameworks such as YPIR [63] and replace their ring packing algorithms with InspiRING. This immediately results in an improved PIR scheme. However, we will present a new PIR protocol that both leverages InspiRING as well as polynomial evaluation to obtain further communication improvements.

We start by presenting an overview of the prior framework using ring packing and highlight some limitations. Initially, suppose the database consists of N entries, each as an element in $\mathbb{Z}_p^d$. Note, the choice of using d here is for simplicity and we will generalize later. We introduce an additional configurable parameter t and model the plaintext database as a matrix $\mathbf{D} \in \mathbb{Z}_p^{td \times N/t}$. Each column of the matrix corresponds to t database entries of $\mathbb{Z}_p^d$.

During the query phase, the client queries for the column i that contains the desired entry. To achieve this, the client generates LWE encryptions of a one-hot selection vector of length N/t. All ciphertexts encrypt zero except the i-th ciphertext which encrypts one. The N/t LWE ciphertexts can be represented as the rows of the matrix $[\mathbf{A}, \mathbf{b}] \in \mathbb{Z}_q^{N/t \times (d+1)}$ which are sent to the server. Next, the server performs matrix-matrix multiplication to obtain $\mathbf{D} \cdot [\mathbf{A}, \mathbf{b}]$, which yields td LWE encryptions of the i-th column of $\mathbf{D}$. At this point, each consecutive chunk of d LWE ciphertexts corresponds to a database entry over $\mathbb{Z}_p^d$. Finally, the server uses ring packing (such as InspiRING) to translate each chunk of d LWE ciphertexts into a single RLWE ciphertext before returning the resulting t RLWE ciphertexts back to the client.

In this scheme, the server returns encryptions corresponding to the entire column of t entries. To reduce the response size, we ideally want the server to only return the desired entry instead. Previous approaches, such as OnionPIR [67] and Spiral [62], employed encryptions of one-hot selection vectors for this step. However, these methods typically require $\Theta(\lg(t))$ ciphertexts or large evaluation keys, adding significant communication.

6.1. Homomorphic Polynomial Evaluation

We present a new PIR protocol that introduces a new way of encoding the plaintext database. Instead of explicitly representing each column as a concatenation of t database entries, our key idea is to implicitly represent it as coefficients of a polynomial that *evaluates* to the entries in that column for some publicly fixed evaluation points.

With this new database encoding, the initial homomorphic column selection will obtain encrypted polynomial coefficients. Afterwards, we use homomorphic polynomial evaluation to extract the desired entry. By carefully constructing the polynomial, we can perform the evaluation using a *single* RGSW ciphertext. Furthermore, our design utilizes RLWE-RGSW external products (see Section 2) to minimize noise growth during homomorphic evaluation. By choosing the evaluation points appropriately, our homomorphic evaluation will only incur *additive* noise growth for each multiplication.

Polynomials for Database Encoding. We proceed more formally to define the polynomial representations for each database column. For convenience, we will assume that N, and t are powers of two, but we will generalize later. Recall that our plaintext database is modeled as $\mathbf{D} \in \mathbb{Z}_p^{td \times N/t}$ where each column contains t entries from $\mathbb{Z}_p^d$. We denote $\mathbf{y}_j^{(i)} \in \mathbb{Z}_p^d$ as the j-th entry in the i-th column.

We will use polynomials to represent each column of the matrix $\mathbf{D}$. We first begin by re-interpreting each $\mathbf{y}_j^{(i)} \in \mathbb{Z}_p^d$ as an element in $R_p = \mathbb{Z}_p[X]/(X^d + 1)$ by mapping the d elements in $\mathbb{Z}_p$ as the polynomial coefficients. We will denote these re-interpretations as $y_j^{(i)} \in R_p$.

Let $R_p[Z]$ be the polynomial ring over R_p. Imagine that we have t distinct points $z_0, \ldots, z_{t-1} \in R_p$ that are fixed publicly beforehand. We will pick these t points later. For each column i, suppose there exists a polynomial $h^{(i)}(Z) = \sum_{j=0}^{t-1} c_j^{(i)} Z^j \in R_p[Z]$ such that the following holds:

$$h^{(i)}(z_j) = y_j^{(i)}, \forall j \in \{0, 1, \ldots, t-1\}.$$

Suppose the server can compute the coefficients $c_0^{(i)}, \ldots, c_{t-1}^{(i)} \in R_p$ of the polynomial $h^{(i)}(Z)$ for each column i. We construct encoded database $\mathbf{D}' \in \mathbb{Z}_p^{td \times N/t}$, where each column i consists of the corresponding polynomial coefficients $c_j^{(i)}$ interpreted as elements in $\mathbb{Z}_p^d$.

Homomorphic Polynomial Evaluation. Suppose the client wants to retrieve the k-th entry from the i-th column, $y_k^{(i)}$. Consider the point after the server has performed initial homomorphic selection of column i *on the encoded database* $\mathbf{D}'$. At this point, the server holds t RLWE encryptions of the coefficients $c_0^{(i)}, \ldots, c_{t-1}^{(i)}$ of the polynomial $h^{(i)}(Z)$. Since the evaluation points z_j are publicly fixed and the polynomials $h^{(i)}(Z)$ are constructed such that $h^{(i)}(z_j) = y_j^{(i)}$ for each $0 \leq j < t$, the client can send a *single* RGSW ciphertext encrypting the evaluation point z_k to the server. The server can then homomorphically evaluate the polynomial $h^{(i)}(Z)$ using the RLWE encrypted coefficients $c_j^{(i)}$ and a single RGSW encrypted evaluation point z_k. This would obtain the desired entry $h^{(i)}(z_k) = y_k^{(i)}$. To perform evaluation, we will use the Horner-style method (see EVALPOLY in Algorithm 9), that only involves RLWE-RGSW external products and RLWE additions.

So far, we have not discussed how the evaluation points z_j are chosen. For our PIR protocol to be efficient, we want the evaluation points z_j to satisfy the following properties:

1) *Noise Management during Evaluation:* Homomorphic polynomial evaluation involves large-depth multiplications (e.g. in Horner's method), each increasing the noise in the resulting ciphertext. We want to strategically choose the evaluation points so that the noise growth from multiplications is minimal (ideally, additive).
2) *Interpolation in Rings:* The evaluation points must guarantee a polynomial interpolation within the ring R_p. Since R_p is not necessarily a field, the standard conditions for interpolation (requiring distinct points) are not sufficient. We need to ensure the existence of the interpolating polynomial for any set of database entries.

Unit Monomials as Evaluation Points. To address the challenges of noise management and interpolation, we propose a specific choice for the evaluation points. Our key insight is to select evaluation points $z_0, \ldots, z_{t-1}$ that are both *unit monomials* and *roots of unity*. Here, we will define unit monomials as elements in R_p of the form $\pm X^k$ where $0 \leq k < d$. If the RGSW ciphertext encrypting the evaluation point encrypts a unit monomial, the external product operation incurs only *additive noise* growth, rather than multiplicative.

Lemma 2. *[Theorem 2.19 [62], adapted] Given* $\mathsf{RLWE}(m_0)$, *let* $\mathsf{RGSW}(m_1)$ *be a fresh ciphertext encrypting a unit monomial, e.g.* $m_1 = \pm X^k$ *for some* $0 \leq k < d$. *Let the error distribution* χ *be subgaussian with parameter* σ_χ, *and let* ℓ *and* z *be the gadget parameters of the RGSW ciphertext. Under the independence heuristic, the external product* $\mathsf{RLWE}(m_0) \boxdot \mathsf{RGSW}(m_1)$ *incurs additive noise* $e_{ep} \in R_q$ *whose coefficients are subgaussian with parameter* σ_{ep} *and* $\sigma_{ep}^2 \leq \ell d z^2 \sigma_\chi^2 / 2$.

By ensuring that all $z_0, \ldots, z_{t-1}$ are unit monomials, the entire homomorphic evaluation process (involving $t-1$ multiplications and additions) incurs only additive noise.

Next, we discuss how to guarantee polynomial interpolation within R_p using these unit monomial evaluation points. To guarantee interpolation, our key insight is to use powers of a *primitive t-th root of unity* ω as the evaluation points. Specifically, for $t \leq 2d$ a power of two, we will choose a primitive t-th root $\omega = X^{2d/t}$. With this choice, we define the evaluation points as powers of ω: $z_k := \omega^k$ for $k = 0 \ldots t-1$. Crucially, we see that all the evaluation points z_k are unit monomials. These points are distinct and the Vandermonde matrix (e.g. DFT matrix) formed by them is invertible over R_p, assuming p is odd [15]. Essentially, the polynomial interpolation then amounts to a matrix-vector product between the corresponding IDFT matrix and the vector of $y_j^{(i)}$s, which can be done efficiently using Cooley-Tukey style FFT algorithm.

Constraints and Generalizations. This specific construction imposes two main constraints:

- $t \leq 2d$: This is required for the evaluation points to be unit monomials. While our experimental evaluations indicate that $t \ll 2d$ often suffices for effective PIR, scenarios requiring larger folding factor ($t > 2d$) are possible. We discuss extensions using multivariate polynomial interpolation in Appendix G.
- t **is a power of two:** This is required for the Cooley-Tukey style FFT algorithm used for the interpolation during database encoding. If t is not a power of two, a computationally less efficient Lagrange interpolation can be used instead. We present this Lagrange interpolation based method in Appendix G.

Algorithm 7 InsPIRe.Setup

```
procedure SETUP($1^\lambda, \mathbf{D} \in \mathbb{Z}_p^{td \times N/t}$)
    $\mathsf{rp} \leftarrow (d(\lambda), q(\lambda), \chi(\lambda), p)$ as the RLWE params
    $\mathsf{gp}_{ks} \leftarrow (z_{ks}, \ell_{ks})$ as the gadget params for the KS matrices
    $\mathsf{gp}_{gsw} \leftarrow (z_{gsw}, \ell_{gsw})$ as the gadget params for the RGSW ciphertext
    $\mathsf{qry}_{\mathsf{off}} \leftarrow$ GENFIXEDQUERYPARTS()
    $\mathsf{pp} \leftarrow (N, t, \mathsf{rp}, \mathsf{gp}_{ks}, \mathsf{gp}_{gsw}, \mathsf{qry}_{\mathsf{off}})$
    $\mathbf{D}' \leftarrow$ ENCODEDB($\mathbf{D}$)
    return $(\mathsf{pp}, \mathbf{D}')$

procedure ENCODEDB($\mathbf{D} \in \mathbb{Z}_p^{td \times N/t}$)
    Initialize $\mathbf{D}' \in \mathbb{Z}_p^{td \times N/t}$
    for $i = 0 \dots N/t - 1$ do
        $y_j^{(i)} \leftarrow j$-th element in column $i$ interpreted as $R_p$ for $0 \leq j < t$
        $[c_0^{(i)}, \dots, c_{t-1}^{(i)}]^\top \leftarrow$ INTERPOLATE($[y_0^{(i)}, \dots, y_{t-1}^{(i)}]^\top$)
        Cast $[c_0^{(i)}, \dots, c_{t-1}^{(i)}]^\top$ as a vector over $\mathbb{Z}_p^{td}$, and embed into $\mathbf{D}'[:][i]$
    return $\mathbf{D}'$

procedure INTERPOLATE($[y_0, \dots, y_{t-1}]^\top \in R_p^t$)
    $\mathbf{c} \leftarrow$ COOLEYTUKEY($[y_0, \dots, y_{t-1}]^\top$)
    return $t^{-1} \cdot \mathbf{c}$

procedure COOLEYTUKEY($[y_0, \dots, y_{m-1}]^\top \in R_p^m$)
    if $m = 1$ then
        return $[y_0]^\top$
    $\mathbf{c}_{even} \leftarrow$ COOLEYTUKEY($[y_0, y_2, \dots, y_{m-2}]^\top$)
    $\mathbf{c}_{odd} \leftarrow$ COOLEYTUKEY($[y_1, y_3, \dots, y_{m-1}]^\top$)
    Initialize $\mathbf{c} \in R_p^m$
    for $i = 0 \dots m/2 - 1$ do
        $\omega \leftarrow X^{-2d/m \cdot i}$
        $\mathbf{c}[i] \leftarrow \mathbf{c}_{even}[i] + \omega \cdot \mathbf{c}_{odd}[i]$
        $\mathbf{c}[i + m/2] \leftarrow \mathbf{c}_{even}[i] - \omega \cdot \mathbf{c}_{odd}[i]$
    return $\mathbf{c}$

procedure GENFIXEDQUERYPARTS()
    $\mathbf{A} \leftarrow U(\mathbb{Z}_q^{N/t \times d})$
    $\mathbf{w}_g \leftarrow U(R_q^{\ell_{ks}})$
    $\mathbf{w}_h \leftarrow U(R_q^{\ell_{ks}})$
    return $\mathsf{qry}_{\mathsf{off}} = (\mathbf{A}, \mathbf{w}_g, \mathbf{w}_h)$
```

Algorithm 8 InsPIRe.Query

```
procedure QUERY(pp, idx)
    Parse pp as $(N, t, \mathsf{rp}, \mathsf{gp}_{ks}, \mathsf{gp}_{gsw}, \mathsf{qry}_{\mathsf{off}})$
    Parse $\mathsf{qry}_{\mathsf{off}}$ as $(\mathbf{A}, \mathbf{w}_g, \mathbf{w}_h)$
    Parse idx as $(i, j)$
    $\mathbf{s} \leftarrow \chi(\mathbb{Z}^d)$
    $\tilde{s} \leftarrow \sum_{k=0}^{d-1} \mathbf{s}[k] \cdot X^k$
    Initialize $\mathbf{b} \in \mathbb{Z}_q^{N/t}$
    for $k = 0 \dots N/t - 1$ do
        $m_k \leftarrow k = i$
        $e_k \leftarrow \chi(\mathbb{Z})$
        $\mathbf{b}[k] \leftarrow -\langle \mathbf{A}[k], \mathbf{s} \rangle + e_k + \Delta m_k$          ▷ $\Delta = \lfloor q/p \rceil$
    $\omega \leftarrow X^{2d/t}$
    $\mathsf{RGSW}(\omega^j) \leftarrow \mathsf{RGSW.Enc}(\tilde{s}, \omega^j)$
    $\mathbf{y}_g \leftarrow$ GENKSMATY($\tilde{s}, \mathbf{w}_g, \tau_g$)
    $\mathbf{y}_h \leftarrow$ GENKSMATY($\tilde{s}, \mathbf{w}_h, \tau_h$)
    $\mathsf{qry} \leftarrow (\mathbf{b}, \mathsf{RGSW}(\omega^j), \mathbf{y}_g, \mathbf{y}_h)$
    return $(\mathsf{st} = \tilde{s}, \mathsf{qry})$

procedure GENKSMATY($\tilde{s}, \mathbf{w}, \rho \in \{\tau_g, \tau_h\}$)
    $\mathbf{e} \leftarrow \chi(R_q^{\ell_{ks}})$
    $\mathbf{y} \leftarrow \tilde{s}\mathbf{w} + \mathbf{e} + \rho(\tilde{s}) \cdot \mathbf{g}_{z_{ks}}$
    return $\mathbf{y}$
```

6.2. Putting It Together

We present our PIR protocol InsPIRe = (Setup, Query, Respond, Extract) using InspiRING and homomorphic polynomial evaluation in Algorithms 7, 8, 9, 10. For convenience, we assume that the public parameters pp are globally available to the subroutines of the PIR algorithms and do not explicitly specify them as inputs. For instance, we assume InspiRING uses the RLWE parameters rp and the gadget parameters gp_{ks}. Note that GENKSMATY() in Algorithm 8 is essentially the same as KS.Setup except it only generates the pseudorandom component of the key-switching matrix.

Preprocessing in the CRS Model. We remind the readers that the highlighted parts of InsPIRe.Respond correspond to the

Algorithm 9 InsPIRe.Respond

```
procedure Respond(pp, $\mathbf{D}'$ $\in \mathbb{Z}_q^{td \times N/t}$, qry)
    Parse pp as $(N, t, \mathsf{rp}, \mathsf{gp}_{ks}, \mathsf{gp}_{gsw}, \mathsf{qry}_{\mathsf{off}})$
    Parse $\mathsf{qry}_{\mathsf{off}}$ as $(\mathbf{A}, \mathbf{w}_g, \mathbf{w}_h)$
    Parse qry as $(\mathbf{b}, \mathsf{RGSW}(\omega^j), \mathbf{y}_g, \mathbf{y}_h)$
    $\mathbf{K}_g \leftarrow [\ \mathbf{w}_g\ , \mathbf{y}_g]$
    $\mathbf{K}_h \leftarrow [\ \mathbf{w}_h\ , \mathbf{y}_h]$
    $[\ \mathbf{H}\ , \mathbf{b}'] \leftarrow\ \mathbf{D}'\ \cdot [\ \mathbf{A}\ , \mathbf{b}]$
    for $k = 0 \ldots t-1$ do
        $\mathbf{H}_k \leftarrow \mathbf{H}[kd : kd + d)][:]$
        $\mathbf{b}'_k \leftarrow \mathbf{b}'[kd : kd + d]$
        $(\ a_k\ , b_k) \leftarrow$ InspiRING.Pack($[\ \mathbf{H}_k\ , \mathbf{b}'_k], \mathbf{K}_g, \mathbf{K}_h$)
        Rename $(\ a_k\ , b_k)$ to $\mathsf{RLWE}(c_k)$
    resp $\leftarrow$ EvalPoly($\mathsf{RLWE}(c_0), \ldots, \mathsf{RLWE}(c_{t-1}), \mathsf{RGSW}(\omega^j)$)
    return resp
procedure EvalPoly($\mathsf{RLWE}(c_0), \ldots, \mathsf{RLWE}(c_{t-1}), \mathsf{RGSW}(\omega^j)$)
    ct $\leftarrow \mathsf{RLWE}(c_{t-1})$
    for $k = t-2 \ldots 0$ do
        ct $\leftarrow$ ct $\boxdot\ \mathsf{RGSW}(\omega^j) + \mathsf{RLWE}(c_k)$
    return ct
```

Algorithm 10 InsPIRe.Extract

```
procedure Extract(pp, st, resp)
    Extract $\tilde{s}$ from st
    $y_j^{(i)} \leftarrow$ RLWE.Dec($\tilde{s}$, resp)
    return $y_j^{(i)}$
```

preprocessable parts of the algorithm prior to serving queries (see Section 2.2). These correspond to parts that only depend on the outputs from InsPIRe.Setup. We note that this extends to InspiRING.Pack as well (Algorithm 1).

Generalizing to Arbitrary Entry Size. The pseudocode assumes that each entry is encoded as $\mathbb{Z}_p^d$ of length d. To support arbitrary length m, we use standard reductions:

- $m > d$**:** Divide each entry into m/d chunks. Construct m/d databases $\in \mathbb{Z}_q^{td \times N/t}$ where the i-th database contains the i-th chunk. In the Respond algorithm, the server processes the client's query on all databases.
- $m < d$**:** Bundle d/m entries as a single entry over $\mathbb{Z}_p^d$. The client queries for the bundle with the desired entry.

Theorem 9. *Let the error distribution χ be subgaussian with parameter σ_χ. Let ℓ_{ks} and z_{ks} be the gadget parameters of the key-switching matrices. Let ℓ_{gsw} and z_{gsw} be the gadget parameters of the RGSW ciphertext. The error polynomial of the RLWE ciphertext* ct *has the form $e = e_{main} + e_{overflow} \in R_q$. Under the independence heuristics, e_{main} has subgaussian coefficients with parameter $\sigma_{main}^2 \leq Np^2\sigma_\chi^2 + t\ell_{ks}d^2z_{ks}^2\sigma_\chi^2/4 + t\ell_{gsw} \cdot dz_{gsw}^2 \cdot \sigma_\chi^2/2$ and $||e_{overflow}||_\infty \leq tp/2$.*

The $e_{overflow}$ term is introduced from the homomorphic polynomial evaluation because the sum of the embedded plaintexts typically overflows the R_p plaintext space. However, Theorem 9 shows that this additional $e_{overflow}$ is negligible in practice. The full proof is given in Appendix F.

Theorem 10. *For subgaussian χ with parameter σ_χ, define $\bar{\sigma}_{main}^2 = Np^2\sigma_\chi^2 + t\ell_{ks}d^2z_{ks}^2\sigma_\chi^2/4 + t\ell_{gsw} \cdot dz_{gsw}^2 \cdot \sigma_\chi^2/2$. Then, for $\delta > 2d\exp\left(-\pi(\Delta/2 - tp/2)^2/\bar{\sigma}_{main}^2\right)$,* InsPIRe *is $(1-\delta)$-correct.*

Theorem 11. *Let ℓ_{ks} and ℓ_{gsw} be the gadget parameters of the key-switching matrices and the query RGSW ciphertext respectively. In the CRS model,* InsPIRe *runs in offline time $O(Nd^2 + t(d^3 + \ell_{ks}d^2 \lg d))$ and online time $O(Nd + t\ell_{ks}d^2 + t\ell_{gsw}d\lg(d))$.*

Theorem 12. *The total communication cost of* InsPIRe *is*

$$d\ell_{ks}\log_2 q + (N/t)\log_2 q + 4\ell_{gsw}d\log_2 q + 2d\log_2 q$$

Security. The security of our protocol relies on the RLWE hardness assumption and the circular security assumption for the key-switching matrices used in packing (see 3). The full proof is security is given in Appendix F.

Comparison with $\mathsf{InsPIRe}_0$**.** Because $\mathsf{InsPIRe}_0$ is built on top of DoublePIR, it can only support small entry size such as 1 bit. On the other hand, InsPIRe is much more flexible and can support arbitrary entry size with very low communication overhead. As we show in Section 7, InsPIRe can even obtain lower communication than $\mathsf{InsPIRe}_0$, but the downside is the higher server runtime from packing during the initial homomorphic column selection. In contrast, $\mathsf{InsPIRe}_0$ packs the DoublePIR response consisting of a smaller number of LWE ciphertexts. In general, $\mathsf{InsPIRe}$ is the recommended choice unless the entry size is very small and server runtime is the primary metric to optimize.

Further optimizations to $\mathsf{InsPIRe}$ using approximate gadget decomposition is discussed in Appendix G.

7. Experimental Evaluation

In this section, we compare our constructions $\mathsf{InsPIRe}_0$, $\mathsf{InsPIRe}^{(2)}$, and $\mathsf{InsPIRe}$ with existing efficient PIR protocols with server-side preprocessing. In particular, we do not compare with protocols that rely on client-specific cryptographic keys [4], [62], database-dependent hints [43] that need to be sent to the client beforehand, or streaming the database in an offline phase [81]. In our experimental evaluation, we are particularly interested in two metrics: the server's online runtime and the total communication required for a single PIR query. We compare our proposed protocols with state-of-the-art PIR protocols, for small (1 bit), medium (64 B), and large (32 KB) entries.

Implementation & Evaluation Details. We implemented $\mathsf{InspiRING}$ with about 3,000 lines of Rust using building blocks for RLWE operations from spiral-rs [2]. Using $\mathsf{InspiRING}$ and building blocks from the YPIR implementation [3], we implemented $\mathsf{InsPIRe}^{(2)}$ in an additional 3,000 lines of Rust and $\mathsf{InsPIRe}$ in an additional 2,000 lines of Rust. Our code is publicly available on Github [4]. We perform all experimental evaluations on an Intel Xeon CPU @ 2.6 GHz, where we run all experiments in single-threaded mode for fair comparison with related work. We are primarily interested in two metrics, the total communication and the online server runtime. We provide breakdowns of the communication and runtime as well. In all experiments, offline runtimes are measured once and online runtimes are averaged over 5 runs with a standard deviation of less than 5%. Table 1 summarized our parameter choices. For $\mathsf{InsPIRe}_0$, we use parameters identical to those used in YPIR to isolate the benefits of our new protocol. $\mathsf{InsPIRe}_0$ uses two different sets of parameters for the first and second layer, which are listed in the table. For $\mathsf{InsPIRe}^{(2)}$ and $\mathsf{InsPIRe}$, we use lattice parameters which provide 128-bit security based on the lattice-estimator [2] and correctness parameter $\delta = 2^{-40}$.

TABLE 1: Lattice parameters for $\mathsf{InsPIRe}_0$, $\mathsf{InsPIRe}^{(2)}$, and $\mathsf{InsPIRe}$.

	d	$\log_2 q$	σ_χ	p	ℓ_{KS}	ℓ_{GSW}	z_{KS}	z_{GSW}
$\mathsf{InsPIRe}_0$	1024	32	6.4	256	-	-	-	-
	2048	56	6.4	8192	3	-	2^{19}	-
$\mathsf{InsPIRe}^{(2)}$	2048	53	6.4	65536	3	-	2^{19}	-
$\mathsf{InsPIRe}$	2048	56	6.4	65535	3	3	2^{19}	2^{19}

7.1. Parameterizing $\mathsf{InsPIRe}^{(2)}$

In this section, we describe how to choose parameters of $\mathsf{InsPIRe}^{(2)}$, specifically $\{\gamma_i\}_{i\in[3]}$ and t to optimize performance and balance the communication/computation tradeoff. For communication costs, we measure the upload and download costs, and break down the upload costs into the key material (the packing keys) and the query (the encrypted LWE indicator vector and RGSW ciphertext). We also break down the server's online time into five parts: the first matrix multiplication, the first packing, the second matrix multiplication, and the second and third packing.

Note that γ_0 determines the size of the return entry, i.e., an entry of size $\gamma_0 \log_2 p$ bits is returned. In choosing the values of $\{\gamma_i\}_{i\in[3]}$, similar values allows reuse of the key material, hence reducing overall communication. In contrast, different parts of the protocol benefit from different values of γ. For example, the second packing (which uses γ_1) packs τd ciphertexts with offline preprocessing, but the overall runtime of the second packing does not depend on the value of γ_1 (Theorem 5), so it is best to set γ_1 to be largest value possible, i.e., $\gamma_1 = d/2$. In contrast, large values of γ_0 and γ_2 increase the online runtime, so we choose them to be small and set $\gamma_0 = \gamma_2$ to reuse the key material. While other combinations of $\{\gamma_i\}_{i\in[3]}$, we see that no other combination of parameters achieve such small total communication cost, while also maintaining competitive runtime.

The choice of t offers a communication-computation trade-off. The total communication is minimized at $t = \sqrt{N}$, for fixed $\{\gamma_i\}_{i\in[3]}$, based on Theorem 6, but smaller values of t reduce the computation cost, based on Theorem 5. So we vary the value of t as part of our experiments to observe the effect. Figure 2 shows the results for a 1 GB database. Results for other database sizes are given in Appendix I.

[2] https://github.com/menonsamir/spiral-rs/

[3] https://github.com/menonsamir/ypir/

[4] https://github.com/google/private-membership/tree/main/research/InsPIRe

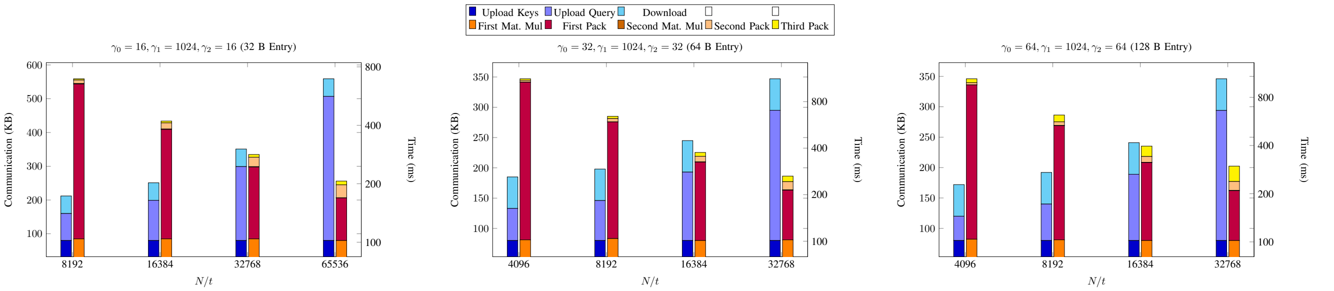

Figure 2: Comparison of communication and computation costs of InsPIRe$^{(2)}$ over a 1 GB database for different parameter sets and entry sizes. From left to right: 32 B, 64 B, and 128 B entry sizes.

Figure 2 shows that as we reduce t (increase N/t), the query size increases, but the query always makes up the majority of the total communication. The smallest total communication for retrieving an entry size of 16B, 32B, and 64B is 211 KB, 185 KB, and 172 KB, respectively. The communication cost can be increased to achieve better runtime. The majority of the runtime is used for the first packing, which is consistent with the analysis of Theorem 5.

7.2. Parameterizing InsPIRe

InsPIRe introduces the interpolation degree (t) as a new parameter of the PIR protocol which can not be chosen in a simple optimization. To guide in choosing this parameters, we first show the effect of the interpolation degree on the performance of InsPIRe. In summary, we show that the choice of the interpolation degree results in a tradeoff between communication and computation.

Figure 3 shows the performance metrics of InsPIRe as we vary the interpolation degree, for a 1 GB database. More results for other database sizes are given in Appendix I. For communication costs, we measure the upload and download costs, and break down the upload costs into the key material (the packing keys) and the query (the encrypted LWE indicator vector and RGSW ciphertext). We also break down the server's online time into three parts: the first matrix multiplication, the packing, and the polynomial evaluation.

For the server online runtime, Figure 3 shows that, for a fixed database size, a larger interpolation degree results in a higher runtime. For large enough interpolation degree, the most costly step in the protocol is the packing, which is compatible with the analysis in Theorem 11.

For the communication costs, the key size and the response size are fixed, regardless of the interpolation degree. The query consists of two parts, the encrypted indicator vector and the RGSW encrypted evaluation point. The size of the RGSW ciphertext is fixed (84 KB), regardless of the interpolation degree. The length of the encrypted indicator vector depends on the first dimension of the database, which inversely depends on the size of each row, which is linear in the interpolation degree. Hence, for larger interpolation degree, the query size reduces, which in turn reduces the overall communication cost.

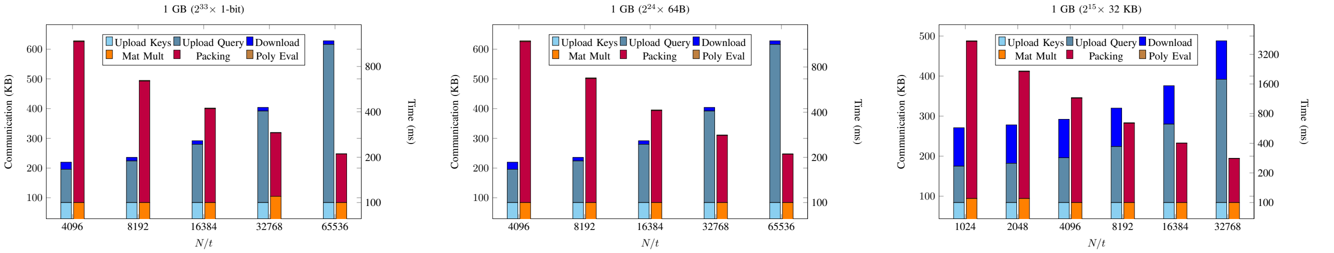

Figure 3: Breakdown of Communication and Online Time of InsPIRe over 1 GB database for various entry sizes as a function of the first dimension (N/t). In each case, we use the largest polynomial degree that satisfies correctness.

7.3. PIR Evaluation with Various Entry Sizes

In the final section of our evaluation, we compare all our construction with existing work across various database sizes with different entry sizes. Tables 2, 3, and 4 show the comparison between our constructions and related work for 1-bit, 64B, and 32KB entries, respectively. In each case, we compare with the most competitive protocols that support the specified entry size.

TABLE 2: PIR performance metrics for different database sizes with 1-bit entries.

Metric	YPIR	SimpleYPIR	KSPIR	HintlessPIR	InsPIRe$_0$	InsPIRe$^{(2)}$						InsPIRe		
1 GB ($2^{33}\times$ 1-bit)														
Offline	32 s	119 s	14 s	213 s	43 s	103 s	66 s	47 s	42 s	37 s	35 s	87 s	77 s	99 s
Upload (Keys)	462 KB	462 KB	2352 KB	360 KB	84 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	84 KB	84 KB	84 KB
Upload (Query)	384 KB	112 KB	14 KB	128 KB	384 KB	40 KB	60 KB	109 KB	113 KB	214 KB	215 KB	140 KB	196 KB	308 KB
Download	12 KB	228 KB	224 KB	1748 KB	36 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	12 KB	12 KB	12 KB
Total Comm.	858 KB	802 KB	2590 KB	2236 KB	504 KB	172 KB	191 KB	241 KB	244 KB	345 KB	347 KB	236 KB	292 KB	404 KB
Server Time	140 ms	600 ms	780 ms	750 ms	120 ms	1100 ms	670 ms	470 ms	440 ms	360 ms	320 ms	650 ms	400 ms	280 ms
Throughput	7420 MB/s	1720 MB/s	1310 MB/s	1370 MB/s	8750 MB/s	930 MB/s	1530 MB/s	2200 MB/s	2310 MB/s	2880 MB/s	3230 MB/s	1580 MB/s	2530 MB/s	3670 MB/s
8 GB ($2^{36}\times$ 1-bit)														
Offline	187 s	283 s	118 s	1500 s	200 s	402 s	235 s	157 s	118 s	94 s		581 s	597 s	579 s
Upload (Keys)	462 KB	462 KB	2688 KB	360 KB	84 KB	80 KB	80 KB	80 KB	80 KB	80 KB		84 KB	84 KB	84 KB
Upload (Query)	1024 KB	448 KB	14 KB	512 KB	1024 KB	106 KB	132 KB	225 KB	431 KB	855 KB		532 KB	532 KB	980 KB
Download	12 KB	444 KB	224 KB	3316 KB	36 KB	52 KB	52 KB	52 KB	52 KB	52 KB		12 KB	96 KB	12 KB
Total Comm.	1498 KB	1354 KB	2926 KB	4188 KB	1144 KB	238 KB	264 KB	357 KB	562 KB	986 KB		628 KB	712 KB	1076 KB
Server Time	830 ms	1850 ms	5910 ms	2030 ms	810 ms	4680 ms	2670 ms	1850 ms	1390 ms	1150 ms		1360 ms	1340 ms	1120 ms
Throughput	9830 MB/s	4420 MB/s	1390 MB/s	4040 MB/s	10060 MB/s	1750 MB/s	3070 MB/s	4440 MB/s	5880 MB/s	7150 MB/s		6010 MB/s	6110 MB/s	7320 MB/s
32 GB ($2^{38}\times$ 1-bit)														
Offline	724 s	706 s	445 s	5700 s	731 s	589 s	421 s	379 s	339 s	318 s	291 s	2306 s	2128 s	2340 s
Upload (Keys)	462 KB	462 KB	2912 KB	360 KB	84 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	84 KB	84 KB	84 KB
Upload (Query)	2048 KB	896 KB	14 KB	1024 KB	2048 KB	265 KB	450 KB	477 KB	861 KB	874 KB	1709 KB	980 KB	1876 KB	1876 KB
Download	12 KB	888 KB	224 KB	6452 KB	36 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	96 KB	12 KB	96 KB
Total Comm.	2522 KB	2246 KB	3150 KB	7836 KB	2168 KB	397 KB	582 KB	609 KB	993 KB	1006 KB	1841 KB	1160 KB	1972 KB	2056 KB
Server Time	3390 ms	5610 ms	50570 ms	5910 ms	3390 ms	7390 ms	5280 ms	5260 ms	4430 ms	4410 ms	4010 ms	4320 ms	3880 ms	3820 ms
Throughput	9680 MB/s	5840 MB/s	650 MB/s	5550 MB/s	9670 MB/s	4430 MB/s	6210 MB/s	6230 MB/s	7400 MB/s	7430 MB/s	8180 MB/s	7580 MB/s	8450 MB/s	8570 MB/s

TABLE 3: PIR performance metrics for different database sizes with 64 B entries.

Metric	SimpleYPIR	KSPIR	HintlessPIR	InsPIRe$^{(2)}$						InsPIRe				
1 GB ($2^{24}\times$ 64B)														
Offline Time	119 s	14 s	213 s	103 s	66 s	47 s	42 s	37 s	35 s	87 s	77 s	98 s	99 s	
Upload (Keys)	462 KB	2352 KB	360 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	84 KB	84 KB	84 KB	84 KB	
Upload (Query)	112 KB	14 KB	128 KB	40 KB	60 KB	109 KB	113 KB	214 KB	215 KB	140 KB	196 KB	308 KB	532 KB	
Download	228 KB	224 KB	1748 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	12 KB	12 KB	12 KB	12 KB	
Total Comm.	802 KB	2590 KB	2236 KB	172 KB	191 KB	241 KB	244 KB	345 KB	347 KB	236 KB	292 KB	404 KB	628 KB	
Server Time	600 ms	780 ms	750 ms	1100 ms	670 ms	470 ms	440 ms	360 ms	320 ms	650 ms	400 ms	280 ms	210 ms	
Throughput	1720 MB/s	1310 MB/s	1370 MB/s	930 MB/s	1530 MB/s	2200 MB/s	2310 MB/s	2880 MB/s	3230 MB/s	1580 MB/s	2530 MB/s	3670 MB/s	4850 MB/s	
8 GB ($2^{27}\times$ 64B)														
Offline Time	283 s	118 s	1500 s	402 s	235 s	157 s	118 s	94 s		581 s	597 s	579 s	467 s	
Upload (Keys)	462 KB	2688 KB	360 KB	80 KB	80 KB	80 KB	80 KB	80 KB		84 KB	84 KB	84 KB	84 KB	
Upload (Query)	448 KB	14 KB	512 KB	106 KB	132 KB	225 KB	431 KB	855 KB		532 KB	532 KB	980 KB	1876 KB	
Download	444 KB	224 KB	3316 KB	52 KB	52 KB	52 KB	52 KB	52 KB		12 KB	96 KB	12 KB	96 KB	
Total Comm.	1354 KB	2926 KB	4188 KB	238 KB	264 KB	357 KB	562 KB	986 KB		628 KB	712 KB	1076 KB	2056 KB	
Server Time	1850 ms	5910 ms	2030 ms	4680 ms	2670 ms	1850 ms	1390 ms	1150 ms		1360 ms	1340 ms	1120 ms	1020 ms	
Throughput	4420 MB/s	1390 MB/s	4040 MB/s	1750 MB/s	3070 MB/s	4440 MB/s	5880 MB/s	7150 MB/s		6010 MB/s	6110 MB/s	7320 MB/s	8050 MB/s	
32 GB ($2^{29}\times$ 64B)														
Offline Time	706 s	446 s	5700 s	589 s	421 s	379 s	339 s	318 s	291 s	2306 s	2128 s	2340 s	2114 s	2111 s
Upload (Keys)	462 KB	2912 KB	360 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	84 KB	84 KB	84 KB	84 KB	84 KB
Upload (Query)	896 KB	14 KB	1024 KB	265 KB	450 KB	477 KB	861 KB	874 KB	1709 KB	980 KB	1876 KB	1876 KB	3668 KB	7252 KB
Download	888 KB	224 KB	6452 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	96 KB	12 KB	96 KB	12 KB	12 KB
Total Comm.	2246 KB	3150 KB	7836 KB	397 KB	582 KB	609 KB	993 KB	1006 KB	1841 KB	1160 KB	1972 KB	2056 KB	3764 KB	7348 KB
Server Time	5610 ms	50570 ms	5910 ms	7390 ms	5280 ms	5260 ms	4430 ms	4410 ms	4010 ms	4320 ms	3880 ms	3820 ms	3630 ms	3510 ms
Throughput	5840 MB/s	650 MB/s	5550 MB/s	4430 MB/s	6210 MB/s	6230 MB/s	7400 MB/s	7430 MB/s	8180 MB/s	7580 MB/s	8450 MB/s	8570 MB/s	9030 MB/s	9330 MB/s

Our evaluation shows that for all configurations, our proposed constructions outperform existing work. For 1-bit entries, InsPIRe$_0$ is strictly better than related work, both in terms of communication cost and server runtime. Table 2 shows at least one data point for InsPIRe that outperforms both in communication and server runtime. However, InsPIRe can achieve even lower communication costs at the cost of higher runtime as shown in Table 9.

The breakdown of the communication cost shows that the cryptographic key material of InsPIRe$_0$ and InsPIRe is over 4x smaller than all other protocols. This is due to the smaller keys used in InspiRING. Similarly, the query and the response size of InsPIRe are smaller than that of all other protocols. This is mainly due to our homomorphic polynomial evaluation technique that allows us to efficiently reduce the PIR response using small additional query overhead.

In summary, InsPIRe strictly improves over existing PIR schemes with server-side preprocessing, and may also be parameterized to optimize specific metrics such as communication.

7.4. Benchmarking Ring Packing

In this section, we provide a benchmark of InspiRING compared to existing ring packing algorithms such as CDKS [18] and the packing algorithms used in HintlessPIR [53]. In this benchmark, we pack 2^{12} LWE ciphertexts and measure the offline and online runtime of this procedure. Additionally, we report the size of additional material the client must provide for packing, i.e., the packing keys in the case of InspiRING. We run each protocol using the parameters provided in the respective paper or implementation to achieve the best performance and compare with InsPIRe that is instantiated with similar parameters. Using the notation of Section 3, we use two parameters set for InspiRING which provide 128-bit security based on the lattice estimator:

TABLE 4: PIR performance metrics for different database sizes with 32 KB entries.

Metric	**SimpleYPIR**	**KSPIR**	**HintlessPIR**	InsPIRe						
1 GB ($2^{15}\times$ 32 KB)										
Offline Time	119 s	14 s	213 s	307 s	177 s	106 s	83 s	72 s	86 s	
Upload (Keys)	462 KB	2352 KB	360 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	
Upload (Query)	112 KB	14 KB	128 KB	91 KB	98 KB	112 KB	140 KB	196 KB	308 KB	
Download	228 KB	224 KB	1748 KB	96 KB	96 KB	96 KB	96 KB	96 KB	96 KB	
Total Comm.	802 KB	2590 KB	2236 KB	271 KB	278 KB	292 KB	320 KB	376 KB	488 KB	
Server Time	600 ms	780 ms	750 ms	4270 ms	2210 ms	1170 ms	640 ms	410 ms	280 ms	
Throughput	1720 MB/s	1310 MB/s	1370 MB/s	240 MB/s	460 MB/s	880 MB/s	1600 MB/s	2500 MB/s	3620 MB/s	
8 GB ($2^{18}\times$ 32 KB)										
Offline Time	283 s	118 s	1500 s	547 s	470 s	598 s	597 s	538 s	467 s	
Upload (Keys)	462 KB	2688 KB	360 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	
Upload (Query)	448 KB	14 KB	512 KB	140 KB	196 KB	308 KB	532 KB	980 KB	1876 KB	
Download	444 KB	224 KB	3316 KB	96 KB	96 KB	96 KB	96 KB	96 KB	96 KB	
Total Comm.	1354 KB	2926 KB	4188 KB	320 KB	376 KB	488 KB	712 KB	1160 KB	2056 KB	
Server Time	1850 ms	5910 ms	2030 ms	5220 ms	3010 ms	1920 ms	1340 ms	1120 ms	1020 ms	
Throughput	4420 MB/s	1390 MB/s	4040 MB/s	1570 MB/s	2720 MB/s	4260 MB/s	6110 MB/s	7280 MB/s	8050 MB/s	
32 GB ($2^{20}\times$ 32 KB)										
Offline Time	706 s	446 s	5700 s	2052 s	2581 s	2569 s	2306 s	2340 s	2053 s	1945 s
Upload (Keys)	462 KB	2912 KB	360 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB
Upload (Query)	896 KB	14 KB	1024 KB	196 KB	308 KB	532 KB	980 KB	1876 KB	3668 KB	7252 KB
Download	888 KB	224 KB	6452 KB	192 KB	96 KB	96 KB	96 KB	96 KB	96 KB	96 KB
Total Comm.	2246 KB	3150 KB	7836 KB	472 KB	488 KB	712 KB	1160 KB	2056 KB	3848 KB	7432 KB
Server Time	5610 ms	50570 ms	5910 ms	15470 ms	7640 ms	5480 ms	4320 ms	3820 ms	3650 ms	3500 ms
Throughput	5840 MB/s	650 MB/s	5550 MB/s	2120 MB/s	4290 MB/s	5980 MB/s	7580 MB/s	8570 MB/s	8980 MB/s	9360 MB/s

1) $\log_2 d = 10$, $\log_2 q = 28$, $\log_2 p = 6$, $\ell = 8$, $z = 2^4$
2) $\log_2 d = 11$, $\log_2 q = 56$, $\log_2 p = 15$, $\ell = 3$, $z = 2^{19}$

Table 5 summarizes the results.

TABLE 5: Concrete costs of packing 2^{12} LWE ciphertexts into RLWE ciphertexts using the approach in HintlessPIR [53], CDKS [18] and InspiRING.

Packing Type	HintlessPIR	InspiRING	CDKS	InspiRING
Unpacked Size	16 MB	14 MB	56 MB	56 MB
$\log_2(d, q, p)$	(10, 32, 8)	(10, 28, 6)	(11, 56, 15)	(11, 56, 15)
Key Material	360 KB	**60 KB**	462 KB	**84 KB**
Packed Size	180 KB	**32 KB**	56 KB	**56 KB**
Total Size	540 KB	**92 KB**	518 KB	**140 KB**
Offline Runtime	2.0 s	2.4 s	11 s	36 s
Online Runtime	141 ms	**16 ms**	56 ms	**40 ms**

We make the following observations from this table. Firstly, InspiRING requires significantly smaller key material compared to existing work, specifically, 84%, 76%, and over 99% less key material than CDKS and HintlessPIR, respectively. Secondly, the online time of InspiRING is 28% lower than the fastest existing work, CDKS. This, however, comes at the cost of a slower offline phase, which can be attributed to the quadratic dependence on d

Lastly, we confirm the analytic noise analysis of InspiRING from Theorem 2. In the same experiment, we observe that the bitlength of $||e_{pack}||_\infty$ for ciphertexts packed using CDKS and InsPIRe ($d = 2048$) are equal to 38.5 and 33.4, respectively. This confirms that proposed ring packing construction has less noise growth than CDKS, up to 5 bits in this example.

8. Applications of InsPIRe

The server-side preprocessing and low communication of InsPIRe makes it an ideal candidate for the applications of PIR where an apriori setup is not feasible. We explore two such applications and show how InsPIRe is suitable solution with very good performance. The first application is the use of PIR in IPFS [60] for private queries and the second is private device enrollment in Chrome OS [80]. We use the same experimental setup as Section 7 of single-threaded execution on an Intel Xeon CPU @ 2.6 GHz.

8.1. Private Queries in IPFS

IPFS is a distributed file system which gives clients the ability to access content by querying multiple peers in the network to acquire the necessary routing information and eventually, the content itself. The process of locating and retrieving content can be simplified to three types of database queries, i.e., peer routing, content discovery, and content retrieval. Mazmudar et al. [60] showed how these queries can be done privately with PIR. However, existing PIR protocols require exchanging large key material or a setup phase, which are not compatible with the query pattern in IPFS. A user iteratively contacts different servers to access different parts of the routing information, so the high cost of setup is not amortized over many queries. Hence, the authors proposed alternative PIR protocols which required no setup and had low overall communication costs.

InsPIRe can be used as a more suitable solution for this application, given the server-side preprocessing and extremely low communication. Table 6 shows examples of the communication and computation costs of using PIR in IPFS. We provide the communication and computation cost of the best solution for each functionality of Mazmudar et al. [60] and show the corresponding cost of using InsPIRe. In the case of Content Discovery and Content Retrieval, we use InsPIRe with the parameters specified in Section 7 but in Peer Routing, due to the extremely small database size, we use smaller parameters ($d = 1024$) to achieve very low communication cost. InsPIRe provides strict improvement in communication and computation for all three functionalities.

TABLE 6: Costs of PIR queries in the three functionalities of IPFS. *Comm.* denotes the total communication required to serve the query and *Comp.* denotes the server runtime.

Step	**Metric**	**Current** [60]	**InsPIRe**	**Improvement**
Peer Routing (256 × 1.5 KB)	Comm. Comp.	$>$ 100 KB $>$ 56 ms	69 KB 51 ms	32% 8%
Content Discovery (200k Records)	Comm. Comp.	$>$ 280 KB $>$ 875 ms	128 KB 123 ms	46% 86%
Content Retrieval ($2^{14}\times$ 256 KB)	Comm. Comp.	$>$ 2.1 MB $>$ 3.0 s	1.02 MB 1.5 s	51% 50%

8.2. Privacy-Preserving Device Enrollment

Upon starting a Chrome OS device, the enrollment process requires checking the membership of the device in a server-held database. Since the introduction of Chrome 94, Google has performed this membership check in a privacy-preserving manner using techniques such as PIR [80]. As Chrome devices perform the device enrollment procedure with PIR immediately when it is powered on for the first time or right after a factory reset, there is no opportunity for the device to perform setup with the server. InsPIRe is an ideal solution for this application due to the server-side preprocessing step. While the precise number of Chromebooks is not disclosed, publicly available data shows over 20 million devices are sold every year from 2019 until 2023[5]. We assume each device requires about 64 bytes of information from the database and provide runtimes for various number of devices. The cost of device enrollment using InsPIRe is shown in Table 7.

TABLE 7: Cost of PIR for Chrome Device Enrollment

Devices	**DB Size**	**Offline Time**	**Communication**	**Response Time**
20 M	1.19 GB	78 s	292 KB	416 ms
40 M	2.38 GB	131 s	292 KB	815 ms
80 M	4.76 GB	241 s	304 KB	1400 ms

Table 7 shows that the concrete cost of using PIR for device enrollment is only a few hundred KiloBytes, which is very practical network cost for a personal computer.

9. Related Works

PIR was introduced by Chor *et al.* [24] in the multi-server setting and Kushilevitz and Ostrovsky [48] in the single-server setting. Most early single-server PIR schemes were built upon number-theoretic assumptions including [71], [29], [70] using a linearly homomorphic encryption scheme. Multi-server PIR has been built without any cryptographic assumptions [35] and one-way functions [11].

[5] https://www.statista.com/statistics/749890/worldwide-chromebook-unit-shipments/

PIR using Server-Stored Client-Specific Keys. In recent years, many works aim to design practically efficient single-server PIR schemes using lattice-based homomorphic encryption starting with XPIR [61] leading to a long line of work including [4], [3], [67], [1], [62], [16]. For efficiency purposes, these protocols assume that the server stores a key that is specific to each client that performs PIR queries. With these server-stored client-specific keys, these schemes obtain very low query communication, but require large computation due to relying on RLWE ciphertexts.

PIR using Client-Stored Database Hints. In another line of work, prior works consider single-server PIR where clients download and store database hints that will be used later for queries. The works of SimplePIR [43] and FrodoPIR [30] built schemes using LWE, but require each client to download large database hints before querying. In an orthogonal line, it was shown that client-stored database hints could enable sublinear query times [72], [28], [47], [27], [44]. Several very recent works have considered practical single-server implementations of this PIR paradigm such as [81], [52], [76], [37]. However, these works require either streaming the entire database in the offline phase or requiring the server to perform heavy computation per client.

PIR using the CRS Model. Very recently, several works have considered PIR without offline communication to avoid the obstacles and costs of offline preprocessing. Starting with Tiptoe [42], follow-ups such as HintlessPIR [53] and YPIR [63] consider practical PIR in the CRS model. Prior to InsPIRe, all these works sacrificed larger query communication in order to eliminate offline communication. In a more theoretical line of work, a recent breakthrough work [56] presented a doubly efficient PIR construction with sublinear query time and silent preprocessing from RLWE, but the practical costs remain impractical [69].

PIR with Additional Features. PIR with more advanced query functionalities have also been studied such as batch PIR that efficiently retrieves multiple entries at once. Some examples include [57], [41], [5], [4], [79], [68], [8]. Constructions for keyword PIR has also been presented where queries are performed over sparse databases consisting of keys and values (see [3], [59], [73], [17] as examples). PIR with additional security features has also been studied such as in the presence of malicious servers including [25], [31] as well as symmetric PIR with database privacy [3], [55].

Ring Packing. The problem of ring packing to transform LWE to RLWE ciphertexts has been studied in three different classes. The row method [18] uses the least amount of cryptographic material while the diagonal [45] and column methods [22], [65], [10], [7] use more cryptographic material to obtain faster packing times.

Labeled Unbalanced PSI. The idea of using polynomial interpolation to represent sets and homomorphic polynomial evaluation for retrieval have been explored in the context of labeled unbalanced PSI [20], [19], [26], but the detailed constructions are quite different from ours. In particular, in these works, the interpolation is done over the plaintext SIMD slots (e.g. over $\mathbb{Z}_p$), while our polynomial interpolation is done over the *entire plaintext ring* R_p. Importantly, the PSI constructions encode the evaluation points as SIMD slots which are then encrypted as RLWE ciphertexts. The encrypted plaintexts generally have high norms, and so each homomorphic multiplication results in a large multiplicative noise growth. To overcome the large depth homomorphic multiplications, these works proposed tricks such as sending multiple powers of the evaluation points to the sender. In contrast, our homomorphic evaluation uses RGSW encrypted unit monomials which only incur additive noise growth for each multiplication, allowing the use of efficient RLWE parameters.

10. Conclusions

We present InsPIRe that obtains higher throughput and smaller query communication than all prior PIR schemes, using only server-side preprocessing. Along the way, we introduce a novel ring packing algorithm, InspiRING, requiring smaller cryptographic material and faster online packing times as well as a new approach to PIR using homomorphic polynomial evaluation.

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. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[3] Asra Ali, Tancrède 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: 30th USENIX Security Symposium*, pages 1811–1828. USENIX Association, August 11–13, 2021.

[4] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. PIR with Compressed Queries and Amortized Query Processing. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 962–979, San Francisco, CA, May 2018. IEEE.

[5] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 551–569, 2016.

[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] 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, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 37–69, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[8] 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: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[9] Nikita Borisov, George Danezis, and Ian Goldberg. DP5: A private presence service. *Proceedings on Privacy Enhancing Technologies*, 2015(2):4–24, April 2015.

[10] 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.

[11] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1292–1303, Vienna, Austria, October 24–28, 2016. ACM Press.

[12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual cryptology conference*, pages 868–886. Springer, 2012.

[13] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. *SIAM Journal on computing*, 43(2):831–871, 2014.

[15] D. J. Britten and F. W. Lemire. A structure theorem for rings supporting a discrete fourier transform. *SIAM Journal on Applied Mathematics*, 41(2):222–226, 1981.

[16] Alexander Burton, Samir Jordan Menon, and David J Wu. Respire: High-rate pir for databases with small records. In *ACM CCS*, 2024.

[17] Sofía Celi and Alex Davidson. Call me by my name: Simple, practical private information retrieval for keyword queries. In *ACM CCS*, 2024.

[18] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient Homomorphic Conversion Between (Ring) LWE Ciphertexts. In Kazue Sako and Nils Ole Tippenhauer, editors, *Applied Cryptography and Network Security*, volume 12726, pages 460–479. Springer International Publishing, Cham, 2021.

[19] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled psi from fully homomorphic encryption with malicious security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1223–1237, 2018.

[20] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1243–1255, 2017.

[21] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *international conference on the theory and application of cryptology and information security*, pages 3–33. Springer, 2016.

[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, *Advances in Cryptology – ASIACRYPT 2017, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 377–408, Hong Kong, China, December 3–7, 2017. Springer, Cham, Switzerland.

[23] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[24] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *Journal of the ACM*, 45(6):965–981, November 1998.

[25] 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: 32nd USENIX Security Symposium*, pages 3835–3851, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

[26] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled psi from homomorphic encryption with reduced computation and communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1135–1150, 2021.

[27] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 3–33, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[28] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 44–75, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

[29] 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: 4th International Workshop on Theory and Practice in Public Key Cryptography*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136, Cheju Island, South Korea, February 13–15, 2001. Springer Berlin Heidelberg, Germany.

[30] Alex Davidson, Gonçalo Pestana, and Sofía Celi. FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval. *Proceedings on Privacy Enhancing Technologies*, 2023(1):365–383, January 2023.

[31] Leo de Castro and Keewoo Lee. Verisimplepir: verifiability in simplepir at no online cost for honest servers. In *USENIX Security*, 2024.

[32] Leo de Castro, Kevin Lewi, and Edward Suh. WhisPIR: Stateless private information retrieval with low communication. Cryptology ePrint Archive, Paper 2024/266, 2024.

[33] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies*, 2018(4):159–178, October 2018.

[34] David Steven Dummit, Richard M Foote, et al. *Abstract algebra*, volume 3. Wiley Hoboken, 2004.

[35] Zeev Dvir and Sivakanth Gopi. 2-server PIR with sub-polynomial communication. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 577–584, Portland, OR, USA, June 14–17, 2015. ACM Press.

[36] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[37] Ben Fisch, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. ThorPIR: Single server PIR via homomorphic thorp shuffles. In *ACM CCS*, 2024.

[38] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[39] Matthew Green, Watson Ladd, and Ian Miers. A protocol for privately reporting ad impressions at scale. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1591–1601, Vienna, Austria, October 24–28, 2016. ACM Press.

[40] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In *13th USENIX symposium on networked systems design and implementation (NSDI 16)*, pages 91–107, 2016.

[41] Ryan Henry. Polynomial batch codes for efficient IT-PIR. *Proceedings on Privacy Enhancing Technologies*, 2016(4):202–218, October 2016.

[42] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th symposium on operating systems principles*, pages 396–416, 2023.

[43] 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 *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3889–3905, Anaheim, CA, August 2023. USENIX Association.

[44] Alexander Hoover, Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Plinko: Single-server PIR with efficient updates via invertible PRFs. Cryptology ePrint Archive, Report 2024/318, 2024.

[45] Wen jie 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, San Francisco, CA, USA, May 24–27, 2021. IEEE Computer Society Press.

[46] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019: 28th USENIX Security Symposium*, pages 1447–1464, Santa Clara, CA, USA, August 14–16, 2019. USENIX Association.

[47] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 875–892. USENIX Association, August 11–13, 2021.

[48] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science*, pages 364–373, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[49] Albert Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. Riffle: An efficient communication system with strong anonymity. *Proceedings on Privacy Enhancing Technologies*, 2016(2):115–134, April 2016.

[50] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 2015.

[51] 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.

[52] Arthur Lazzaretti and Charalampos Papamanthou. Single pass client-preprocessing private information retrieval. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[53] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz. Hintless single-server private information retrieval. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part IX*, volume 14928 of *Lecture Notes in Computer Science*, pages 183–217, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.

[54] 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: 26th Conference on Computer and Communications Security*, pages 1387–1403, London, UK, November 11–15, 2019. ACM Press.

[55] 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: 27th European Symposium on Research in Computer Security, Part I*, volume 13554 of *Lecture Notes in Computer Science*, pages 217–236, Copenhagen, Denmark, September 26–30, 2022. Springer, Cham, Switzerland.

[56] 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 Annual ACM Symposium on Theory of Computing*, pages 595–608, Orlando, FL, USA, June 20–23, 2023. ACM Press.

[57] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In Rainer Böhme and Tatsuaki Okamoto, editors, *FC 2015: 19th International Conference on Financial Cryptography and Data Security*, volume 8975 of *Lecture Notes in Computer Science*, pages 168–186, San Juan, Puerto Rico, January 26–30, 2015. Springer Berlin Heidelberg, Germany.

[58] Ming Luo, Feng-Hao Liu, and Han Wang. Faster fhe-based single-server private information retrieval. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1405–1419, 2024.

[59] 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: 31st USENIX Security Symposium*, pages 1723–1740, Boston, MA, USA, August 10–12, 2022. USENIX Association.

[60] Miti Mazmudar, Shannon Veitch, and Rasoul Akhavan Mahdavi. Peer2PIR: Private Queries for IPFS . In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 4438–4456, San Fransisco, CA, USA, May 2025. IEEE Computer Society.

[61] 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.

[62] 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 (SP)*, pages 930–947, San Francisco, CA, USA, May 2022. IEEE.

[63] 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: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[64] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer Berlin Heidelberg, Germany.

[65] Daniele Micciancio and Jessica Sorrell. Ring packing and amortized FHEW bootstrapping. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *ICALP 2018: 45th International Colloquium on Automata, Languages and Programming*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 100:1–100:14, Prague, Czech Republic, July 9–13, 2018. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[66] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. Pir-tor: scalable anonymous communication using private information retrieval. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, page 31, USA, 2011. USENIX Association.

[67] Muhammad Haris Mughees, Hao Chen, and Ling Ren. OnionPIR: Response efficient single-server PIR. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2292–2306, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[68] Muhammad Haris Mughees and Ling Ren. Vectorized batch private information retrieval. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 437–452, 2023.

[69] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. Towards practical doubly-efficient private information retrieval. In *Financial Cryptography and Data Security 2024 (FC '24)*, 2024.

[70] 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: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 393–411, Beijing, China, April 16–20, 2007. Springer Berlin Heidelberg, Germany.

[71] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer Berlin Heidelberg, Germany.

[72] 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: 25th Conference on Computer and Communications Security*, pages 1002–1019, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[73] 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: 32nd USENIX Security Symposium*, pages 3853–3870, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

[74] Chris Peikert et al. A decade of lattice cryptography. *Foundations and trends® in theoretical computer science*, 10(4):283–424, 2016.

[75] Chris Peikert and Brent Waters. Lossy trapdoor functions and their applications. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 187–196, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

[76] Ling Ren, Muhammad Haris Mughees, and I Sun. Simple and practical amortized sublinear private information retrieval. In *ACM CCS*, 2024.

[77] 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: 28th USENIX Security Symposium*, pages 1556–1571, Santa Clara, CA, USA, August 14–16, 2019. USENIX Association.

[78] Frank Wang, Catherine Yun, Shafi Goldwasser, Vinod Vaikuntanathan, and Matei Zaharia. Splinter: Practical private queries on public data. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 299–313, 2017.

[79] Kevin Yeo. Cuckoo hashing in cryptography: Optimal parameters, robustness and applications. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part IV*, volume 14084 of *Lecture Notes in Computer Science*, pages 197–230, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[80] 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.

[81] 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 (SP)*, pages 4296–4314. IEEE Computer Society, 2024.

Appendix

1. Security and Correctness Definitions of PIR

Security. We define single query privacy for a PIR scheme using a standard indistinguishability game.

1) **Setup Phase:**
   a) On input security parameter 1^λ, $\mathcal{A}$ outputs a database $\mathbf{D}$.
   b) The challenger computes $(\mathsf{pp}, \mathbf{D}') \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{D})$ and gives it to $\mathcal{A}$.
2) **Challenge Phase:**
   a) $\mathcal{A}$ outputs a pair of challenge indices $(\mathsf{idx}^{(0)}, \mathsf{idx}^{(1)})$.
   b) The challenger chooses a bit $b \in \{0, 1\}$ uniformly at random.
   c) The challenger computes the challenge query $(\mathsf{st}^{(b)}, \mathsf{qry}^{(b)}) \leftarrow \mathsf{Query}(\mathsf{pp}, \mathsf{idx}^{(b)})$. The challenger sends $\mathsf{qry}^{(b)}$ to $\mathcal{A}$.
3) **Guess Phase:** $\mathcal{A}$ outputs a bit b'.

The advantage of the adversary $\mathcal{A}$ in this game is:

$$\mathsf{Adv}^{\mathsf{QP}}_{\mathcal{A}}(\lambda) = |\Pr[b' = b] - (1/2)|$$

We say that a PIR scheme achieves single query privacy if for any PPT adversary $\mathcal{A}$, its advantage $\mathsf{Adv}^{\mathsf{QP}}_{\mathcal{A}}(\lambda)$ is a negligible function in λ. As in SimplePIR [43], we may extend the notion of single query privacy to a sequence of Q queries. We defer the details to the full paper.

Correctness. For all $\lambda \in \mathbb{N}$, all databases $\mathbf{D}$, and all indices idx, let $(\mathsf{pp}, \mathbf{D}') \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{D})$, $(\mathsf{st}, \mathsf{qry}) \leftarrow \mathsf{Query}(\mathsf{pp}, \mathsf{idx})$ and $\mathsf{resp} \leftarrow \mathsf{Respond}(\mathsf{pp}, \mathbf{D}', \mathsf{qry})$. For a correctness parameter δ, we say that a PIR protocol is $(1-\delta)$-correct if $\Pr[\mathsf{Extract}(\mathsf{pp}, \mathsf{st}, \mathsf{resp}) = \mathbf{D}[\mathsf{idx}]] \geq 1 - \delta$.

2. LWE to Intermediate Ciphertexts

We give a detailed mathematical derivation of the first stage of our packing algorithm that transforms the input LWE ciphertexts into intermediate ciphertexts. As discussed, our approach transforms the RLWE ciphertext $(\tilde{a}, \tilde{b})$ into an intermediate ciphertext that resembles an MLWE ciphertext. A key characteristic of this transformed ciphertext is that its non-constant term coefficients are zeroed out. Furthermore, these intermediate ciphertexts retain homomorphic properties, enabling aggregation of multiple ciphertexts into a single one that contains the LWE messages as the coefficients.

To facilitate this transformation, we introduce an alternative interpretation of the $\tilde{b}$ component, leveraging Lemma 1. In essence, Lemma 1 states that for any polynomial $p(X) = \sum_{i=0}^{d-1} c_i X^i$, summing $d/2$ versions of $\tau_g^j(p) + \tau_h \circ \tau_g^j(p)$ (obtained by applying the automorphism $X \mapsto X^{g^j}$ and $X \mapsto X^{g^j \cdot h}$ for $j \in \{0, \ldots, d/2-1\}$) yields a polynomial where all coefficients are zero except the constant term, which is scaled by a factor of d. As we will show, this operation is crucial for isolating the LWE message m into a constant term polynomial.

Our initial step involves applying the operator Tr from Lemma 1 to the $\tilde{b}$ component of the RLWE representation $(\tilde{a}, \tilde{b})$. More specifically, from Eq. 1, we may re-interpret everything over $\mathbb{Z}[X]/(X^d+1)$ and obtain $\tilde{b} = -\tilde{a}\tilde{s} + \tilde{m} + q\tilde{u}$ for some $\tilde{u} \in \mathbb{Z}[X]/(X^d+1)$. Applying Tr to both sides, we get:

$$\begin{aligned}\mathrm{Tr}(\tilde{b}) &= \mathrm{Tr}(-\tilde{a}\tilde{s} + \tilde{m} + q\tilde{u}) \\ &= -\mathrm{Tr}(\tilde{a}\tilde{b}) + \mathrm{Tr}(\tilde{m}) + q \cdot \mathrm{Tr}(\tilde{u}).\end{aligned}$$

Given that $\tilde{b}$ is a constant polynomial, $\mathrm{Tr}(\tilde{b}) = \sum_{j=0}^{d/2-1} \tau_g^j(\tilde{b}) + \tau_h \circ \tau_g^j(\tilde{b}) = d \cdot \tilde{b}$. Assuming d is invertible mod q (which is true when q is odd), we can write:

$$\begin{aligned}\tilde{b} &= d^{-1}\left(-\mathrm{Tr}(\tilde{a}\tilde{s}) + \mathrm{Tr}(\tilde{m})\right) \pmod{q} \\ &= -d^{-1}\mathrm{Tr}(\tilde{a}\tilde{s}) + d^{-1}\mathrm{Tr}(\tilde{m}) \pmod{q}.\end{aligned}$$

We have $\mathrm{Tr}(\tilde{m}) = d \cdot m$. We define a new message representation as $\hat{m}(X) := d^{-1} \cdot \mathrm{Tr}(\tilde{m}) = m$.

Substituting this expression for $\tilde{b}$:

$$\begin{aligned}\tilde{b} &= -d^{-1} \cdot \mathrm{Tr}(\tilde{a}\tilde{b}) + \hat{m} \pmod q \\ &= -\left(\sum_{j=0}^{d/2-1} \left(d^{-1} \cdot \tau_g^j(\tilde{a})\right) \cdot \tau_g^j(\tilde{s})\right) \\ &\quad -\left(\sum_{j=0}^{d/2-1} \left(d^{-1} \cdot \tau_h \circ \tau_g^j(\tilde{a})\right) \cdot \left(\tau_h \circ \tau_g^j(\tilde{s})\right)\right) + \hat{m} \pmod q \\ &= -\langle \hat{\mathbf{a}}, \hat{\mathbf{s}} \rangle + \hat{m} \pmod q\end{aligned}$$

The second equality follows from the definition of Tr and the property that τ_g and τ_h are ring automorphisms. The final equality is written succinctly as an inner product of two vectors $\hat{\mathbf{a}}, \hat{\mathbf{s}} \in R_q^d$. Here, $\hat{\mathbf{a}}[j] := d^{-1} \cdot \tau_g^j(\tilde{a})$ and $\hat{\mathbf{a}}[j + d/2] := d^{-1} \cdot \tau_h \circ \tau_g^j(\tilde{a})$ for $j < d/2$ e.g. the second half is an automorphic image of the first half by τ_h. Similarly, $\hat{\mathbf{s}}[j] := \tau_g^j(\tilde{s})$ and $\hat{\mathbf{s}}[j + d/2] := \tau_h \circ \tau_g(\tilde{s})$ for $j < d/2$.

This formulation allows $\tilde{b}$ to be interpreted as a pseudorandom component of an MLWE-like ciphertext with the random component $\hat{\mathbf{a}}$ and the secret key $\hat{\mathbf{s}}$.

3. Conversion to RLWE Ciphertext

This section provides the detailed procedure for converting the aggregated intermediate ciphertext $\mathsf{IRCtx}(\hat{m}) = (\hat{\mathbf{a}}_{agg}, \tilde{b}_{agg}) \in R_q^d \times R_q$ into a standard two-component RLWE ciphertext. The input $\mathsf{IRCtx}(\hat{m}_{agg})$ encrypts the packed message $\hat{m}_{agg}$ (as defined in the main body) and consists of $d+1$ ring elements. The components $\hat{\mathbf{a}}_{\mathbf{agg}}[j]$ are associated with secret key shares $\hat{\mathbf{s}}[j] = \tau_g^j(\tilde{s})$ and $\hat{\mathbf{s}}[j + d/2] = \tau_h \circ \tau_g^j(\tilde{s})$ for $j < d/2$. The goal is to obtain a ciphertext (a_{fin}, b_{fin}) encrypted under base secret key $\tilde{s}$.

The conversion relies on iteratively reducing the number of components using key-switching.

Key-Switching Preliminaries. Recall the key-switching key generation algorithm $\mathsf{KS.Setup}(s_{in}, s_{out})$ which produces a key $\mathbf{K}$ that can transform a ciphertext component encrypted under s_{in} into one encrypted under s_{out}.

For our specific purpose, we require two elementary key-switching matrices. The first matrix is constructed as $\mathbf{K}_g \leftarrow \mathsf{KS.Setup}(\tau_g(\tilde{s}), \tilde{s})$. By applying Galois automorphisms τ_g^i (resp. $\tau_h \circ \tau_g^i$) to $\mathbf{K}_g$, we obtain a key-switching matrix from $\tau_g^{j+1}(\tilde{s})$ to $\tau_g^j(\tilde{s})$ (resp. $\tau_h \circ \tau_g^{j+1}(\tilde{s})$ to $\tau_h \circ \tau_g^j(\tilde{s})$). The second matrix is constructed as $\mathbf{K}_h \leftarrow \mathsf{KS.Setup}(\tau_h(\tilde{s}), \tilde{s})$ and used in the final step of the algorithm.

The CollapseOne Subroutine. The core building block for this conversion is a subroutine to "collapse" ciphertext components, adapted from standard relinearization techniques. The goal of this subroutine is to use the key-switching to re-express the pseudorandom component masked by one secret key share so that it is now masked by another key share. We refer to these key shares as the source key share and the target key share, respectively.

In line 2, key-switching is performed on the last component $\mathbf{a}[k-1]$ with respect to the current pseudorandom component b. This in turn results in $b' = -\langle \mathbf{a}'', \mathbf{s}' \rangle + m + e_{noise} + e_{ks}$, where $\mathbf{a}''$ and $\mathbf{s}'$ are the "reduced" vectors without the corresponding source key share component. e_{noise} is the initial noise present in the input and e_{ks} is the additive noise from the key-switching. Note that the returned pair $(\mathbf{a}', b')$ is essentially a *partial ciphertext* where $\mathbf{a}'$ masks a portion of the pseudorandom component b'.

The CollapseHalf Subroutine. This subroutine is used to collapse the random components masked by the key shares $\tau_g^k(\tilde{s})$ (resp. $\tau_h \circ \tau_g^k(\tilde{s})$) so that it is masked by a single key share $\tilde{s}$ (resp. $\tau_h(\tilde{s})$).

Using the CollapseOne procedure as the building block, we can iteratively collapse the ciphertext components.

In the k-th iteration of the loop, the component to be eliminated is $\mathbf{a}^{(k)}[k]$, which is associated with the secret $\tau_g^k(\tilde{s})$ (resp. $\tau_h \circ \tau_g^k(\tilde{s})$). The target secret for this component is $\tau_g^{k-1}(\tilde{s})$ (resp. $\tau_h \circ \tau_g^{k-1}(\tilde{s})$). Consequently, the required key-switching matrix is $\mathbf{K}_g^{k-1} = \tau_g^{k-1}(\mathbf{K}_g)$ (resp. $\tau_h \circ \tau_g^{k-1}(\mathbf{K}_g)$ and we can invoke CollapseOne to reduce the number of ciphertext components by one.

After $d/2-1$ such key-switching steps, we obtain $(\mathbf{a}^{(0)}, b^{(0)})$. Note, the masking part of $b^{(0)}$ that was originally masked by secret key shares $\tau_g^j(\tilde{s})$ (resp. $\tau_h \circ \tau_g^j(\tilde{s})$) is now masked by $\tilde{s}$ (resp. $\tau_h(\tilde{s})$) with $\mathbf{a}^{(0)}$ as its random component. This process incurs additive noise corresponding to the $d/2 - 1$ key-switchings.

The Collapse Procedure. After invoking CollapseHalf on both halves of the random components, the resultant pseudorandom component b_2 can be expressed as $b_2 = -a_1\tilde{s} - a_2\tau_h(\tilde{s}) + \hat{m}_{agg} + e_{tot}$ where e_{tot} is the total accumulated noise from the $d-2$ key-switchings. The final key-switching using $\mathbf{K}_h$ is performed to eliminate the secret key share $\tau_h(\tilde{s})$ in the expression.

Online Computation. We can see that the online execution of each invocation of $\mathsf{KS.KeySwitch}$ involves an inner product between $\mathbf{y}^{k-1}$ and the gadget decomposed $\mathbf{a}^{(k)}$, which costs $O(\ell)$ polynomial multiplications and additions. This in turn implies that there are in total $O(\ell d)$ polynomial multiplications and additions. In NTT space, each polynomial multiplication and addition costs $O(d)$, and so the total online running time of packing is $O(\ell \cdot d^2)$.

4. Proof of Lemma 1

First, we state a useful number theoretic fact from [34].

Lemma 3. *[34] Let d be a power of two and $\gamma < d$ also a power of two. Let $g = 2d/\gamma + 1 \in \mathbb{Z}_{2d}^*$. Then $\mathsf{ord}(g) = \gamma$.*

Lemma 4. *Let d be a power of two and $\gamma < d$ also a power of two. Let $g = 2d/\gamma + 1 \in \mathbb{Z}_{2d}^*$. Then the map $g^i \pmod{2d} \mapsto g^i + g - 1 \pmod{2d}$ is a bijection.*

Proof. First, we see that $g^i \equiv 1 \pmod{g-1}$, which can be proved by induction. By Lemma 3, $\mathsf{ord}(g) = \gamma$. Pigeon hole principle implies that the elements of $\{g^i \pmod{2d} \mid 0 \le i < \gamma\}$ are precisely of the form $g + j(2d/\gamma)$ for $0 \le j < \gamma$. Therefore, the map $g^i \pmod{2d} \mapsto g^i + g - 1 \pmod{2d}$ is a bijection. □

The next lemma is central to the proof of Lemma 1.

Lemma 5. *Let $p(X) = \sum_{j=0}^{d-1} c_j X^j$. Let $\gamma < d$ be a power of two integer and $g = 2d/\gamma + 1$. Then,*

$$\pi_\gamma(p) := \sum_{i=0}^{\gamma-1} \tau_g^i(p) \implies \pi_\gamma(p) = \gamma \sum_{\gamma | i} c_i X^i.$$

Proof. We have $\tau_g^i(p) = \sum_{j=0}^{d-1} c_j X^{jg^i}$ and so

$$\pi_\gamma(p) = \sum_{i=0}^{\gamma-1}\sum_{j=0}^{d-1} c_j X^{jg^i} = \sum_{j=0}^{d-1} c_j \Big(\sum_{i=0}^{\gamma-1} X^{jg^i}\Big).$$

Thus, it suffices to show that $\sum_{i=0}^{\gamma-1} X^{jg^i} = \gamma \cdot X^j$ if $\gamma \mid j$ and 0 otherwise.

Suppose that $\gamma \mid j$. It suffices to show that $X^{jg} = X^j$, which would imply that $X^{jg^i} = X^j$ for all $0 \le i < \gamma$. Now, since $g = 2d/\gamma + 1$, $jg = 2dj/\gamma + j$. But since $\gamma \mid j$, it follows that $2dj/\gamma$ is a multiple of $2d$, which implies that $X^{jg} = X^{2dj/\gamma + j} = X^j$. Therefore, we have $\sum_{i=0}^{\gamma-1} X^{jg^i} = \gamma \cdot X^j$. On the other hand, suppose that $\gamma \nmid j$. We want to show that $\sum_{i=0}^{\gamma-1} X^{jg^i} = 0$. Let $\omega = X^{j(g-1)}$. Then,

$$\omega \sum_{i=0}^{\gamma-1} X^{jg^i} = \sum_{i=0}^{\gamma-1} X^{j(g^i+g-1)}.$$

By Lemma 4, the map $g^i \pmod{2d} \mapsto g^i + g - 1 \pmod{2d}$ is a bijection, which implies that

$$\sum_{i=0}^{\gamma-1} X^{j(g^i+g-1)} = \sum_{i=0}^{\gamma-1} X^{jg^i}.$$

This shows that $(\omega - 1)\sum_{i=0}^{\gamma-1} X^{jg^i} = 0$. It remains to show that $\omega \ne 1$, which would prove that $\sum_{i=0}^{\gamma-1} X^{jg^i} = 0$ because $\mathbb{Z}[X]/(X^d+1)$ is an integral domain. But $\omega = X^{j(g-1)} = X^{2dj/\gamma} = 1$ if and only if $2dj/\gamma$ is a multiple of $2d$ if and only if $\gamma \mid j$. But since $\gamma \nmid j$, it follows that $\omega \ne 1$, completing the proof. □

Finally, we prove the main lemma using the above.

Lemma 1. *Let $p(X) \in \mathbb{Z}[X]/(X^d+1)$ such that $p(X) = \sum_{i=0}^{d-1} c_i X^i$ where d is a power of two. Let $g = 5$ and $h = 2d-1$, and define Tr : $R \to R$ as*

$$Tr(p) := \sum_{j=0}^{d/2-1} \tau_g^j(p) + \tau_h \circ \tau_g^j(p).$$

Then Tr$(p) = d \cdot c_0$.

Proof of Lemma 1. Applying Lemma 5 with $\gamma = d/2$ and $g = 5$ yields $\pi_{d/2}(p) = (d/2)\left(c_0 + c_{d/2}X^{d/2}\right)$. Now

$$\tau_h \circ \pi_{d/2}(p) = \frac{d}{2}\left(c_0 + c_{d/2} \cdot \tau_h(X^{d/2})\right) = \frac{d}{2}\left(c_0 - c_{d/2}X^{d/2}\right)$$

But $\mathrm{Tr} = \pi_{d/2} + \tau_h \circ \pi_{d/2}$ implying $\mathrm{Tr}(p) = d \cdot c_0$. □

5. Analysis of InsPIRe

Lemma 6 (Polynomials with Subgaussian Coefficients [62], adapted)**.** *Let $f(X) = \sum_{j=0}^{d-1} f_j X^j \in R$ where each coefficient f_j is independently sampled from a subgaussian distribution with parameter σ. Let $g(X) = \sum_{k=0}^{d-1} g_k X^k \in R$ be a B-bounded polynomial, e.g. $|g_k| \leq B$. Then $h = fg$ has coefficients h_i that are subgaussian with parameter $\sqrt{d}B\sigma$.*

Lemma 2. *[Theorem 2.19 [62], adapted] Given $\mathsf{RLWE}(m_0)$, let $\mathsf{RGSW}(m_1)$ be a fresh ciphertext encrypting a unit monomial, e.g. $m_1 = \pm X^k$ for some $0 \leq k < d$. Let the error distribution χ be subgaussian with parameter σ_χ, and let ℓ and z be the gadget parameters of the RGSW ciphertext. Under the independence heuristic, the external product $\mathsf{RLWE}(m_0) \boxdot \mathsf{RGSW}(m_1)$ incurs additive noise $e_{ep} \in R_q$ whose coefficients are subgaussian with parameter σ_{ep} and $\sigma_{ep}^2 \leq \ell d z^2 \sigma_\chi^2/2$.*

Proof. Let $\mathsf{RLWE}(m_0) = (a, b)$ where $b = -as + e_{in} + \Delta m_0$. Let $\mathsf{RGSW}(m_1) = [\mathbf{a}, -s\mathbf{a} + \mathbf{e}] + m_1 \cdot \mathbf{G}_{2,z}$ where $\mathbf{a} \in R_q^{2\ell}$ and $\mathbf{e} \in \chi(R_q^{2\ell})$. Computing the external product expression directly, we get:

$$\begin{aligned}
&\mathsf{RLWE}(m_0) \boxdot \mathsf{RGSW}(m_1) \\
&\quad = \mathbf{G}_{2,z}^{-1}([a,b]) \cdot ([\mathbf{a}, -s\mathbf{a} + \mathbf{e}] + m_1 \cdot \mathbf{G}_{2,z}) \\
&\quad = [\mathbf{a} \cdot \mathbf{G}_{2,z}^{-1}([a,b]), -s \cdot \mathbf{a} \cdot \mathbf{G}_{2,z}^{-1}([a,b)] + \mathbf{e} \cdot \mathbf{G}_{2,z}^{-1}([a,b])] \\
&\qquad + [m_1 a, m_1 b] \\
&\quad = [a', -a's + e_{ep}] + [m_1 a, m_1(-as + e_{in} + \Delta m_0)] \\
&\quad = [a' + m_1 a, -(a' + m_1 a)s + m_1 e_{in} + e_{ep} + \Delta m_0 m_1] \\
&\quad = [a'', -a''s + m_1 e_{in} + e_{ep} + \Delta m_0 m_1]
\end{aligned}$$

where we have defined $a' := \mathbf{a} \cdot \mathbf{G}_{2,z}^{-1}([a,b])$ and $e_{ep} := \mathbf{e} \cdot \mathbf{G}_{2,z}^{-1}([a,b])$, and $a'' := a' + m_1 a$. Since m_1 is a unit monomial and $\|m_1\|_\infty = 1$, we have $||m_1 e_{in}||_\infty = ||m_1||_\infty ||e_{in}||_\infty = ||e_{in}||_\infty$. Therefore, the only additional noise from the external product is the additive noise e_{ep}.

We now analyze e_{ep}. $\mathbf{e}$ is a fresh noise vector and thus the coefficients of each polynomial are subgaussian with parameter σ_χ. By the definition of $\mathbf{G}_{2,z}^{-1}$, each element $||\mathbf{G}_{2,z}^{-1}([a,b])[k]||_\infty \leq z/2$. By Lemma 6, each product $\mathbf{e}[k] \cdot \mathbf{G}_{2,z}^{-1}([a,b])[k]$ has coefficients that are subgaussian with parameter $\sqrt{d} z \sigma_\chi/2$. The entire product $e_{ep} = \mathbf{e} \cdot \mathbf{G}_{2,z}^{-1}([a,b])$ involves adding 2ℓ such polynomials. Thus, invoking independence heuristics, e_{ep} has subgaussian coefficients with parameter σ_{ep} satisfying $\sigma_{ep}^2 \leq \ell d z^2 \sigma_\chi^2/2$. □

Theorem 9. *Let the error distribution χ be subgaussian with parameter σ_χ. Let ℓ_{ks} and z_{ks} be the gadget parameters of the key-switching matrices. Let ℓ_{gsw} and z_{gsw} be the gadget parameters of the RGSW ciphertext. The error polynomial of the RLWE ciphertext ct has the form $e = e_{main} + e_{overflow} \in R_q$. Under the independence heuristics, e_{main} has subgaussian coefficients with parameter $\sigma_{main}^2 \leq Np^2\sigma_\chi^2 + t\ell_{ks} d^2 z_{ks}^2 \sigma_\chi^2/4 + t\ell_{gsw} \cdot d z_{gsw}^2 \cdot \sigma_\chi^2/2$ and $||e_{overflow}||_\infty \leq tp/2$.*

Proof. We first analyze the main noise e_{main} of the final ciphertext ct. The first level of the recursion involves multiplying each LWE ciphertext by an element in $\mathbb{Z}_p$ and summing up N/t of them. Thus, the error term of each of the resulting LWE ciphertexts has subgaussian coefficients with parameter σ_1 and $\sigma_1^2 \leq Np^2\sigma_\chi^2$.

By Theorem 2, InspiRING incurs an additive noise e_{pack} whose coefficients are subgaussian with parameter σ_{pack}: $\sigma_{pack}^2 \leq \ell_{ks} d^2 z_{ks}^2 \sigma_\chi^2/4$. Thus, after packing, each of the n RLWE ciphertexts at this point have noise e_2 whose coefficients are subgaussian with parameter $\sigma_2 : \sigma_2^2 = \sigma_1^2 + \sigma_{pack}^2$.

By Lemma 2, each external product incurs an additive noise e_{ep} with subgaussian coefficients with parameter σ_{ep} : $\sigma_{ep}^2 \leq \ell_{gsw} d z_{gsw}^2 \sigma_\chi^2/2$. Since there are $t-1$ external products and $t-1$ additions of the packed ciphertexts, the final ciphertext has main noise e_{main} with parameter σ_{main}:

$$\begin{aligned}
\sigma_{main}^2 &= t\sigma_2^2 + (t-1)\sigma_{ep}^2 \\
&\leq t(\sigma_2^2 + \sigma_{ep}^2) \\
&\leq Np^2\sigma_\chi^2 + t\ell_{ks} d^2 z_{ks}^2 \sigma_\chi^2/4 + t\ell_{gsw} d z_{gsw}^2 \sigma_\chi^2/2
\end{aligned}$$

completing the analysis for the e_{main} term.

We now analyze the additional noise introduced from the plaintext overflow from the n homomorphic additions during the homomorphic polynomial evaluation.

Formally, consider the final ciphertext $\mathsf{ct} = (a, -as + \Delta m + e_{main})$ where e_{main} is the primary noise. Decryption yields $[\Delta m + e_{main}]_q$. An overflowed plaintext m can be written as $m = [m]_p + p \cdot \hat{m}$. Here, $[m]_p$ is the intended message (modulo p), and $p \cdot \hat{m}$ is the overflow, with $||\hat{m}||_\infty \leq n$. The scaling factor is $\Delta = \lfloor q/p \rfloor = q/p + \epsilon$ where $\epsilon \in [-1/2, 1/2)$ is the fractional part. Thus,

$$\begin{aligned}\Delta m + e_{main} &= \Delta[m]_p + (q/p + \epsilon)p \cdot \hat{m} + e_{main} \\ &= \Delta[m]_p + q \cdot \hat{m} + \epsilon p \cdot \hat{m} + e_{main}\end{aligned}$$

Modulo q, the $q \cdot \hat{m}$ term vanishes, leaving $[\Delta[m]_p + \epsilon p \cdot \hat{m} + e_{main}]_q$. The overflow-induced error is $e_{overflow} = \epsilon p \cdot \hat{m}$ with $||e_{overflow}||_\infty \leq tp/2$, completing the proof. □

For the $e_{overflow}$ error to significantly affect decryption, its magnitude upper bound $tp/2$ must be comparable to the noise tolerance, e.g. $tp/2 = \Omega(q/p)$. This implies $q = O(tp^2)$. In practice, however, for a typical choice of RLWE parameters and the interpolation degree t, we usually have $q \gg tp^2$. Therefore, the overflow error $e_{overflow} = \epsilon p \cdot \hat{m}$ is negligible compared to the decryption threshold $\approx q/p$.

Theorem 10. *For subgaussian χ with parameter σ_χ, define $\bar{\sigma}^2_{main} = Np^2\sigma^2_\chi + t\ell_{ks}d^2z^2_{ks}\sigma^2_\chi/4 + t\ell_{gsw} \cdot dz^2_{gsw} \cdot \sigma^2_\chi/2$. Then, for $\delta > 2d\exp\left(-\pi(\Delta/2 - tp/2)^2/\bar{\sigma}^2_{main}\right)$,* InsPIRe *is $(1-\delta)$-correct.*

Proof. By Theorem 9, we know that the final ciphertext has noise $e = e_{main} + e_{overflow}$ for a subgaussian e_{main} with parameter $\sigma_{main} < \bar{\sigma}_{main}$. Hence, we know that for every i, $\mathbb{P}[e_{main}[i] > \Delta/2 - tp] < \delta/d$, so $\mathbb{P}[||e_{main}||_\infty > \Delta/2 - tp] < \delta$, so $||e||_\infty < \Delta/2$ with probability $1-\delta$. So the decryption is correct with $1-\delta$, which proves the theorem. □

Theorem 11. *Let ℓ_{ks} and ℓ_{gsw} be the gadget parameters of the key-switching matrices and the query RGSW ciphertext respectively. In the CRS model,* InsPIRe *runs in offline time $O(Nd^2 + t(d^3 + \ell_{ks}d^2 \lg d))$ and online time $O(Nd + t\ell_{ks}d^2 + t\ell_{gsw}d\lg(d))$.*

Proof. The first level of the recursion incurs $O(Nd)$ multiplications and additions over $\mathbb{Z}_q$. The LWEs to RLWE translation involves invoking InspiRING t times, and by Theorem 1, the total time is $O(t\ell_{ks}d^2)$. Each iteration of the loop in the homomorphic polynomial evaluation can be broken down as follows:

1) Inverse NTT ct back to coefficient form.
2) Gadget decompose ct
3) NTT gadget decomposed ct
4) Matrix-matrix product $\mathbf{AB}$ with $\mathbf{A} \in R_q^{1\times 2\ell_{gsw}}$ and $\mathbf{B} \in R_q^{2\ell_{gsw}\times 2}$
5) Add two RLWE ciphertexts

The first three steps can be done in $O(\ell_{gsw}d\lg(d))$ times. The matrix-matrix product costs $O(\ell_{gsw}d)$ time. The addition of two RLWE ciphertext costs d times. Overall, each iteration of the loop takes $O(\ell_{gsw}d\lg(d))$ time. Therefore, the entire homomorphic polynomial evaluation takes $O(t\ell_{gsw}d\lg(d))$ time. Thus, the entire protocol takes $O(Nd + t\ell_{ks}d^2 + t\ell_{gsw}d\lg(d))$ time. □

Theorem 12. *The total communication cost of* InsPIRe *is*

$$d\ell_{ks}\log_2 q + (N/t)\log_2 q + 4\ell_{gsw}d\log_2 q + 2d\log_2 q$$

Proof. The upload cost consists of the packing keys, the indicator vector, and the RGSW ciphertext. The packing key size is $2d\ell_{ks}\log_2 q$. The indicator vector is of length N/t, so the total size is $(N/t)\log_2 q$. Finally, the RGSW ciphertext is of size $4 \times \ell_{gsw}$. The download cost only consists of one ciphertext, which is of size $2d\log_2 q$. □

6. Security Analysis of InsPIRe

Key-Dependent RLWE Hardness. The security of InsPIRe relies on the hardness of the following variant of the decision RLWE problem, adapted from [63]:

Definition 1 ([63]). *Let λ be a security parameter, $d = d(\lambda)$ a power of two, $m = m(\lambda)$ be the number of samples, $q = q(\lambda)$ a ciphertext modulus, and $\chi(R) = \chi(\lambda, R), \chi(R^m) = \chi(\lambda, R^m)$ be error distributions over the ring $R = \mathbb{Z}[X]/(X^d + 1)$ and vector R^m. Let $\mathcal{F}$ be an efficiently computable set of functions from R_q to R_q. For a bit $\beta \in \{0, 1\}$ and an adversary $\mathcal{A}$, let*

$$\mathcal{W}_\beta := \Pr[\mathcal{A}^{\mathcal{O}(\cdot)}(1^\lambda, \mathbf{a}, \mathbf{t}_\beta) \mid {}^{s \leftarrow \chi(R), \mathbf{a} \leftarrow U(R_q^m), \mathbf{e} \leftarrow \chi(R^m)}_{\mathbf{t}_0 \leftarrow U(R_q^m), \mathbf{t}_1 \leftarrow s\mathbf{a} + \mathbf{e}}]$$

The oracle $\mathcal{O}$ takes as input a function $f \in \mathcal{F}$. If $\beta = 0$, it outputs a uniformly random pair $(x, r) \in R_q \times R_q$. If $\beta = 1$, it outputs $(a, as + e + f(s))$, where $a \leftarrow U(R_q)$ and $e \leftarrow \chi(R)$. We say that the key-dependent RLWE hardness assumption with parameters $(d, m, q, \chi, \mathcal{F})$ holds if for all PPT adversaries $\mathcal{A}$, the advantage $|\mathcal{W}_0 - \mathcal{W}_1|$ is a negligble function in λ.

We require the assumption to hold for a family $\mathcal{F}_{auto}$ of scaled Galois automorphisms:

$$\mathcal{F}_{auto} = \{k \cdot \tau_g \mid k \in \mathbb{Z}_q, g \in \mathbb{N}\}$$

Security Analysis of $\mathsf{InsPIRe}$. We now show that $\mathsf{InsPIRe}$ satisfies the query privacy definition assuming the hardness of the key-dependent RLWE problem.

Theorem 13. *Let λ be the security parameter, $z_{ks}, \ell_{ks}, z_{gsw}, \ell_{gsw}$ be the gadget parameters for* $\mathsf{InspiRING}$ *and the RGSW encryption respectively. Suppose that the key-dependent RLWE hardness assumption holds for $m \geq N/t + 2\ell_{ks} + 2\ell_{gsw}$ with the function family $\mathcal{F}_{auto}$. Then* $\mathsf{InsPIRe}$ *satisfies query privacy.*

Proof. Suppose, for contradiction, that there exists a PPT adversary $\mathcal{A}$ that wins the query privacy game for $\mathsf{InsPIRe}$ with a non-negligible advantage $\mathsf{Adv}^{\mathsf{QP}}_{\mathcal{A}}(\lambda) = \epsilon(\lambda)$. We construct a PPT algorithm $\mathcal{B}$ that breaks the key-dependent RLWE assumption with non-negligible advantage.

Algorithm $\mathcal{B}$ is given 1^λ, a vector $\mathbf{a} \in R_q^m$, a vector $\mathbf{t}_\beta \in R_q^m$ (where $\beta \in \{0,1\}$ is the RLWE challenge bit) and access to an oracle $\mathcal{O}(\cdot)$. $\mathcal{B}$ proceeds as follows:

1) **Adversary's DB Choice:** $\mathcal{B}$ runs $\mathcal{A}$ on input 1^λ. $\mathcal{A}$ outputs a database $\mathbf{D}$.
2) **Simulate Offline Query Components:** To generate $\mathsf{qry_{off}}$ for $\mathcal{A}$, $\mathcal{B}$ uses its RLWE challenge samples and oracle $O(\cdot)$. Recall from $\mathsf{InsPIRe.Setup}$ that $\mathsf{qry_{off}}$ in pp consists of the random matrix $\mathbf{A}$ for the LWE ciphertexts and the random parts of the key-switching matrices $\mathbf{K}_g$ and $\mathbf{K}_h$.
   - $\mathcal{B}$ forms the matrix $\mathbf{A}$ using the first N/t polynomials from its input $\mathbf{a}$.
   - For the offline parts of the key-switching matrices, which are $\mathbf{w}_g$ and $\mathbf{w}_h$: For each $k \in [\ell_{ks}]$, $\mathcal{B}$ queries its oracle $\mathcal{O}(\cdot)$ with $f_g^k = z_{ks}^{k-1} \cdot \tau_g$ and $f_h^k = z_{ks}^{k-1} \cdot \tau_h$. Let the outputs be $(w_k^{(g)}, y_k^{(g)})$ and $(w_k^{(h)}, y_k^{(h)})$. $\mathcal{B}$ sets $\mathbf{w}_g = [w_0^{(g)}, \ldots, w_{\ell_{ks}-1}^{(g)}]$ and $\mathbf{w}_h = [w_0^{(h)}, \ldots, w_{\ell_{ks}-1}^{(h)}]$.

   $\mathcal{B}$ sets $\mathsf{qry_{off}} \leftarrow (\mathbf{A}, \mathbf{w}_g, \mathbf{w}_h)$.
3) **PIR Setup:** $\mathcal{B}$ computes $(\mathsf{pp}, \mathbf{D}') \leftarrow \mathsf{InsPIRe.Setup}(1^\lambda, \mathbf{D})$. It replaces pp's offline query parts with the computed $\mathsf{qry_{off}}$. Let this updated public params be pp'. It gives $(\mathsf{pp}', \mathbf{D}')$ to $\mathcal{A}$.
4) **Adversary's Challenge Indices:** $\mathcal{A}$ outputs a pair of indices $(\mathsf{idx}^{(0)}, \mathsf{idx}^{(1)})$.
5) **$\mathcal{B}$'s Internal Challenge:** $\mathcal{B}$ chooses a random bit $c \in \{0,1\}$. $\mathcal{B}$ will simulate $\mathsf{qry}^{(c)}$ for $\mathcal{A}$. Let $\mathsf{idx}^{(c)} = (i, j)$.
6) **Construct Challenge Query $\mathsf{qry}^{(c)}$:** $\mathcal{B}$ constructs $\mathsf{qry}^{(c)} = (\mathbf{b}, \mathsf{RGSW}(\omega^j), \mathbf{y}_g, \mathbf{y}_h)$ as follows:
   - For the LWE pseudorandom vector $\mathbf{b}$: For $k = 0 \ldots N/t - 1$, $\mathcal{B}$ computes $b_k = \mathsf{Coeff}(\mathbf{t}_\beta[k] + \Delta\mu_k, 0)$ (e.g. constant term of the polynomial), where $\mu_k = 1$ if $i = k$ and $\mu_k = 0$ otherwise. Set $\mathbf{b} = [b_0, \ldots, b_{N/t-1}]^\top \in \mathbb{Z}_q^{N/t}$.
   - For the pseudorandom parts of the key-switching matrices $\mathbf{y}_g$ and $\mathbf{y}_h$: Set $\mathbf{y}_g = [y_0^{(g)}, \ldots, y_{\ell_{ks}-1}^{(g)}]$ and $\mathbf{y}_h = [y_0^{(h)}, \ldots, y_{\ell_{ks}-1}^{(h)}]$, which are constructed from the second components of the oracle query responses.
   - For $\mathsf{RGSW}(\omega^j)$: $\mathcal{B}$ uses the next $2\ell_{gsw}$ samples from its RLWE challenge $(\mathbf{a}, \mathbf{t}_\beta)$. Let these be $(\mathbf{a}_{gsw}, \mathbf{t}_{gsw})$. $\mathcal{B}$ constructs $\mathsf{RGSW}(\omega^j) = [\mathbf{a}_{gsw}, \mathbf{t}_{gsw}] + \mathbf{G}_{2,z_{gsw}} \cdot \omega^j$ where $\omega = X^{2d/t}$.

   $\mathcal{B}$ gives the fully constructed qry to $\mathcal{A}$.
7) **Adversary's Guess:** $\mathcal{A}$ outputs a bit b'.
8) **$\mathcal{B}$'s Output:** Output 1 if $b' = c$. Otherwise, output 0.

Now, we analyze $\mathcal{B}$'s advantage in the key-dependent RLWE game. Let $\mathcal{W}_\beta = \Pr[\mathcal{B}$ outputs $1 \mid$ RLWE challenge bit is $\beta]$.

- **Case 1:** $\beta = 0$ **(RLWE samples are random)**: In this case, we see that the outputs in $\mathsf{qry_{on}}$ are uniformly random and independent of the query index idx_c. Therefore, the adversary cannot do better than a random guessing, and thus:

$$\mathcal{W}_0 = \Pr[b' = c \mid \beta = 0] = 1/2$$

- **Case 2:** $\beta = 1$ **(RLWE samples are real)**: In this case, $\mathcal{B}$ perfectly simulates the query privacy game for $\mathcal{A}$. The components $\mathbf{t}_\beta[k]$ used for constructing $\mathbf{b}$ generates identically distributed pseudorandom components of the LWE ciphertexts. The outputs from the oracle to generate RLWE encryptions of the automorphic images of the secret keys are identically distributed as the outputs from $\mathsf{KS.Setup}$. The RGSW ciphertext is also identically distributed since $\mathcal{B}$ perfectly simulates the RGSW encryption.
  Thus, $\mathcal{A}$'s view is identical to a real query privacy experiment. $\mathcal{A}$'s advantage is $\mathsf{Adv}^{\mathsf{QP}}_{\mathcal{A}}(\lambda) = |\Pr[b' = c] - 1/2| = \epsilon(\lambda)$. Therefore, $\Pr[b' = c \mid \beta = 1] = 1/2 \pm \epsilon(\lambda)$. $\mathcal{B}$ outputs 1 if $b' = c$, and so

$$\mathcal{W}_1 = \Pr[b' = c \mid \beta = 1] = 1/2 \pm \epsilon(\lambda))$$

Therefore, the advantage of $\mathcal{B}$ in breaking the key-dependent RLWE hardness assumption is:

$$|\mathcal{W}_0 - \mathcal{W}_1| = |1/2 - (1/2 \pm \epsilon(\lambda))| = \epsilon(\lambda)$$

contradicting the hardness of the key-dependent RLWE problem. Therefore, $\mathsf{InsPIRe}$ satisfies query privacy. □

6.1. Extension to Multiple Queries. As in prior works, the security of $\mathsf{InsPIRe}$ extend to multiple queries with common random components as long as a new secret key is sampled for each query.

We can modify the single query privacy experiment so that the adversary can adaptively make oracle queries for encryptions of one of the query indices, up to Q oracle queries.

1) **Setup Phase:**
   a) On input security parameter 1^λ, $\mathcal{A}$ outputs a database $\mathbf{D}$.
   b) The challenger computes $(\mathsf{pp}, \mathbf{D}') \leftarrow \mathsf{Setup}(1^\lambda, \mathbf{D})$ and gives it to $\mathcal{A}$.
2) **Challenge Phase:**
   a) The challenger chooses a bit $b \in \{0, 1\}$ uniformly at random.
   b) For $i = 0 \ldots Q - 1$
      i) $\mathcal{A}$ outputs a pair of challenge indices $(\mathsf{idx}_i^{(0)}, \mathsf{idx}_i^{(1)})$.
      ii) The challenger computes the challenge query $(\mathsf{st}_i^{(b)}, \mathsf{qry}_i^{(b)}) \leftarrow \mathsf{Query}(\mathsf{pp}, \mathsf{idx}_i^{(b)})$. The challenger sends $\mathsf{qry}_i^{(b)}$ to $\mathcal{A}$.
3) **Guess Phase:** $\mathcal{A}$ outputs a bit b'.

Proof Sketch. The proof proceeds via a standard hybrid argument. We define a sequence of $Q + 1$ hybrid experiments, $\mathcal{H}_k$ for $k = 0 \ldots Q$.

In experiment $\mathcal{H}_k$, the first k challenge queries are "real" (e.g. correctly generated) while the remaining $Q - k$ queries are random. Observe that $\mathcal{H}_Q$ is the real query privacy experiment and $\mathcal{H}_0$ is the experiment where the adversary receives completely random values.

The core of the proof is to show that for any k, the experiments $\mathcal{H}_{k-1}$ and $\mathcal{H}_k$ are computationally indistinguishable. Otherwise, we can construct an algorithm $\mathcal{B}$ that can break the key-dependent RLWE assumption.

The reduction at a high level works as follows. For the k-th query, $\mathcal{B}$ generates the query using the RLWE challenge as in the single query privacy reduction. For the first $k - 1$ queries, $\mathcal{B}$ generates real challenge queries as in $\mathcal{H}_Q$, using the same random components as in the k-th query to generate the encryptions (with a freshly sampled secret key for each query). For the remaining $Q - k$ queries, $\mathcal{B}$ generates random queries as in $\mathcal{H}_0$.

Note that if $\mathcal{B}$ received real RLWE samples, it perfectly simulates $\mathcal{H}_k$, and otherwise, it perfectly simulates $\mathcal{H}_{k-1}$. In particular, any non-negligible advantage $\mathcal{A}$ has in distinguishing $\mathcal{H}_{k-1}$ and $\mathcal{H}_k$ implies $\mathcal{B}$ can break the key-dependent RLWE hardness assumption.

By the triangle inequality, we can thus conclude that any non-negligible advantage $\mathcal{A}$ has in distinguishing between $\mathcal{H}_0$ and $\mathcal{H}_Q$ implies that the key-dependent RLWE hardness assumption is broken. This completes the proof.

□

7. Optimizations and Extensions to $\mathsf{InsPIRe}$

7.1. Approximate Gadget Decomposition. We note that our construction can use the standard approximate gadget decomposition technique to further reduce the query size and the server computation. This was first introduced in the context of TFHE bootstrapping [21], but the same technique can be applied to the BFV ciphertexts that we use in our work.

We provide a high level overview of this technique in the context of key-switching, but the same principle applies for the RGSW ciphertexts. Consider a RLWE ciphertext $(\tilde{a}, \tilde{b} = -\tilde{a}\tilde{s} + \tilde{e} + \Delta\tilde{m})$. Let $\tilde{a}_{low}$ consist of the least significant digits of the coefficients of $\tilde{a}$ in base z (e.g. $\|a_{low}\|_\infty \leq z$), and $\tilde{a}_{high} = \tilde{a} - \tilde{a}_{low}$. Then, we can express $\tilde{b}$ as:

$$\begin{aligned}\tilde{b} &= -\tilde{a}\tilde{s} + \tilde{e} + \Delta\tilde{m} \pmod{q}\\ &= -(\tilde{a}_{low} + \tilde{a}_{high})\tilde{s} + \tilde{e} + \Delta\tilde{m} \pmod{q}\\ &= -\tilde{a}_{high}\tilde{s} + \tilde{e} - \tilde{a}_{low}\tilde{s} + \Delta\tilde{m} \pmod{q}\end{aligned}$$

In particular, $(\tilde{a}_{high}, \tilde{b})$ is a valid RLWE ciphertext where the term $-\tilde{a}_{low}\tilde{s}$ is now part of the noise. If z and the secret key $\tilde{s}$ have sufficiently low norm (e.g. $\tilde{s}$ from ternary or error distribution), then this additive noise term typically has minimal impact on decryption. Now, suppose that we want to key-switch this ciphertext to be encrypted under a different secret key $\tilde{s}'$. Because the least significant digits (in base z) of $\tilde{a}_{high}$ are zero, the key-switching matrix component corresponding to the least significant digit is not necessary. In particular, instead of the key-switching matrix consisting of ℓ components, it now only requires $\ell - 1$ components. This technique is particularly effective when the number of components ℓ in the standard gadget decomposition is small (as in our case). This key-switching strategy can be applied to Stage 3 of the $\mathsf{InspiRING}$ algorithm, and the general idea can also be applied to the homomorphic polynomial evaluation part as well.

We expect this technique to reduce the size of the key-switching matrices and the RGSW encrypted evaluation points by around 33% while reducing the total computation up to $\approx 25\%$.

7.2. Multivariate. The main limitation of the original InsPIRe protocol is the constraint $t \leq 2d$, which is necessary to ensure that the evaluation points are unit monomials (of the form $\pm X^k$). We can overcome this limitation by employing a multivariate polynomial interpolation/evaluation instead of univariate, at the cost of small additional query overhead.

Let α be the number of variables. For simplicity, we will assume that $t = \hat{t}^\alpha$ for some power of two $\hat{t} \leq 2d$. Each database entry is indexed by a tuple $(i_0, \ldots, i_{\alpha-1})$ and denoted $y_{i_0,\ldots,i_{\alpha-1}}$, where each $0 \leq i_k < \hat{t}$.

As in the univariate approach, we first define $\hat{t} - th$ primitive root of unity $\omega = X^{2d/\hat{t}}$. We then construct a α-variate interpolation polynomial $h(Z_0, \ldots, Z_{\alpha-1}) \in R_p[Z_0, \ldots, Z_{\alpha-1}]$ such that $h(\omega^{i_0}, \ldots, \omega^{i_{\alpha-1}}) = y_{i_0,\ldots,i_{\alpha-1}}$ for each tuple $(i_0, \ldots, i_{\alpha-1})$. Note that this can be accomplished via the multidimensional FFT-based interpolation. h is constructed in a way that the partial degree of the polynomial for each variable is $\hat{t} - 1$.

The homomorphic evaluation of h leverages the recursive structure of the multivariate polynomial rings. The ring $R_p[Z_0, \ldots, Z_{\alpha-1}]$ can be viewed as a ring of univarate polynomials in the variable $Z_{\alpha-1}$ with coefficients in $R_p[Z_0, \ldots, Z_{\alpha-2}]$.

In particular, h can be expressed as

$$h(Z_0, \ldots, Z_{\alpha-1}) = \sum_{k=0}^{\hat{t}-1} c_k(Z_0, \ldots, Z_{\alpha-2}) \cdot Z_{\alpha-1}^k$$

where each coefficient c_k is a polynomial in $\alpha - 1$ variables. This decomposition can be applied recursively to each c_k. In particular, this structure naturally maps to a complete $\hat{t}$-ary tree of depth α. Each internal node represents a univariate polynomial in some variable Z_j, and its children are the coefficient polynomials.

The homomorphic evaluation of h proceeds via a depth-first traversal of this tree structure. At each node, the corresponding univariate polynomial is evaluated using the Horner's method, with the required coefficients recursively evaluated. This construction requires α RGSW ciphertexts in the query, and costs $O(t)$ RLWE-RGSW external products and RLWE additions, same as in the univariate construction.

7.3. Extension to non-power of two t. Cooley-Tukey can only be used when t is a power of two, but the runtime is only $O(t \log t)$ ring operations. For cases where t is not a power of two, we can use Lagrange interpolation, which runs using $O(t^2)$ ring operations. The interpolation happens over R_p with the evaluation points defined as $z_i = X^i$. However, since R_p is not a field, we require that there exist an inverse for $z_i - z_j$ in R_p. For this, we rely on the following lemmas.

Lemma 7. *Consider the plaintext ring $R_p = \mathbb{Z}_p[X]/(X^d + 1)$ where p is odd and d is a power of two. Let $X^k \in R_p$ and $0 \leq k < 2d$. Let $g = \gcd(k, 2d)$ Then*

$$(X^k - 1)^{-1} = (p-2)^{-1} \sum_{i=0}^{d/g-1} X^{ik}$$

Proof. We need to prove that $(X^k - 1)(p-2)^{-1} \sum_{i=0}^{d/g-1} X^{ik} = 1$. We have

$$\begin{aligned}(X^k - 1) \sum_{i=0}^{d/g-1} X^{ik} &= \sum_{i=0}^{d/g-1} X^{(i+1)k} - X^{ik} \\ &= X^{dk/g} - 1 \\ &= (X^d)^{k/g} - 1 \\ &= (-1)^{k/g} - 1 = -2\end{aligned}$$

where the last equality follows because k/g is odd. Since $-2 = p - 2$, multiplying by $(p-2)^{-1}$ (which exists because $\gcd(p, p-2) = 1$) completes the proof. □

Lemma 8. *Suppose that $0 \leq a, b < 2d$ and $a \neq b$. Then*

$$(X^a - X^b)^{-1} = \mathsf{sign}(b-a) \cdot X^{d-\min(a,b)} \cdot (p-2)^{-1} \sum_{i=0}^{d/g-1} X^{i \cdot |a-b|}$$

Proof. We have $(X^a - X^b) = -\mathsf{sign}(b-a) \cdot X^{\min(a,b)}(X^{|a-b|} - 1)$. It is easy to see that $(X^{\min(a,b)})^{-1} = -X^{d-\min(a,b)}$, and applying Lemma 7 to $X^{|a-b|} - 1$ completes the proof. □

Using these two lemmas, we can prove the following theorem about interpolating a polynomial over points in R_p^2.

Theorem 14. *Consider t points $\{(z_i, y_i)\}_{i \in [t]} \in R_p^2$ where $t \leq 2d$ and $z_i = X^i$. Then these points are interpolable over the ring R_p, i.e., there exists $P(Z) \in R_p[Z]$ such that $P(z_i) = y_i$.*

Proof. The i-th Lagrangian basis is defined by

$$L_i(Z) = \prod_{0 \le j < t, j \ne i} \frac{Z - z_j}{z_i - z_j}$$

and by Lemma 8 $(z_i - z_j)^{-1} = (X^i - X^j)^{-1}$ exists. Note, $L_i(z_i) = 1$ and $L_i(z_j) = 0$ for $j \ne i$. So the interpolated polynomial can be computed as

$$P(Z) = \sum_{i=0}^{n-1} y_i \cdot L_i(Z)$$

and it is easy to check that $P(z_i) = y_i$. □

Algorithm 11 shows the pseudocode for Lagrange interpolation. We remark that the same multivariate polynomial evaluation strategy can be used for the multivariate Lagrange interpolated database.

Algorithm 11 Lagrange Interpolation

```
procedure INTERPOLATE($[y_0, \dots, y_{t-1}]^\top \in R_p^t$)
    for $i = 0 \dots t-1$ do
        $z[i] \leftarrow y_i$
    for $j = 1, \cdots, t-1$ do
        for $i = t-1, \cdots, j$ do
            $z[i] \leftarrow (z[i] - z[i-1]) \cdot (X^i - X^{i-j})^{-1}$
    for $i = 0 \dots t-1$ do
        $\mathbf{c}[i] \leftarrow 0$
    $\mathbf{c}[t-1] \leftarrow z[n-1]$
    for $k = t-2, \cdots, 0$ do
        for $i = k, \cdots, 1$ do
            $\mathbf{c}[i] \leftarrow X^k(\mathbf{c}[i] + \mathbf{c}[i-1])$
        $\mathbf{c}[0] \leftarrow X^k \mathbf{c}[0] + z[k]$
    return $\mathbf{c}$
```

8. Analysis of $\mathsf{InsPIRe}^{(2)}$

Lemma 9 (Modulus Switching [62], Theorem 3.4, adapted)**.** . *Let $q \ge \tilde{q} > p$ and let $R = \mathbb{Z}[x]/(x^d + 1)$ where d is a power of two. Suppose $c = (c_1, c_2)$ and $c_1 s + c_2 = \lfloor q/p \rfloor \mu + e \pmod{q}$ for some $s \in R_q$, $c \in R_q^2$, $\|\mu\|_\infty \le p/2$ and $e \in R$. Suppose the components of s are independent subgaussian random variables with parameter σ_s and e is subgaussian with parameter σ_e. Let $(\tilde{c}_1, \tilde{c}_2) = (\lfloor \tilde{q} c_1 / q \rceil, \lfloor \tilde{q} c_2 / q \rceil)$. Then $\tilde{c}_1 s + \tilde{c}_2 = \lfloor \tilde{q}/p \rfloor \mu + \tilde{e} \pmod{\tilde{q}}$ where $\tilde{e} = e_1 + e_2$,*

$$\|e_1\|_\infty \le \frac{1}{2}\left(2 + (\tilde{q} \mod p) + \frac{\tilde{q}}{q}(q \mod p)\right),$$

and the components of e_2 are subgaussian with parameter $\sigma_2^2 = d\sigma_s^2/4 + (\tilde{q}/q)^2 \sigma_e^2$.

Lemma 9 also holds in the case where only a subset of the coefficients of e follow a subgaussian distribution. In that case, the claim of the lemma holds for the same subset of coefficients in e_2.

Theorem 7. *Let the error distribution χ be subgaussian with parameter σ_χ. For $i \in \{0, 1, 2\}$, let e_i denote the error term of an RLWE ciphertext in $\mathsf{resp_i}$, defined as in Algorithm 5. Moreover, let $e[:\gamma]$ denote a polynomial which only consists of the first γ coefficients of e. Then for $i \in \{0, 1, 2\}$, under the independence heuristic, $\tilde{e}_i[:\gamma_i] = \tilde{e}_i^{(1)} + \tilde{e}_i^{(2)}$ such that*

$$||e_i^{(1)}||_\infty \le f(q, \tilde{q}, p) = \frac{1}{2}(2 + (\tilde{q} \mod p) + \frac{\tilde{q}}{q}(q \mod p))$$

and $\tilde{e}_i^{(2)}$ has subgaussian coefficients with parameter $\tilde{\sigma}_i$ where

$$\tilde{\sigma}_0^2 \le \Sigma_0 = d\sigma_\chi^2/4 + (\tilde{q}/q)^2((N/t)p^2\sigma_\chi^2 + \ell_{ks}\gamma_0 d z_{ks}^2 \sigma_\chi^2/4) \quad (3)$$

$$\tilde{\sigma}_1^2 \le \Sigma_1 = d\sigma_\chi^2/4 + (\tilde{q}/q)^2(tp^2\sigma_\chi^2 + \ell_{ks}\gamma_1 d z_{ks}^2 \sigma_\chi^2/4) \quad (4)$$

$$\tilde{\sigma}_2^2 \le \Sigma_2 = d\sigma_\chi^2/4 + (\tilde{q}/q)^2(tp^2\sigma_\chi^2 + \ell_{ks}\gamma_2 d z_{ks}^2 \sigma_\chi^2/4) \quad (5)$$

Proof. We prove the theorem for $\mathsf{resp_0}$ and proof for the other two follows similarly.

The first level of the recursion involves multiplying each LWE ciphertext by an element in $\mathbb{Z}_p$ and summing up N/t of them. Thus, the error term of each of the resulting LWE ciphertexts is subgaussian with parameter $\sigma_{i.p.}$ and $\sigma_{i.p.}^2 \le$

$(N/t)p^2\sigma_\chi^2$. By Theorem 4, PartialInspiRING incurs an additive noise e_{pack} such that $e_{pack}[:\gamma_0]$ has subgaussian coefficients with parameter σ_{pack} where $\sigma_{pack}^2 \leq \ell_{ks}\gamma_0 d z_{ks}^2 \sigma_\chi^2/4$. Thus, after packing, each of the t RLWE ciphertexts have noise e_0 such that the coefficients of $e_0[:\gamma_0]$ are subgaussian with parameter σ_0 where $\sigma_0^2 = \sigma_{i.p.}^2 + \sigma_{pack}^2 \leq (N/t)p^2\sigma_\chi^2 + \ell_{ks}\gamma_0 d z_{ks}^2 \sigma_\chi^2/4$. Finally, after performing modulus switching, based on Lemma 9, we obtain a final error $\tilde{e}_0$ which follows the theorem.

The same rationale holds for $\tilde{e}_1$ and $\tilde{e}_2$, but with inner product of length t and packing parameters γ_1 and γ_2, respectively. □

Theorem 8. *For $f(q, \tilde{q}, p)$ and Σ_i defined as in Theorem 7, then if*

$$\delta_0 \geq 2t\gamma_0 \exp\left(-\pi(\tilde{q}/2p - f(q,\tilde{q},p))^2/\Sigma_0^2\right)$$
$$\delta_1 \geq 2\tau d \exp\left(-\pi(\tilde{q}/2p - f(q,\tilde{q},p))^2/\Sigma_1^2\right)$$
$$\delta_2 \geq 2\tau\gamma_0 \exp\left(-\pi(\tilde{q}/2p - f(q,\tilde{q},p))^2/\Sigma_2^2\right)$$

then InsPIRe$^{(2)}$ *is $(1-\delta_0-\delta_1-\delta_2)$-correct.*

Proof. Failure can occur if the noise over flows in the three steps outlines below as three events.

- E_0: Failure in the first level of PIR + packing using γ_0
- E_1: Failure in the second level of PIR + packing using γ_1
- E_2: Failure in second level of PIR + packing using γ_2

The total failure can be bounded as $\mathbb{P}[E] = \mathbb{P}[E_0 \vee E_1 \vee E_2] \leq \mathbb{P}[E_0] + \mathbb{P}[E_1] + \mathbb{P}[E_2]$. In each case,

$$\begin{aligned}\mathbb{P}[E_0] &= t\gamma_0\mathbb{P}[|\tilde{e}_0| > \tilde{q}/2p] \\ &\leq t\gamma_0\mathbb{P}[|e_0| > \tilde{q}/2p - f(q,\tilde{q},p)] \\ &\leq 2t\gamma_0 \exp\left(-\pi(\tilde{q}/2p - f(q,\tilde{q},p))^2/\Sigma_0^2\right) \leq \delta_0\end{aligned}$$

Similarly,

$$\mathbb{P}[E_1] \leq 2\tau d \exp\left(-\pi(\tilde{q}/2p - f(q,\tilde{q},p))^2/\Sigma_1^2\right) \leq \delta_1$$
$$\mathbb{P}[E_2] \leq 2\tau\gamma_0 \exp\left(-\pi(\tilde{q}/2p - f(q,\tilde{q},p))^2/\Sigma_2^2\right) \leq \delta_2$$

Hence, the total failure probability is bounded by $\delta_0 + \delta_1 + \delta_2$. □

Theorem 5. *Let ℓ_{ks} be the gadget parameter of the key-switching matrices. In the CRS model,* InsPIRe$^{(2)}$ *runs in offline time*

$$O(N\gamma_0 d + t(\gamma_0^2 d + \gamma_0\ell_{ks} d \lg d) + \tau t d^2 + \tau d(\gamma_1 d + \ell_{ks} d \lg d))$$

and online time

$$O(N\gamma_0 + t\gamma_0 d\ell_{ks} + t\tau\gamma_0 + \tau d^2\ell_{ks} + \tau\gamma_0 d(\gamma_2 + \ell_{ks}\lg d)).$$

Proof. The runtime consists of five major parts, for which we provide the offline and online runtime.

- The first matrix multiplication runs in offline time $O(Nd\gamma_0)$ and online time $O(N\gamma_0)$.
- The first packing takes advantage of the offline preprocessing and packs t batches of γ_0 ciphertexts, so it runs in offline time $O(t(\gamma_0^2 d + \gamma_0\ell_{ks} d \lg d))$ and online time $O(t\gamma_0 d\ell_{ks})$.
- The second matrix multiplication runs in offline time $O(\tau t d^2)$ and online time $O(\tau t\gamma_0)$.
- The second packing also takes advantage of offline preprocessing, packing $\frac{\tau d}{\gamma_1}$ batches of γ_1 ciphertexts. Hence, this step runs in offline time $O(\tau d(\gamma_1 d + \ell_{ks} d \lg d))$ and online time $O(\tau d^2\ell_{ks})$.
- The third packing, which packs $\frac{\tau\gamma_0}{\gamma_2}$ batches of γ_2 ciphertexts, can not use preprocessing since the first component is not known before hand. So all operations occur online and the online time is $O((\tau\gamma_0/\gamma_2)(\gamma_2^2 d + \gamma_2 d\ell_{ks})) = O(\tau\gamma_0 d(\gamma_2 + \ell_{ks}\lg d))$.

□

9. Additional Experiments

We provide additional experiments for benchmarking of InsPIRe$^{(2)}$ and InsPIRe.We provide the breakdown for PIR over a 1 GB, 8 GB, and 32 GB database using InsPIRe$^{(2)}$ and InsPIRe.

TABLE 8: Performance metrics for different configurations of InsPIRe$^{(2)}$

1 GB Configuration

	$\gamma_0 = 16, \gamma_1 = 1024, \gamma_2 = 16$				$\gamma_0 = 32, \gamma_1 = 1024, \gamma_2 = 32$				$\gamma_0 = 64, \gamma_1 = 1024, \gamma_2 = 64$			
Entry Size	32 B				64 B				128 B			
N/t	8192	16384	32768	65536	4096	8192	16384	32768	4096	8192	16384	32768
Upload Keys	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB
Upload Query	80 KB	119 KB	219 KB	427 KB	53 KB	66 KB	113 KB	215 KB	40 KB	60 KB	109 KB	214 KB
Download	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB
Total Comm.	211 KB	251 KB	350 KB	559 KB	185 KB	198 KB	244 KB	347 KB	172 KB	191 KB	241 KB	345 KB
First Mat. Mul	104 ms	104 ms	104 ms	102 ms	102 ms	104 ms	101 ms	102 ms	104 ms	103 ms	102 ms	102 ms
First Pack	549 ms	277 ms	140 ms	67 ms	966 ms	487 ms	224 ms	112 ms	852 ms	429 ms	211 ms	107 ms
Second Mat. Mul	3.7 ms	1.8 ms	1.0 ms	0.5 ms	3.3 ms	1.9 ms	1.2 ms	0.8 ms	2.0 ms	1.1 ms	0.6 ms	0.4 ms
Second Pack	28 ms	29 ms	29 ms	28 ms	29 ms	29 ms	28 ms	28 ms	29 ms	29 ms	28 ms	29 ms
Third Pack	10 ms	9 ms	9 ms	9 ms	24 ms	21 ms	21 ms	21 ms	57 ms	58 ms	55 ms	59 ms
Total Time	750 ms	470 ms	330 ms	260 ms	1180 ms	690 ms	440 ms	320 ms	1100 ms	670 ms	470 ms	360 ms

8 GB Configuration

	$\gamma_0 = 16, \gamma_1 = 1024, \gamma_2 = 16$				$\gamma_0 = 32, \gamma_1 = 1024, \gamma_2 = 32$				$\gamma_0 = 64, \gamma_1 = 1024, \gamma_2 = 64$			
Entry Size	32 B				64 B				128 B			
N/t	16384	32768	65536	131072	16384	32768	65536	131072	8192	16384	32768	65536
Upload Keys	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB
Upload Query	212 KB	265 KB	450 KB	861 KB	159 KB	238 KB	437 KB	855 KB	106 KB	132 KB	225 KB	431 KB
Download	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB
Total Comm.	344 KB	397 KB	582 KB	993 KB	291 KB	370 KB	569 KB	986 KB	238 KB	264 KB	357 KB	562 KB
First Mat. Mul	844 ms	819 ms	911 ms	836 ms	827 ms	845 ms	827 ms	824 ms	849 ms	840 ms	823 ms	814 ms
First Pack	2221 ms	1038 ms	538 ms	289 ms	1837 ms	994 ms	480 ms	225 ms	3462 ms	1681 ms	875 ms	424 ms
Second Mat. Mul	13.7 ms	7.1 ms	3.9 ms	1.7 ms	7.0 ms	4.1 ms	2.0 ms	1.1 ms	6.4 ms	3.9 ms	1.7 ms	1.1 ms
Second Pack	34 ms	29 ms	30 ms	30 ms	30 ms	28 ms	30 ms	28 ms	34 ms	28 ms	28 ms	28 ms
Third Pack	10 ms	10 ms	11 ms	9 ms	23 ms	22 ms	20 ms	20 ms	63 ms	59 ms	56 ms	56 ms
Total Time	3150 ms	1960 ms	1550 ms	1210 ms	2850 ms	1930 ms	1420 ms	1150 ms	4680 ms	2670 ms	1850 ms	1390 ms

32 GB Configuration

	$\gamma_0 = 16, \gamma_1 = 1024, \gamma_2 = 16$				$\gamma_0 = 32, \gamma_1 = 1024, \gamma_2 = 32$				$\gamma_0 = 64, \gamma_1 = 1024, \gamma_2 = 64$			
Entry Size	32 B				64 B				128 B			
N/t	32768	65536	131072	262144	32768	65536	131072	262144	32768	65536	131072	
Upload Keys	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	80 KB	
Upload Query	424 KB	530 KB	901 KB	1722 KB	318 KB	477 KB	874 KB	1709 KB	265 KB	450 KB	861 KB	
Download	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	52 KB	
Total Comm.	556 KB	662 KB	1033 KB	1854 KB	450 KB	609 KB	1006 KB	1841 KB	397 KB	582 KB	993 KB	
First Mat. Mul	4336 ms	3308 ms	3430 ms	3406 ms	3419 ms	3279 ms	3325 ms	3434 ms	3369 ms	3313 ms	3391 ms	
First Pack	8404 ms	2049 ms	1055 ms	555 ms	4131 ms	1855 ms	935 ms	494 ms	3778 ms	1685 ms	870 ms	
Second Mat. Mul	37.9 ms	13.7 ms	7.1 ms	3.3 ms	13.3 ms	7.0 ms	3.9 ms	2.0 ms	6.8 ms	3.6 ms	1.9 ms	
Second Pack	28 ms	103 ms	27 ms	29 ms	36 ms	32 ms	29 ms	30 ms	36 ms	31 ms	30 ms	
Third Pack	11 ms	10 ms	10 ms	10 ms	24 ms	24 ms	23 ms	22 ms	62 ms	61 ms	57 ms	
Total Time	11780 ms	5540 ms	4560 ms	4070 ms	7760 ms	5260 ms	4410 ms	4010 ms	7390 ms	5280 ms	4430 ms	

TABLE 9: Performance metrics for InsPIRe using different polynomial interpolation degrees.

Database Size: 1 GB																		
	1 bit Entry					64 B Entry							32 KB Entry					
N/t	4096	8192	16384	32768	65536	4096	8192	16384	32768	65536			1024	2048	4096	8192	16384	32768
Interpolation Degree	32	32	16	8	4	32	32	16	8	4			32	16	8	4	2	1
Offline Time	113 s	87 s	79 s	99 s	96 s	110 s	86 s	77 s	98 s	99 s			305 s	177 s	106 s	83 s	72 s	86 s
Upload Keys	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB			84 KB	84 KB	84 KB	84 KB	84 KB	84 KB
Upload Query	112 KB	140 KB	196 KB	308 KB	532 KB	112 KB	140 KB	196 KB	308 KB	532 KB			91 KB	98 KB	112 KB	140 KB	196 KB	308 KB
Download	24 KB	12 KB	12 KB	12 KB	12 KB	24 KB	12 KB	12 KB	12 KB	12 KB			96 KB	96 KB	96 KB	96 KB	96 KB	96 KB
Total Comm.	220 KB	236 KB	292 KB	404 KB	628 KB	220 KB	236 KB	292 KB	404 KB	628 KB			271 KB	278 KB	292 KB	320 KB	376 KB	488 KB
Mat Mul.	100 ms	100 ms	100 ms	110 ms	100 ms	100 ms	100 ms	100 ms	100 ms	100 ms			110 ms	110 ms	100 ms	100 ms	100 ms	100 ms
Packing	1070 ms	540 ms	320 ms	180 ms	110 ms	1080 ms	570 ms	310 ms	180 ms	110 ms			4220 ms	2030 ms	1050 ms	540 ms	300 ms	180 ms
Poly Eval	14 ms	7 ms	4 ms	2 ms	1 ms	15 ms	7 ms	4 ms	2 ms	1 ms			59 ms	29 ms	14 ms	7 ms	3 ms	2 ms
Total Time	1180 ms	650 ms	420 ms	280 ms	210 ms	1180 ms	660 ms	400 ms	280 ms	210 ms			4390 ms	2210 ms	1170 ms	640 ms	410 ms	280 ms

Database Size: 8 GB																		
	1 bit Entry					64 B Entry							32 KB Entry					
N/t	8192	16384	32768	65536	131072	8192	16384	32768	65536	131072	262144	524288	8192	16384	32768	65536	131072	262144
Interpolation Degree	32	32	32	32	16	32	32	32	32	16	8	4	32	16	8	4	2	1
Offline Time	547 s	470 s	596 s	564 s	579 s	544 s	476 s	601 s	581 s	566 s	556 s	560 s	547 s	470 s	598 s	597 s	538 s	467 s
Upload Keys	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB
Upload Query	140 KB	196 KB	308 KB	532 KB	980 KB	140 KB	196 KB	308 KB	532 KB	980 KB	1876 KB	3668 KB	140 KB	196 KB	308 KB	532 KB	980 KB	1876 KB
Download	96 KB	48 KB	24 KB	12 KB	12 KB	96 KB	48 KB	24 KB	12 KB	12 KB	12 KB	12 KB	96 KB	96 KB	96 KB	96 KB	96 KB	96 KB
Total Comm.	320 KB	328 KB	416 KB	628 KB	1076 KB	320 KB	328 KB	416 KB	628 KB	1076 KB	1972 KB	3764 KB	320 KB	376 KB	488 KB	712 KB	1160 KB	2056 KB
Mat Mul.	850 ms	820 ms	800 ms	840 ms	820 ms	810 ms	830 ms	810 ms	810 ms	810 ms	910 ms	830 ms	810 ms	810 ms	810 ms	810 ms	830 ms	810 ms
Packing	4040 ms	2080 ms	1220 ms	560 ms	320 ms	4050 ms	2090 ms	1110 ms	540 ms	310 ms	190 ms	180 ms	3970 ms	2050 ms	1270 ms	540 ms	290 ms	190 ms
Poly Eval	59 ms	30 ms	15 ms	7 ms	4 ms	59 ms	29 ms	15 ms	8 ms	4 ms	2 ms	1 ms	59 ms	30 ms	15 ms	7 ms	4 ms	2 ms
Total Time	5220 ms	3000 ms	1890 ms	1400 ms	1120 ms	5530 ms	3050 ms	1890 ms	1360 ms	1130 ms	1120 ms	970 ms	5630 ms	3010 ms	1920 ms	1340 ms	1120 ms	1020 ms

Database Size: 32 GB																		
	1 bit Entry					64 B Entry						32 KB Entry						
N/t	32768	65536	131072	262144	1048576	32768	65536	131072	262144	524288	1048576	16384	32768	65536	131072	262144	524288	1048576
Interpolation Degree	32	32	32	32	8	32	32	32	32	16	8	32	32	16	8	4	2	1
Offline Time	2539 s	2393 s	2406 s	2115 s	2111 s	2581 s	2250 s	2420 s	2128 s	2114 s	2178 s	2052 s	2486 s	2569 s	2306 s	2340 s	2053 s	1945 s
Upload Keys	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB	84 KB
Upload Query	308 KB	532 KB	980 KB	1876 KB	7252 KB	308 KB	532 KB	980 KB	1876 KB	3668 KB	7252 KB	196 KB	308 KB	532 KB	980 KB	1876 KB	3668 KB	7252 KB
Download	96 KB	48 KB	24 KB	12 KB	12 KB	96 KB	48 KB	24 KB	12 KB	12 KB	12 KB	192 KB	96 KB	96 KB	96 KB	96 KB	96 KB	96 KB
Total Comm.	488 KB	664 KB	1088 KB	1972 KB	7348 KB	488 KB	664 KB	1088 KB	1972 KB	3764 KB	7348 KB	472 KB	488 KB	712 KB	1160 KB	2056 KB	3848 KB	7432 KB
Mat Mul.	3200 ms	4300 ms	3400 ms	3500 ms	3300 ms	3300 ms	3300 ms	3700 ms	3300 ms	3300 ms	3400 ms	3700 ms	3400 ms	3400 ms	3300 ms	3300 ms	3300 ms	3300 ms
Packing	4210 ms	2440 ms	2150 ms	560 ms	230 ms	4120 ms	2220 ms	2050 ms	560 ms	310 ms	220 ms	15140 ms	4930 ms	2110 ms	1030 ms	540 ms	310 ms	210 ms
Poly Eval	59 ms	30 ms	15 ms	7 ms	2 ms	59 ms	30 ms	15 ms	7 ms	3 ms	2 ms	118 ms	61 ms	30 ms	15 ms	8 ms	4 ms	2 ms
Total Time	8300 ms	6220 ms	4800 ms	4040 ms	3510 ms	7640 ms	5370 ms	5040 ms	3880 ms	3630 ms	3620 ms	15470 ms	8360 ms	5480 ms	4320 ms	3820 ms	3650 ms	3500 ms