ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# How Practical is Single-Server Private Information Retrieval?

Master semester project

Sophia Artioli

February 14, 2023

Supervisor: Prof. Dr. Kenny Paterson, Co-supervisor: Dr. Tianxin Tang

Applied Cryptography Group
Institute of Information Security
Department of Computer Science, ETH Zürich

# Contents

Chapter 1

# Introduction

## 1.1 Motivation

A *private information retrieval* (PIR) protocol allows a client to retrieve a record from a *public database* without the database server knowing which record has been requested. In other words, the main objective of PIR is to protect query privacy. The definition can also be extended to *private databases*, making PIR an appealing cryptographic primitive for building various privacy-enhancing applications, such as private media consumption, web browsing, anonymous messaging, and certificate auditing [1].

PIR schemes can be divided into two categories: *multi-server PIR*, where the database is distributed across multiple servers, and *single-server PIR*, where the database is hosted on a single server. Multi-server PIRs [4, 5, 6] are typically much more efficient than single-server ones, but they come with the assumption of non-colluding servers, which can be difficult to achieve in the real world. This is because, in many cases, only one company provides the PIR service and therefore has access to all data stored on the servers. In comparison, single-server PIRs do not require such an assumption, but existing ones usually have higher computational costs. For instance, a ”naive” single-server PIR involves the client downloading the entire database to retrieve the desired record. It was believed that this approach could not be surpassed due to the strict PIR security property, but this solution is clearly infeasible for practical use.

This underscores the challenges faced by prior PIR protocols, as they either suffer from poor performance in a single-server setting or require unrealistic trust assumptions of non-colluding servers. However, the situation has changed dramatically with the introduction of the *offline-online* PIRs. During the offline phase, some information about the database is precomputed and exchanged to optimize the server’s processing time during the online phase. This information, known as the *”hint”*, allows the client to make unlimited

PIR queries with improved efficiency.

With the hope of getting closer to a single-server PIR that could possibly be deployed soon, we study the practicality of three recent state-of-the-art single-server offline-online PIR schemes: Simple [1], Double [1], and Spiral [2]. Given the limited evaluation results presented in their respective papers, we have established our own evaluation metrics to reassess their efficiency and determine if they are ready for deployment.

## 1.2 Protocol Overview

Before delving into performance evaluation, we first provide a high-level overview of the three protocols and highlight their similarities and differences.

Both **Simple** and **Double** organize a $N$-record database as a $\sqrt{N} \times \sqrt{N}$ matrix. In Simple, the client first downloads a hint from the server and can then make an unlimited number of PIR queries. To reduce the size of the hint, which grows linearly with $\sqrt{N}$, Double is introduced as a more efficient alternative when $N$ becomes sufficiently large. Double is built on Simple and stores an additional hint on the server, so that the client only needs to download a smaller hint that does not depend on $N$. The client can then make PIR queries using the downloaded hint and through recursive use of Simple, with similar server-side processing costs.

On the other hand, **Spiral** organizes the database as a hypercube. The client does not download the hint, but instead generates and uploads some public parameters (referred to as the hint in our work) to the server, which are used during the online phase. Additionally, Spiral compresses the queries to reduce the communication cost.

Despite these differences, the three offline-online PIR schemes share some key properties. The database is stored in plaintext on the server and they all use lattice-based encryption schemes as building blocks. Simple and Double use Regev encryption under the Learning With Errors (LWE) assumption, where each record is represented as a sequence of elements in some finite field. To achieve query compression and limit noise growth, Spiral uses both Regev and GSW encryption under the Ring Learning with Errors (RLWE) assumption, representing each database record as an element in some polynomial ring.

## 1.3 Performance Re-Evaluation and Results

We now briefly describe how we conduct our performance evaluation and provide an overview of our results. We emphasize that the original eval-

uations in the Simple, Double and Spiral papers were limited in scope. For example, [1] only showed results for specific database configurations with a fixed total database size for comparison.

This motivated us to re-evaluate their performance under our own performance metrics. Specifically, we evaluate the performance of these protocols under a broader range of database configurations (by varying the number of records and the record size), which simulate many practical settings. We run the experiments on the provided open-source implementations, which involve the full execution of the protocols, but in the local setting omitting the network cost.

We also present theoretical concrete expressions for communication costs, such as hint, query, response sizes, as well as asymptotic computational complexity. These theoretical results are compared with our experimental results. A detailed description of the evaluation process and results can be found in Chapter 5, but we provide a brief overview of the findings here.

Our experimental results indicate that during the offline phase, for some database configurations, both Simple and Double may result in significant latency, but this issue can be mitigated in multi-client scenarios by sharing the hint, as discussed in Chapter 6.1. The communication cost during the offline phase, which is related to exchanging the hint, is low for Spiral, acceptable for Simple, but infeasible for Double. During the online phase, the communication cost is manageable for Simple and Spiral, but not for Double in certain cases. The computational time is feasible for all protocols, as long as the record size is relatively small, in the hundreds of bytes range. For larger record sizes (e.g, $>$ 1 KB), however, substantial latency may occur, especially for Spiral.

In Chapter 7, we also examine the application of our experimental findings to practical PIR scenarios and evaluate their feasibility.

In conclusion, we found that the hint size and record size play critical roles in the performance of the three PIR protocols. While the hint size of Double is independent of the number of database records, it is only of theoretical interest as the hint size is prohibitively large, even larger than the total database size in our experiments! For relatively small record sizes, Simple and Spiral can be indeed practical. However, for large record sizes, query processing relies on homomorphic operations on each record, represented as a sequence of field/ring elements, which results in significant performance issues. Nevertheless, by taking into account parallelization, hardware acceleration, and optimization for batched access, there may still be potential for single-server offline-online PIRs to have a wider range of applications in the future.

Chapter 2

# Cryptographic Background

In this chapter, we lay out the cryptographic primitives used in Simple, Double, and Spiral.

## 2.1 Homomorphic Encryption

Homomorphic encryption enables performing operations, such as multiplication or addition, on ciphertexts, resulting in equivalent operations on the underlying plaintexts. We list the following homomorphic operations supported by the schemes employed in Simple, Double and Spiral. We omit the specification of the homomorphic encryption schemes and the plaintext domains for simplicity.

**Addition** Let ciphertexts $c_1$ and $c_2$ encrypt $m_1$ and $m_2$, respectively. Then the encryption algorithm which takes as input $c_1$ and $c_2$ outputs a ciphertext $c'$ such that $c'$ encrypts the addition of the plaintexts $m_1 + m_2$.

Furthermore, Spiral utilizes homomorphic encryption schemes that support multiplication, referred to as fully homomorphic encryption. We define the multiplication between ciphertexts and plaintexts as follows,

**Multiplication** Let ciphertexts $c_1$ encrpyt $m_1$ and let $m_2$ be another message in plaintext, then $c' = c_1 \cdot m_2$ encrypts the multiplication $m_1 \cdot m_2$.

## 2.2 Lattice-based Homomorphic Encryption

Over an $n$-dimensional vector space, a *lattice* is an infinite set of points represented by a collection of vectors. These vectors make up a finite set called the *lattice basis*, making storing the entire lattice grid in memory possible.

Simple, Double, and Spiral are built on homomorphic encryption schemes, relying on the computational hardness of some lattice problems. We discuss the *learning with errors* problem in detail in Section 2.3.

## 2.3 The Learning with Errors Assumption

To build encryption using LWE, we rely on the hardness of distinguishing between a message with added noise and a random sample. The indistinguishability is formally defined, with the goal of ensuring the security of the encryption scheme: for matrices $A$ and $B$, secret $s$, an error vector $e$ sampled from the error distribution $\chi$ and a random vector $r$, the LWE assumption claims that the following distributions are computationally indistinguishable:

$$(A, As + e) \overset{c}{\approx} (A, r) \tag{2.1}$$

Simple, Double and Spiral rely on this assumption. However, Spiral uses its Ring Learning With Errors (RLWE) variant. In Table 2.1, we compare the algorithms and parameters used by the three protocols.

| | Spiral | Simple/Double |
|---|---|---|
| Assumption and parameters | RLWE-$(d, m, q, \chi)$ with ring $R = \frac{\mathbb{Z}[x]}{(x^d+1)}$, $d$ is a power of 2, ring modulus (or encoding modulus) $q \in \mathbb{N}$ | LWE-$(n, m, q, \chi)$, $n \in \mathbb{N}$ the dimension of the secret i.e. the lattice dimension, ciphertext modulus $q \in \mathbb{N}$ |
| Random matrix (or vector) $A$ (or $a$) | $a \xleftarrow{R} R_q^m$ | $A \xleftarrow{R} \mathbb{Z}_q^{m \times n}$ |
| Error distribution $\chi$ | $\chi$ over $R_q$ | $\chi$ over $\mathbb{Z}_q^n$ |
| Secret $s$ | $s \xleftarrow{R} \chi$ | $s \xleftarrow{R} \mathbb{Z}_q^n$ |
| Error vector $e$ | $e \xleftarrow{R} \chi^m$ | $e \xleftarrow{R} \chi^m$ |
| Random vector $r$ | $r \xleftarrow{R} R_q^m$ | $r \xleftarrow{R} \mathbb{Z}_q^m$ |

**Table 2.1:** Comparing parameters of the LWE assumption across various protocols. We note that $m$ refers to the number of samples used. Furthermore, parameters $m, q$ and $\chi$ serve the same purpose in the (R)LWE assumption. However, they have different values in each protocol.

## 2.4 (R)LWE-based Encryption Schemes

Simple and Spiral utilize lattice-based encryption schemes based on Learning With Errors (LWE) and Ring Learning With Errors (RLWE) assumptions, respectively. We first introduce Regev's encryption scheme as follows.

**Regev's additive homomorphic encryption scheme** encrypt a message $\mu$ with the parameters defined in Table 2.1 as follows:

$$(a, c) = (a, a^\top s + e + \mu) \tag{2.2}$$

The decryption of the ciphertext $c$ follows for anyone knowing the secret key $s$ by computing $c - a^\top s$. Furthermore, we can exploit the

additive property of this encryption; given the ciphertexts $(a_1, c_1)$ and $(a_2, c_2)$, the resulting sum is $(a_1 + a_2, c_1 + c_2)$. The decryption of $(a_1 + a_2, c_1 + c_2)$ yields $(c_1 + c_2) - (a_1 + a_2)^\top s = c_1 - a_1^\top s + c_2 - a_2^\top s = \mu_1 + e_1 + \mu_2 + e_2$. Hence, we observe that this result is the sum of the plaintexts and the sum of noises, demonstrating the homomorphic addition property of the scheme.

### 2.4.1 HE Composition in Spiral

| | Supported homomorphic operations | Noise growth for addition | Noise growth for multiplication |
|---|---|---|---|
| BFV | Addition and multiplication | **Linear** in the number of additions | **Exponential** in the number of multiplications |
| GSW | Multiplication | **Linear** in the number of additions | **Linear** in the number of additions |

**Table 2.2:** Comparison of noise growth between BFV and GSW.

In this section, we discuss why Spiral uses a composition of two lattice-based homomorphic encryption schemes.

(R)LWE-based encryption schemes have a significant drawback due to noise growth. As previously mentioned, the ciphertexts produced by these schemes are noisy encodings of the plaintext, and homomorphic operations between ciphertexts increase the magnitude of the noise. If the noise exceeds a certain threshold, the correctness of the decryption may no longer hold. For example, as shown in In Table 2.2, the noise introduced by Brakerski and Fan-Vercauteren (BFV) scheme [7, 8] scales exponentially in the number of multiplications. The table also compares the scaling of noise between BFV and Gentry, Sahai, and Waters (GSW) with respect to the number of additions and multiplications. In this case, GSW increases linearly in the multiplicative depth rather than exponentially for BFV.

For this reason, Spiral uses a composition of two lattice-based schemes: Regev and GSW [3]. The latter achieves full homomorphism, which means it supports both multiplication and addition operations. This combination enables fully homomorphic encryption and control over noise growth.

Chapter 3

# Protocol Overview

This chapter provides a comprehensive overview of Simple, Double, and Spiral, focusing on their similarities and differences. We divide the chapter into the following sections: Section 3.1, database representation and intuition; Section 3.2.1, the online phase for server-side query processing; Section 3.2.2, the offline phase for pre-processing.

## 3.1 Database Representation and Intuition

Instead of using a one-dimensional mechanism, namely, an $N$-element array, for storing the database, Simple, Double, and Spiral employ alternative data structures. Furthermore, the database is maintained in its plaintext form on the server, and the queries are generated based on the database's structure, optimizing both query generation and server-side computations.

#### Simple

An $N$-record database is represented as $\sqrt{N} \times \sqrt{N}$ matrix. The query is a 2-dimensional vector $(i, j)$ that selects values from both database dimensions. The client generates the query in the following way:

**Query transformation** The client first builds a $\sqrt{N}$-dimensional unit vector consisting of zeroes everywhere, except at index $j$ a '1'. We denote it by $u_j = (0, 0, \ldots, 1, \ldots, 0)^T$, which indicates the desired record lies in the $j$-th column of the database matrix.

**Query encoding** The unit vector is then encrypted with Regev's LWE-based homomorphic encryption scheme.

#### Double

Double organizes the database in the same way as Simple. The query generation process involves generating two unit vectors that select the row and the column of the location of the requested record, and then encrypting these two unit vectors using the Regev encryption scheme.

#### Spiral

Spiral, on the other hand, organizes the $N$-record database as a hypercube. Suppose $N = 2^{r+s}$, then it becomes a hypercube with dimensions $2^r \times \underbrace{2 \times \cdots \times 2}_{s}$. To generate a query, the following steps are taken: first, create a query vector that selects each dimension of the database, represented by $(i, j_1, \ldots, j_s)$, where $i \in [0, 2^r - 1]$ and $j_1, \ldots, j_s \in \{0, 1\}$. Then, the query vector is compressed into a single scalar and encrypted with Regev's RLWE-based homomorphic encryption scheme.

## 3.2 Hint Generation and Query Processing

Upon receiving the client's encrypted query, the server performs computations between the query and the database to retrieve the requested record. To improve the efficiency of the server-side computations, some of these computations can be performed in advance, before the query is generated, as they do not depend on the queries.

The three PIR schemes comprise two phases: the offline phase, which includes pre-computations and the exchange of hints, and the online phase, which involves query processing on the server and response decoding on the client.

In Section 3.2.1, we will first outline the steps involved in server-side query processing, disregarding the offline phase. This allows us to identify computations that do not rely on the query and can be performed in advance, which are described in detail in Section 3.2.2.

### 3.2.1 Online Phase

#### Simple

We have seen in Section 3.1 that the query is the encryption of the unit-vector $u_j$ consisting of all zeros except at index $j$, selecting for the column where the record lies in the database.

The server then performs the matrix product between the database $D$ and the unit-vector $u_j$, which outputs the $j$-th column. A simplified version of the computation process is provided to understand which computations

can be performed in advance. Let $A$ be a random matrix and $e$ sampled randomly from the error distribution. As seen in Section 2.4, the encryption of $u_j$ is $Enc(u_j) = As + u_j + e$. The server then computes $D \times Enc(u_j) = D \times (As + u_j + e) = DAs + Du_j + De$. It is observed that the computation of $D \times A$ is not tied to the query and can be moved to the offline phase.

#### Double

Double's online phase uses Simple recursively. The first level operates similarly to Simple: the server executes the protocol as in Simple over the query, selecting the database column, which outputs a response vector. The second level of Simple is then executed over the response vector (which contains the whole column of the record) and the unit vector that selects the row. It is worth noting that, similar to Simple, certain computations can be performed beforehand in Double, but in this case, two computations can be pushed to an offline phase.

#### Spiral

We have seen in Section 3.1 that the query is represented as $(i, j_1, \ldots, j_s)$ where $i \in [0, 2^r - 1]$ and $j_1, \ldots, j_s \in \{0, 1\}$. Upon receiving the encrypted query from the client, i.e. the Regev encoding of a scalar, the server performs the following steps:

**Query expansion** The server expands the received scalar $c$ to the following representation:

1. Represent the first coordinate $i$ of the query: the scalar is expanded into a collection of matrices representing the bits 0 and 1, where the $i$-th matrix represents the plaintext '1' and all other matrices represent the plaintext '0'.
2. Represent subsequent components of the query: the query is further expanded into a second collection of matrices created to represent the components $j_1, \ldots, j_s \in \{0, 1\}$.

**Query processing** With the two collections of matrices, the server can retrieve the requested database record:

1. Process the first dimension of the query by multiplying the matrices representing '0's and a '1' with the database. This will yield a sub-database containing all the records with $i$ as their first coordinate.
2. Multiply the above sub-database with the components $(j_1, \ldots, j_s)$ of the original query.

**Combining lattice-based homomorphic schemes** From the query expansion step, we have $2r$ matrices of Regev encodings and $s$ matrices of GSW encodings. By using an *external product* algorithm [9][10], we can multiply ciphertexts from two different schemes when the two encodings are encoded with respect to the same key. In the query processing phase, the query is encrypted as such respectively:

1. The server expands each Regev-encoded component of the vector containing a '1' at position $i$ and 0's elsewhere to matrices.
2. The vector $(j_1, \ldots, j_s)$ is encoded with Regev and is then expanded to matrices as in the step above. These Regev matrices are then transformed to GSW-encoded matrices.

Finally, perform homomorphic multiplication between the Regev and GSW ciphertexts:

1. Multiply the Regev matrix ciphertexts with the database and obtain the sub-database of records that have as first coordinate $i$. After multiplication, the sub-database is no longer in plaintext.
2. Multiply the resulting sub-database with each GSW matrix in order to retrieve the desired record.

The multiplication between Regev and GSW matrices relies on the external product. This homomorphic encryption results in a linear noise growth in the multiplicative depth of the computation rather than exponential if other schemes were used, as noted in Table 2.2.

### 3.2.2 Offline Phase

From the online phase, we identify computations that are independent of the client's query and can therefore be executed prior to the query being made. These computations are transferred to the offline phase of the scheme, and the following presents the details of how this phase is carried out.

#### Simple

In the online phase of Simple, we observed that the matrix multiplication between the database $D$ and the matrix $A$ does not depend on the client's query. Hence, this matrix multiplication can be moved to the offline phase to improve performance. We refer to $hint_c = D \times A$ as the hint matrix, which the client downloads at the start of the protocol before making any queries.

#### Double

Double's offline phase involves creating two hints, one for each level of Simple. In the first level, the server generates a hint dependent on the database

and stores it. In the second level, another hint is generated and sent to the client, which does not need to depend on the database size anymore, as the first-level hint already accounts for it. This makes the second-level hint smaller in size than the first-level hint.

#### Spiral

The online phase of Spiral involves an expansion and matrix conversion, which require the generation of multiple parameters to carry out these transformations. Hence, the offline phase of Spiral consists of generating this set of parameters and uploading them to the server before the client starts making queries. The set of parameters generated are:

**Automorphism keys** These keys serve as key-switching matrices to enable the expansion of Regev-encoded polynomials as in Section 3.2.1.

**Conversion keys** The server needs Regev to GSW conversion keys *ck* to translate the Regev ciphertexts into GSW matrix ciphertexts as in Section 3.2.1.

Chapter 4

# Protocol Descriptions and Theoretical Costs

We now provide the description of each step of the protocols and compute their theoretical computational and communication costs. This will enable us to compare the theoretical results with the experimental results that we will present in Chapter 5.

## 4.1 Overview of Theoretical Costs

The communication cost, which is analyzed in Table 4.5, refers to the amount of data exchanged between the server and client. The sizes are defined in terms of the number of records $N$ in the database and the record size $r$ in bits, as well as protocol-specific parameters that determine the size of hints, queries, and responses. Furthermore, in Table 4.5, it is assumed that the record size is larger than an element in $\mathbb{Z}_p$ (i.e. $r \geq \log(p)$), which is typical in most relevant applications. If the record size is not larger than an element in $\mathbb{Z}_p$, we can replace the term $\lceil \frac{r}{\log(p)} \rceil$ by 1. As a result, the cost depends on the number of elements in $\mathbb{Z}_p$ stores the database. Additionally, we provide Table 4.2 that shows the asymptotic computational costs of different phases of the protocols.

The parameters necessary for computing the theoretical values are defined as follows:

**Simple & Double** The ciphertext modulus $q = 2^{32}$ and defines $\mathbb{Z}_q$. The plaintext modulus $p$ defines $\mathbb{Z}_p$. The lattice parameter $n_p$ is set to 1024.

**Spiral** Given $N$, the number of records in the database, the database parameters $\nu_1$ and $\nu_2$ satisfy $N = 2^{\nu_1+\nu_2}$ with $\nu_1, \nu_2 \in \mathbb{N}$. The polynomial ring $R$ is defined as $R = \mathbb{Z}[x]/(x^d + 1)$, where $d$ is a power of 2 and set

to 2048. The modulus $q$ is set as $q = 2^{56}$, and it defines the polynomial ring modulus $R_q$. The plaintext modulus $p$ defines the polynomial ring modulus $R_p$. The plaintext dimension $n_d$ represents the database records in $R_p^{n_d \times n_d}$ and is set to $n_d = 2$. Additionally, we define the decomposition bases that allow switching from Regev to GSW. That is, $t_{coeff}$, $t_{conv}$ and $t_{GSW}$, that are further discussed in Section 4.1.1.

We notice that for Simple and Double the costs depend on the term $N\lceil \frac{r}{\log(p)} \rceil$, which represents the number of $\mathbb{Z}_p$ elements in the database.

### 4.1.1 Parameter Selection

In this section, we define additional parameters for Spiral that are important for understanding the costs associated with different stages of the protocol. Additionally, we explain in detail the decomposition bases used for transitioning from one encryption scheme to another.

#### Simple & Double

In Section 4.1, the parameters have been previously defined, however, the plaintext modulus $p$ is not fixed like the other parameters, and its value depends on the number of elements in $\mathbb{Z}_p$ that stores the database. These values are presented in Table 4.1.

| Number of field elements that represents the database | $2^{26}$ | $2^{28}$ | $2^{30}$ | $2^{34}$ | $2^{38}$ | $2^{42}$ |
|---|---|---|---|---|---|---|
| Simple plaintext modulus $p$ | 991 | 833 | 701 | 495 | 350 | 247 |
| Double plaintext modulus $p$ | 929 | 781 | 657 | 464 | 328 | 231 |

**Table 4.1:** The parameters are selected to achieve the highest possible throughput while preserving correctness and security properties. These parameters are specified in the implementation of Simple and Double.

#### Spiral

The performance of Spiral depends on various parameters that must be adjusted to balance communication and computation costs. To determine the optimal values for these parameters, an automatic parameter selection tool is used to find the parameters that minimise the server computation time:

**The first database dimension** $\nu_1$ The parameter $\nu_1$ determines the number of rounds of query expansion the server needs to perform during the query processing phase.

**The subsequent database dimensions** $\nu_2$ The parameter $\nu_2$ determines the number of rounds needed to select for each dimension of the sub-database during the query processing phase.

**Decomposition bases** Spiral uses a combination of two cryptographic protocols: Regev and GSW. Both protocols operate under different decomposition bases.

- $z_{coeff}$ evaluates the automorphism in the expansion algorithm in used in Section 3.2.1. Define $t_{coeff} = \lfloor \log_{z_{coeff}}(q) \rfloor + 1$ and is fixed at $t_{coeff} = 56$.
- $z_{conv}$ is the decomposition base that translates scalar Regev encodings into matrix Regev encodings. Specifically, this is done in step 1 of the query processing to handle the first dimension of the query. Furthermore, we define $t_{conv} = \lfloor \log_{z_{conv}}(q) \rfloor + 1 \in \{2, 4, 8, 16, 32, 56\}$.
- $z_{GSW}$ is the decomposition base used to translate scalar Regev ciphertexts into matrix GSW encodings in step 2 of the query processing process, to translate subsequent components of the query. We define $t_{GSW} = \lfloor \log_{z_{GSW}}(q) \rfloor + 1 \in \{2, 4, 8, 16, 32, 56\}$.

**Gadget matrices** To translate ciphertexts from one encryption scheme to another, we need to be able to switch decomposition bases. To accomplish so, we define gadget matrices that allow switching from one basis to another.

Formally, let $z \in \mathbb{N}$, and $g_z^\top = [1, z, z^2, \ldots, z^{\lfloor \log_z(q) \rfloor}] \in R_q^{\lfloor \log_z(q) \rfloor + 1}$, the gadget matrix is defined as $G_{n,z} = I_n \otimes g_z^\top$. We write $g_z^{-1}$, which allows transforming a high-norm vector into a low-norm vector of a higher dimension. We also expand the notation to matrices and define $G_{n,z}^{-1}$ which applies $g_z^{-1}$ to each column of a matrix.

| Asymptotic computational cost | Spiral | Simple | Double |
|---|---|---|---|
| Hint generation | $O(\log N)$ | $O(n_d N \lceil \frac{r}{\log(p)} \rceil)$ | $O(n_d N \lceil \frac{r}{\log(p)} \rceil + n_d^2 \sqrt{N})$ |
| Query generation | $O(\log N)$ | $O(n_d \sqrt{N \lceil \frac{r}{\log(p)} \rceil})$ | $O(n_d \sqrt{N \lceil \frac{r}{\log(p)} \rceil})$ |
| Query processing | $O(\max(2^{\nu_1}, 2^{\nu_2}))$ | $O(N \lceil \frac{r}{\log(p)} \rceil)$ | $O(N \lceil \frac{r}{\log(p)} \rceil + n_d \sqrt{N \lceil \frac{r}{\log(p)} \rceil})$ |
| Response decoding | $O(n_p^2)$ | $O(n_d)$ | $O(n_d)$ |

**Table 4.2:** This table summarizes the asymptotic computational costs of each step in the PIR protocols. The record size, denoted by r, is measured in bits, and the number of records in the database is represented by N.

## 4.2 Offline Phase: Detailed Process and Theoretical Cost Analysis

In this section, we assess the cost of each phase of the protocol by considering the database representation and addressing the treatment of large records, as outlined in Section 4.2.1.

### 4.2.1 Database Representation and Query Generation

The asymptotic computational cost of generating the query for each protocol is presented in Table 4.2. We summarize the findings in terms of query generation computational cost and query size:

**Simple** We observe in Table 4.3 the query size after encryption grows with the number of field elements that represent the database. Table 4.2 compares the asymptotic cost for query generation against other protocol stages.

**Double** As seen in Table 4.3, the size of the query increases in proportion to the number of field elements in the database, which is multiplied by a factor of 2 due to the presence of two queries. The same reasoning applies to the generation time of the query.

**Spiral** In terms of computational cost, most operations require constant time except for the subsequent dimensions encoding, which requires $\nu_2 \cdot t_{GSW}$ operations in $R_q$.

We now delve into each step of the query generation process, analyzing its cost and providing a thorough explanation of the database representation.

#### Simple & Double

For an $N$-record database, the server stores the database as a matrix of dimension $\sqrt{N} \times \sqrt{N}$, with each matrix element representing a single record from the database. Each record is an element of the set of integers modulo $p$, denoted as $\mathbb{Z}_p$. However, it is worth noting that in many practical applications of private information retrieval (PIR), the size of the records may exceed that of a single element in $\mathbb{Z}_p$ (i.e. $r \geq \lceil \log(p) \rceil$ ). To accommodate such scenarios, Simple and Double supports larger records by breaking each record into multiple elements in $\mathbb{Z}_p$, and subsequently stacking them vertically within the same column of the database matrix.

When a client requests a specific record from the database, the requested index is decomposed into a pair of coordinates $(i, j)$, where $i, j \in [\sqrt{N}]$.

To generate the query, the client follows the following steps:

**Query generation** The client creates a dimension-$\sqrt{N}$ unit-vector $u_j$ from the query $(i, j)$, where the vector contains zeroes everywhere except for a '1' at index $j$.

**Query encoding** The client encrypts the unit-vector using Regev's LWE-based encryption scheme. Let $A \xleftarrow{R} \mathbb{Z}_q^{\sqrt{N} \times n_p}$, secret $s \xleftarrow{R} \mathbb{Z}_q^{\sqrt{N}}$, and error vector $e \xleftarrow{R} \chi$:

$$Enc(u_j) = (A, c) = (A, As + e + \lfloor q/p \rfloor \cdot u_j) \in \mathbb{Z}_q^{\sqrt{N} \times n_p}.$$

**Double.** Double's query generation and encryption procedure are similar to those of Simple. However, it incorporates an additional unit vector $u_i$ that encodes the row of $(i, j)$ as part of the query.

| | Simple | Double |
|---|---|---|
| Query size [bits] | $\sqrt{N\lceil log(\frac{r}{\log(p)})\rceil} \cdot \lceil \log(q) \rceil$ | $2 \cdot \sqrt{N\lceil log(\frac{r}{\log(p)})\rceil}\lceil \log(q) \rceil$ |
| Response size [bits] | $\sqrt{N\lceil log(\frac{r}{\log(p)})\rceil} \cdot \lceil \log(q) \rceil$ | $(2n+1)\lceil \log(q)/\log(p) \rceil \lceil \frac{r}{\log(p)} \rceil \lceil \log(q) \rceil$ |

**Table 4.3:** The table shows that for Simple, both the theoretical query and response size increase with the number of $Z_p$ elements in the database and are equal. On the other hand, for Double, the query size is twice the query of Simple due to the execution of two levels of Simple.

### Spiral

The database is represented as $D = \{d_1, \ldots, d_N\}$ with $N = 2^{\nu_1+\nu_2}$ the number of records in the database, as a $(\nu_2 + 1)$-dimensional hypercube with dimensions $2^{\nu_1} \times \underbrace{2 \times \cdots \times 2}_{\nu_2}$.

Furthermore, each database record $d_i$ is represented as an element in the ring $R_p^{n \times n}$.

**Large records** Let r be the database record size in bits. We qualify a record as 'large' when $r \geq d \cdot n^2 \log(p)$. In this case, each plaintext element can encode at most one record, and we require $2^{\nu_1+\nu_2} \geq N$.

**Small records** When $r < d \cdot n^2 \log(p)$, we can pack multiple records into a single plaintext element.

The elements of the database are indexed using either the tuple $(i, j_1, \ldots, j_{\nu_2})$ where $i \in [0, 2^{\nu_1} - 1]$ and $j_1, \ldots, j_{\nu_2} \in \{0, 1\}$, or the tuple $(i, j)$ where $i \in [0, 2^{\nu_1} - 1]$ and $j \in [0, 2^{\nu_2} - 1]$.

The query generated is represented in plaintext and is matched with the database representation outlined in Section 4.2.1. The client selects each database dimension, $i, j_1, \ldots, j_{\nu_2}$, where $i \in [0, 2^{\nu_1} - 1]$ and $j_1, \ldots, j_{\nu_2} \in \{0, 1\}$, and encodes the vector, and subsequently compacts it into a single scalar Regev ciphertext as follows:

**First dimension encoding** The client encodes the first component of the query, $i$ from the query vector $(i, j_1, \ldots, j_{\nu_2})$, and defines the polynomial $\mu_i(x) = \lfloor q/p \rfloor \cdot x^i$.

**Subsequent dimensions encoding** For $\nu_2$, we define the polynomial $\mu_j = \sum_{l \in [\nu_2]} \mu_{j_l}$ where $\mu_{j_l} = j_l \sum_{k \in [t_{GSW}]} (z_{GSW})^{k-1} x^{(l-1)t_{GSW}+k}$.

**Query compression** We then compress the query to a single Regev scalar by computing $\mu(x) := 2^{-r_1}\mu_i(x^2) + 2^{-r_2} x \mu_j(x^2) \in R_q$.

**Query encryption** We proceed to encode the scalar from the previous step with Regev's RLWE-based encryption scheme as such: let $s$ be the secret key, and $a$ a random vector. The compressed query $\mu$ is encoded as $(a, c) = (a, a^\top s + e + \mu)$.

As shown in Table 4.4, the query size with and without compression is reported, and it can be observed that the query size is reduced by a factor of $(\nu_1 + \nu_2)$.

| *Spiral* | Query before encryption | Query size after Regev encryption |
|---|---|---|
| Encrypted query size without compression | $(\nu_2 + 1)$-dimensional vector | $2 \cdot (\nu_1 + \nu_2) d \lceil \log(q) \rceil$ bits |
| Encrypted query size with compression | scalar $\mu \in R_q$ | $2 \cdot d \cdot \lceil \log(q) \rceil$ bits |

**Table 4.4:** In the Spiral protocol, the client performs query compression to reduce the communication cost before encrypting the query. This results in the encoded query being reduced to the Regev encryption of a scalar in $R_q$.

### 4.2.2 Hint Generation

| | Spiral | Simple | Double |
|---|---|---|---|
| Hint size | $d\lceil \log q \rceil \cdot 2t_{coeff} \cdot \max(\nu_1, \lceil log t_{GSW} \nu_2 \rceil)$ $+(n_p+1)(2t_{conv} + t_{GSW}(1 + t_{GSW}(n_p+1)))$ | $n_d \lceil \log q \rceil \cdot \sqrt{N \lceil \frac{r}{\log(p)} \rceil}$ | $\lceil \frac{r}{\log(p)} \rceil \lceil \log(q)/\log(p) \rceil \lceil \log(q) \rceil n_d^2$ |
| Query size | $d \cdot \lceil \log q \rceil$ | $\lceil \log(q) \rceil \cdot \sqrt{N \lceil \frac{r}{\log(p)} \rceil}$ | $2\lceil \log(q) \rceil \cdot \sqrt{N \lceil \frac{r}{\log(p)} \rceil}$ |
| Response size | $d(n^2 \log q_1 + n \log q_2)$ | $\lceil \log q \rceil \cdot \sqrt{N \lceil \frac{r}{\log p} \rceil}$ | $\lceil \frac{r}{\log p} \rceil \lceil \log q / \log p \rceil \lceil \log q \rceil (2n+1)$ |

**Table 4.5:** We present the theoretical sizes of the PIR schemes we have evaluated as a function of the number of records $N$ and the record size $r$. We assume the general case of PIR where the record size is larger than the size of a single field element (i.e. $\mathbb{Z}_p$ for Simple and $R_p$ for Spiral). It is important to note that the variables in the table are defined within the context of their respective protocols and may not have the same values across different schemes.

The asymptotic computational cost of the query processing phase for each protocol is presented in Table 4.2 and the hint size for all protocols can be observed in Table 4.5. We summarize the findings below:

**Simple** The offline phase enables the server to push $2n_pN$ operations in $\mathbb{Z}_q$ from the online phase to the offline phase. Furthermore, Table 4.2 shows the computational cost of the hint generation time scales with the total number of field elements in $\mathbb{Z}_p$ that represents the database.

**Double** By implementing the offline phase, the computation of

$2nN + 2\lceil \log q / \log p \rceil n_p^2 \sqrt{N}$ operations in $\mathbb{Z}_q$ can be moved from the online phase to the offline phase. Additionally, the hint generation time contains an additional term compared with Simple, which corresponds to the second-level hint generation, enabling the client not to download the first-level hint.

**Spiral** The overall cost of the offline phase is expressed as $\max(\nu_1, \log t_{GSW} \nu_2) \cdot 5t_{coeff} + 4n(t_{conv} + t_{GSW} + 4t_{GSW} + (n+1)^2 t_{GSW}$, with the main factor

being $\max(\nu_1, \log t_{GSW}\nu_2)$, which can be approximated by $\log(N)$.

We now provide a detailed account of the computational costs during the offline phase.

### Simple

The server first generates a random matrix $A \in \mathbb{Z}_q^{\sqrt{N} \times n_p}$ and then computes the hint as $hint_c = D \times A$, where $D$ is the database matrix. This computation does not depend on the queried index or any other parameters. Therefore, the client can download the hint before making queries, improving the efficiency of server-side query processing.

### Double

In Double, two hints are generated, with the first being referred to as the server hint. The server hint is calculated by multiplying a random LWE matrix, $A_1 \in \mathbb{Z}^{n_p \times \sqrt{N}}$, with database $D$, to obtain $hint_s = A_1 \times D \in \mathbb{Z}_q^{n_p \times \sqrt{N}}$. This hint is stored on the server, and its size depends on the database's size. The second hint referred to as the client hint, is obtained by multiplying the server hint with another random LWE matrix, $A_2 \in \mathbb{Z}_q^{\sqrt{N} \times n_p}$, resulting in $hint_c = hint_s \times A_2 \in \mathbb{Z}_q^{n_p \times n_p}$. We can observe that the client hint no longer depends on the database size.

### Spiral

In the online phase of Spiral, most calculations are specific to the given query and must be carried out after the query has been generated. However, certain operations require using various types of keys, which are computationally expensive to generate.

**Automorphism keys** During query processing, automorphism keys are essential to expand the Regev scalar (i.e. the packed query). To improve efficiency, the generation of these keys is moved to the offline phase, where the client generates them in advance. The cost of generating each key is $W_i$ for $i \in [0, \max(\nu_1, \log t_{GSW}\nu_2)]$ and requires $(\max(\nu_1, \log t_{GSW}\nu_2)) \cdot 5(\lfloor \log_{z_{coeff}} q \rfloor + 1)$ ring operations in $R_q$.

**Conversion keys** In step 2 of the query processing phase, the server is required to convert Regev scalar ciphertexts into GSW encoded matrices. To accomplish this, conversion keys $ck$ are utilized, which are composed of three components:

- $V$ The matrix $V$ acts as the key-switching matrix to translate Regev-encoded matrices to GSW-encoded matrices. The process of generating $V$ takes $4(n(t_{conv} + t_{GSW}) + t_{GSW})$ operations in $R_q$.

$\Pi$ The permutation matrix $\Pi$ is utilized in this step to transform a matrix into a gadget matrix, which is particularly useful in the process of translating scalar Regev-encoded ciphertexts to GSW-encoded matrices.

$W$ In step 1 of the query processing phase, the matrix $W$ is used to convert scalar Regev ciphertexts to Regev matrices. $W$ is the key-switching matrix to translate the scalar ciphertext with the corresponding matrix ciphertext. The process of generating matrix $W$ takes $n(\lfloor \log_{z_{conv}} q \rfloor + 1)(4 + n)$ operations in $R_q$.

## 4.3 Online Phase: Detailed Process and Theoretical Cost Analysis

### 4.3.1 Query Processing

The asymptotic computational cost of the query processing phase for each protocol is presented in Table 4.2 and is summarized below:

**Simple** As observed in Table 4.2, the server-side query processing computation time grows with the number of field elements in the database and is concretely equivalent to $2N$ operations in $\mathbb{Z}_q$.

**Double** The query processing phase also grows asymptotically with the number of field elements in the database, as observed in Table 4.2. Precisely, the number of operations for the online phase is $2N + 2(2n + 1) \cdot \sqrt{N} \lceil \log q / \log p \rceil$ operations in $\mathbb{Z}_q$.

**Spiral** The cost of converting Regev scalars to Regev matrices and Regev scalars to GSW matrices are reported in Table 4.6. As a result, after considering all steps of Spiral's server-side computation, the overall computational cost is determined by the largest value between $2^{\nu_1}$ or $2^{\nu_2}$.

We now elaborate on the computations carried out by the server to process the queries.

#### Simple

After receiving the encrypted query $c \in \mathbb{Z}^{\sqrt{N}}$ of the query vector $u_j$, the server computes the product $D \times c$, where $D \in \mathbb{Z}^{\sqrt{N} \times \sqrt{N}}$ is the database.

#### Double

Double first performs an iteration of Simple using the hint stored on the server and the query vector to encrypt the $j$-th column of the database where

the record is located. Then, it performs a second iteration of Simple on the transposed server hint matrix and response vector, thereby retrieving the encrypted record at entry $(i, j)$ in the database.

### Spiral

From the client's query $c \in R_q^2$, the server constructs the response containing the requested database record in its encrypted form as follows, with algorithms for ScalToMat and RegToGSW provided in Appendix A.1.

**Query expansion** The server begins by expanding the initial query $c$, using an algorithm that takes as input polynomial from $R_q$ and outputs a vector containing the coefficients of the polynomial. This algorithm makes use of the automorphism keys defined in Section 4.2.2:

- First, the server uses one iteration of the homomorphic evaluation of the coefficient expansion algorithm on $c \in R_q^2$. The algorithm's output consists of the two encodings $c_{Reg}$ and $c_{GSW} \in R_q^2$.
- Then, evaluate $c_{Reg}$ for $\nu_1$ iterations, which will output $2^{\nu_1}$ Regev ciphertexts.
- Finally, use a Scalar-To-Matrix (ScalToMat) algorithm that converts scalars into a matrix representation: convert the $2^{\nu_1}$ packed Regev scalar ciphertexts to Regev ciphertext matrices by computing for each $i \in [0, 2^{\nu_1} - 1]$,

$$C_i^{(Reg)} = \text{ScalToMat}(W, c_i^{(Reg)}).$$

  Then, $2^{\nu_1}$ number of Regev matrix ciphertexts are constructed from the public parameters $W$ and scalar ciphertexts $c_i^{(Reg)}$.

After that, proceed in the same way as above to expand the GSW ciphertext:

- First, run the homomorphic expansion algorithm for $\lceil \log_{t_{GSW}} \nu_2 \rceil$ iterations on $c_{GSW}$ and discard any encodings when $t_{GSW}\nu_2$ is not a power of 2.
- Next, rely on the Regev to GSW translation algorithm (RegevToGSW) to compute $\nu_2$ GSW matrix ciphertexts from the public parameter's Regev-to-GSW conversion keys and the GSW scalar ciphertexts. Namely, compute

$$C_j^{(GSW)} = \text{RegevToGSW}(ck, c_{(j-1)t_{GSW}+1}^{(GSW)}, \ldots, c_{jt_{GSW}}^{(GSW)}).$$

**Homomorphic encryption** Now proceed by homomorphically multiplying the Regev and GSW ciphertexts:

| Operations | Theoretical computational cost |
|---|---|
| Scalar to Matrix | $2^{\nu_1} n(n+1)(n(\lfloor \log_{z_{conv}}(q) \rfloor + 1) + 1 + n) + 2n)$ |
| Regev to GSW | $t_{GSW} \cdot n(n+1)((n(\lfloor \log_{z_{conv}}(q) \rfloor + 1) + 1 + n) + 2n) + 2t_{GSW} n + (n+1)2t_{conv} t_{GSW} + (n+1)^3 t_{GSW}^2$ |

**Table 4.6:** Theoretical cost of the operations used for query processing in the Spiral protocol.

- First, the first dimension $i$ of the query is processed. For every $j \in [0, 2^{\nu_2} - 1]$: $C_j^{(0)} = C_0^{(Reg)} \cdot d_{0,j}$ and for every $i \in [0, 2^{\nu_1} - 1]$, update $C_j^{(0)} = C_j^{(0)} + C_i^{(Reg)} \cdot d_{i,j}$.
- Then, multiply GSW matrix ciphertext with an Regev matrix ciphertext. For every $r \in [\nu_2]$ and $j \in [0, 2^{\nu_2 - r} - 1]$, $C_j^{(r)} = (G_{n+1, z_{(GSW)}} C_r^{(GSW)}) \cdot C_j^{(r-1)} + C_r^{(GSW)} \cdot C_{2^{\nu_2 - r} + j}^{(r-1)}$.

**Modulus switching** Finally, the protocol scales the ciphertext matrix $C$ down to a smaller ring element while preserving the encoded message. This is called modulus switching. Namely, let $q_1 = 4p$ and $q_2$ be the smaller modulus associated with the PIR response. We scale the first dimension of the response by $q_2/q$ and the subsequent dimensions by $q_1/q$. The server can then send the encoded response in a more compact representation to the client.

### 4.3.2 Response Decoding

Upon receiving the encrypted record computed by the server, the client decrypts it using the relevant decryption algorithm. Table 4.2 demonstrates that the decoding time for Simple, Double, and Spiral is asymptotically constant.

#### Simple

The client receives an encoded response vector, which is the product of the database and the query, resulting in obtaining the $j$-th column of the database in its encrypted form. To obtain the element $i$ from the response vector, the client calculates the difference between the $i$-th component of the response vector and the $i$-th row of the hint, using the secret key $s \in \mathbb{Z}_q^n$: $d = ans[i] - hint_c[i, :] \cdot s$. This difference is rounded to the nearest multiple of $\lfloor q/p \rfloor$ and divided by $\lfloor q/p \rfloor$. This component corresponds to the record query $(i, j)$ and allows the client to retrieve the requested record.

#### Double

Double works similarly. The client retrieves the encrypted record using the downloaded client-side hint.

### Spiral

The client retrieves the original record by first undoing the modulus switching procedure on the received response. This is done by recalculating the response in modulus $q$. Next, the client utilizes Regev-RLWE decryption algorithm to obtain the requested record.

Chapter 5

# Performance Re-Evaluation and Results

As mentioned in the introduction, we are interested in re-evaluating the performance of Simple, Double, and Spiral, under different database configurations that vary the number of records and record size. Our experiments examine both the offline and online phases, focusing on both *computational costs* and *communication costs*. During the offline phase, we measure the *hint generation time* and *hint size*. In the online phase, we assess the *query generation time* and *query size*, as well as the *query processing time* and *response size*. The efficiency of the response size is measured using the *rate*, which represents the ratio between the plaintext record size and the actual response size.

In the first section of this chapter Section 5.1.1, we analyze the communication cost incurred during the offline phase. Our results show that Spiral has a small hint size that never exceeds 13 MB, while Simple has a hint size that is smaller than the database size but still substantial at 915 MB for a 32 GB database. On the other hand, Double has a large hint size that surpasses the database size, with the smallest tested database being 4 MB and the smallest hint size being 3 GB. As a result, Double is unlikely to be practical for use.

We also assess the communication cost of the online phase in terms of query size, response size, and response rate in Section 5.3, Section 5.2.4, and Section 5.2.5, respectively. Our results indicate that these schemes are acceptable in this regard, except for Double, which generates responses of a large size and infeasible rate for practice. For example, the response size of Double reaches 3.7 GB for a single response on a 32 GB database.

Regarding the computational costs, Section 5.1.2 shows that hint generation for Simple and Double may result in substantial query latency. In contrast, query generation time only contributes a small part of the overall query latency, as discussed in Section 5.2.1. In addition, in Section 5.2.3, we demon-

strate that the query processing time for Spiral can become unreasonable (i.e., $> 30$ s) when the record size is larger than 64 KB. On the other hand, all three protocols have query processing times of less than 1 second when the record size is smaller than 1 KB.

Our experiments were conducted under different database configurations, with the number of records $N$ ranging from $2^{14}$ to $2^{20}$ and record size $r$ ranging from 64 B to 256 KB. However, it is important to note that the computational times for some configurations with high numbers of records ($2^{19}$ and $2^{20}$) together with large record sizes (above 64 KB) were not reported as the experiments were unable to output results due to constraints in the implementations and computing resources.

The experiments were conducted locally, ignoring network transmission times, on 4vCPUs as part of Daisen, a computing cluster with $2 \times 28$ Core, Intel Xeon Gold 6258R 2.7GHz Processor, and 384GB DDR4 ECC Memory. The experiments were conducted using a single thread of execution. The statistics reported are an average of 5 experimental runs, replicating the settings used in previous works [1, 2].

We believe that our experimental results provide an understanding of the practicality of these PIR schemes, which will be further discussed in Chapter 7.

## 5.1 Offline Phase

The offline phase of Simple, Double, and Spiral involves computing and exchanging a hint between the client and the server, to improve the efficiency of query processing during the online phase. In our evaluation, we measure the *hint size* and *hint generation time* under various database configurations. The hint size plays a critical role in the communication cost of the offline phase, while the hint generation time can affect the query latency.

### Hint Size vs Total Database Size

In Table 4.5, we presented the theoretical expression for computing the hint size for Simple, which increases with the number of records and the size of records. However, we noticed that in some cases, the hint size exceeds the total database size. We provided examples of such scenarios in Table 5.1.

To address this issue, we established a requirement for the database configurations, which guarantees Simple's hint size is less than the total database size. This requires the total database size $N \cdot r \geq \frac{32KB}{\lceil \log p \rceil}$, where $p$ is the plaintext field modulus, as defined in Table 4.1.

In contrast, for Double, the hint size has never been smaller than the total database size in any of the configurations tested, rendering it infeasi-

| Number of records × record size | Total DB size | Simple hint size |
|---|---|---|
| $2^{14}\times$ 256 B | 4.0 MB | 7.2 MB |
| $2^{15}\times$ 256 B | 8.0 MB | 10.4 MB |
| $2^{17}\times$ 64 B | 8.0 MB | 10.3 MB |
| $2^{14}\times$ 920 B | 14.4 MB | 14.1 MB |
| $2^{17}\times$ 128 B | 16.0 MB | 14.0 MB |

**Table 5.1:** The first three rows present the database configurations where the hint size of Simple is greater than the total size of the database. The last two rows show database configurations where the difference between the database size and the hint size is not substantial for practical use.

ble. We have found that theoretically, for Double's hint to be smaller than the database size, the number of records must be $N \geq \frac{2^{27}}{\lceil \log p \rceil}$, with the record size smaller or equal to that of a field element ($\log p$). Thus, Double's intended objective of reducing the hint size download by the client and creating a hint independent from the number of records still falls short of addressing the practicality of the scheme.

However, Spiral does not face the same limitations as Simple and Double, as its hint size can be bounded. The database configurations used in our experiments are within the parameters defined in the respective paper, with a lower bound of approximately 3 MB and an upper bound of 40 MB. It is also worth mentioning that Spiral utilizes an automatic selection tool to optimize the parameters and minimize the server computation time.

### 5.1.1 Hint Size

Our experimental results confirmed the theoretical analysis of the hint size as follows:

**Simple** The experimental results validated that Simple's hint size increases linearly with the square root of the number of elements in $\mathbb{Z}_p$, representing the database, which is consistent with our theoretical findings.

**Double** If the record size is larger than an element in $\mathbb{Z}_p$, the client hint size is estimated to be approximately $16 \cdot r$ MB. On the other hand, if the record size is less than or equal to an element in $\mathbb{Z}_p$, the client hint size for Double is consistently 16 MB, regardless of the number of records.

**Spiral** Spiral's hint size is primarily influenced by the logarithm of the database dimension and the parameters defined in 4.1.1, yielding the smallest hint size.

Figure 5.1 demonstrates that the hint size for Spiral is significantly smaller than that of Simple and Double. Specifically, Simple's hint size is at least two times larger, while Double's hint size is at least 400 times larger than Spiral.

Both Spiral and Simple maintain reasonable hint sizes for all database configurations, particularly since the offline phase only needs to be performed once to support an unlimited number of queries for static databases.

In contrast, our experimental results revealed that none of the tested database configurations allowed Double to have a smaller hint size than the total database size. It is worth mentioning that Double requires storing an additional hint on the server. This server hint is reduced by $n_d = 1024$ compared with the client hint, ranging from approximately 3 MB to 1 GB for large databases (i.e. $2^{20} \times 32$ KB).

| *Database configuration* | Total DB size | Simple hint size | Double hint size | Spiral hint size |
|---|---|---|---|---|
| $2^{14} \times$ 1 KB | 16 MB | 16 MB | 14 GB | 6 MB |
| $2^{14} \times$ 256 KB | 4 GB | 844 MB | 3807 GB | 41 MB |
| $2^{15} \times$ 16 KB | 512 MB | 105 MB | 224 GB | 7 MB |
| $2^{16} \times$ 256 B | 16 MB | 14 MB | 3 GB | 6 MB |
| $2^{17} \times$ 256 KB | 16 MB | 915 MB | 3807 GB | 13 MB |
| $2^{19} \times$ 256 B | 128 MB | 41 MB | 3 GB | 7 MB |
| $2^{19} \times$ 64 KB | 32 GB | 706 MB | 950 GB | 13 MB |
| $2^{20} \times$ 256 B | 256 MB | 58 MB | 3 GB | 7 MB |
| $2^{20} \times$ 16 KB | 16 GB | 514 MB | 237 GB | 10 MB |

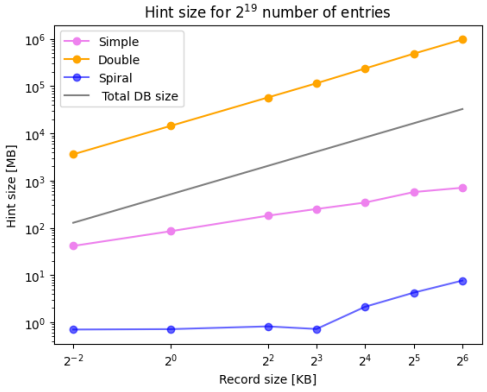


**Figure 5.1:** The left table displays the hint sizes for various database configurations. The right plot illustrates the trend for hint sizes by varying the record size while fixing the number of records.

### 5.1.2 Hint Generation Time

As shown in Table 4.2, the asymptotic hint generation time depends on the total database size. Specifically, we observed that Spiral's hint generates much faster than Simple and Double: under the various database configurations tested, Spiral's hint generation is up to 300 times faster than Simple and Double. As seen in Figure 5.2, Spiral exhibits slower growth as its computational complexity grows logarithmically in the number of records and is primarily affected by other parameters, such as the decomposition bases. The trend observed in Figure 5.2 can be generalized for all record sizes.

It is important to note that the hint generation time is a factor in the overall latency of the protocol for only offline phases, or in other words, the first PIR query with a static database, and it does not affect the latency of subsequent queries. Therefore, Spiral has a low hint generation time and only accounts for a minor portion of the latency for the first PIR query. As illustrated in Figure 5.1, Simple and Double's hint generation time increases with the number of records and their size. It may result in significant latency for

certain combinations, reaching up to 2 minutes for Simple and 4 minutes for Double for a database configuration of $2^{20} \times$ 32 KB. The practicality of the hint generation time depends on the specific application, as it can be distributed over multiple queries or clients if there are minimal database updates, as discussed in Chapter 6.1.

| Hint generation time | Number of records | Simple Record size | Double Record size |
|---|---|---|---|
| under 0. 5 s | $2^{14}$ | $\leq$ 128 KB | $\leq$ 256 B |
| | $2^{17}$ | $\leq$ 1 KB | $\leq$ 256 B |
| | $2^{19}$ | $\leq$ 256 B | None |
| under 1 s | $2^{14}$ | $\leq$ 256 KB | $\leq$ 256 KB |
| | $2^{17}$ | $\leq$ 1 KB | $\leq$ 256 B |
| | $2^{19}$ | $\leq$ 256 B | None |
| | $2^{20}$ | $\leq$ 256 B | None |
| under 10 s | $2^{19}$ | $\leq$ 4 KB | $\leq$ 256 B |
| | $2^{20}$ | $\leq$ 1 KB | $\leq$ 256 B |
| under 30 s | $2^{17}$ | $\leq$ 64 KB | $\leq$ 16 KB |
| | $2^{19}$ | $\leq$ 16 KB | $\leq$ 4 KB |
| | $2^{20}$ | $\leq$ 8 KB | $\leq$ 1 KB |

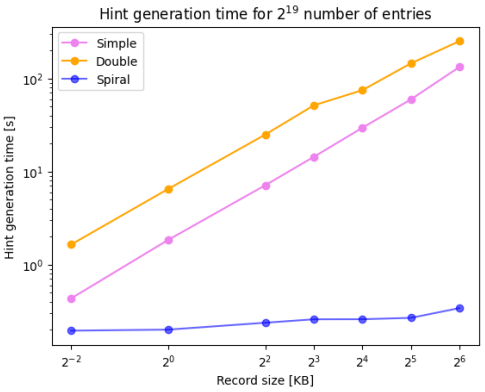


**Figure 5.2:** In the plot on the right, we observe the trend of the hint generation time as a function of the record size with a specific number of records ($N = 2^{19}$) for Simple and Double. The table on the left displays the hint generation time for various database configurations, excluding Spiral because its hint generation time is consistently less than 0.4 seconds in all experiments, significantly faster than both Simple and Double.

## 5.2 Online Phase

Since the offline phase is to precompute the data for more efficient query processing, we now discuss the server-side query processing times in the online phase.

### 5.2.1 Query Generation Time

Query generation time can significantly impact the overall efficiency, especially when multiple queries need to be made. Our experiments showed that for all three protocols, the query generation times are acceptable and only account for a small portion of the total latency. We observed that the query generation time for Spiral remains consistent, around 440 $\mu s$, which is consistent with the query size. This is because Spiral computes a polynomial of degree $n_d$ and compresses it to a single scalar, while Simple and Double involve constructing and encrypting a vector with a size linear in $\sqrt{N}$. It is worth noting that the query generation time for Simple and Double is a minor component of the total latency and does not surpass 400 ms for every database configuration.

### 5.2.2 Query Size

The *query size* refers to the amount of the data a client must send to a server for every PIR query. We now examine the query size for each of the three protocols.

**Simple** The theoretical query size grows as the square root of the number of field elements that represent the database. This is consistent with the experimental results shown in Figure 5.3. We consider the results reasonable. For example, the largest query size for Simple is 682 KB when the total database size is 32 GB.

**Double** The theoretical query size of Double also increases with the square root of the number of field elements that represent the database. However, our experimental results show that the query size remains constant at 256 KB for all database configurations, with a total size under 8 GB. This is due to the fact that the database is represented as a matrix, and Double's implementation fixes the number of columns and adjusts the number of rows accordingly to the number of records and field elements per record. This means that the query size is fixed at 256 KB, regardless of the database configuration. On the other hand, for larger databases, such as a $2^{19} \times 64$ KB configuration with a total size of 32 GB, the query size can reach 1 GB, which is excessively large. Even for small databases with few records and a small record size (i.e. $2^{14} \times 256$ bytes), the query size is still at 256 KB.

**Spiral** The theoretical query size for Spiral is constant, as shown in Table 4.5. This is in line with the experimental results, which show that the query size remains constant at 16 KB, regardless of the database size. This consistency is achieved by compressing each query into a scalar representation.

### 5.2.3 Query Processing Time

*Query processing time* is a key metric for PIR because it directly affects the overall query latency.

Our findings show that the query processing time of Spiral is significantly higher compared with Simple and Double, as depicted in Figure 5.4, which displays the processing times for a fixed number of records of $2^{19}$. This trend holds for other numbers of records as well. This difference in processing time is a result of the query compression/expansion technique utilized by Spiral. The compression reduces the communication cost by converting the query from a vector to a scalar, but it also increases the query processing time on the server side.

| Number of records | Query Size |
|---|---|
| $2^{14}\times$ 1 KB | 12 KB |
| $2^{15}\times$ 256 KB | 128 KB |
| $2^{17}\times$ 64 KB | 256 KB |
| $2^{20}\times$ 256 B | 57 KB |
| $2^{20}\times$ 32 KB | 682 KB |

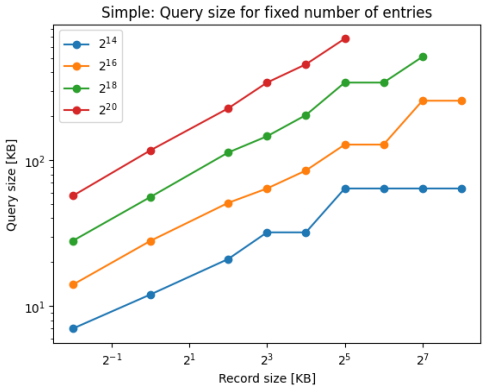


**Figure 5.3:** On the left side of the figure, we display the query sizes for Simple for various database configurations. On the right, we observe a trend where the query size increases proportionately to the square root of the total number of elements in $\mathbb{Z}_p$.

In contrast, Simple and Double mainly involve matrix multiplication, and their computational time increases with the database size, as shown in Table 4.2. As shown in Figure 5.4, when the number of records is high, Simple and Double can handle larger record sizes than Spiral with a similar processing time. Spiral has reasonable processing times (less than 1 second) for smaller record sizes (less than 256 bytes), regardless of the number of records. For the $2^{20} \times 32$ database, the experimental results show that Simple and Double have processing times of 7.5 seconds and 9.4 seconds, respectively. However, Spiral has a processing time of 25.2 seconds, three and two times higher than Simple and Double, respectively. Additionally, Spiral requires 2 minutes for processing a $2^{18} \times 128$ KB query, whereas Simple and Double require 5 seconds and 8 seconds, respectively.

Our experiments also showed that the query processing times of Simple, Double, and Spiral are similar when the record size is below 1KB, regardless of the number of records. However, as the record size increases, Simple and Double demonstrate much faster query processing time than Spiral.

### 5.2.4 Response Size

During the query processing, the server computes the response, and Figure 5.5 shows the trend of the response size under different database configurations. We observe that the response size for the Simple and Spiral protocols is much more reasonable than the substantial response size in Double. We discuss this in more detail below:

**Simple** We observe that Simple's response size equals to the query size, as shown in Table 4.5. However, the results may vary by a few KB due to the uneven division of an $N$-record database into a $\sqrt{N} \times \sqrt{N}$ matrix.

| Query processing time | Number of records | Simple Record size | Double Record size | Spiral Record size |
|---|---|---|---|---|
| under 0.5 s | $2^{14}$ | $\leq$ 256 KB | $\leq$ 16 KB | $\leq$ 1 KB |
| | $2^{17}$ | $\leq$ 32 KB | $\leq$ 8 KB | $\leq$ 128 bytes |
| | $2^{20}$ | $\leq$ 4 KB | $\leq$ 1 KB | None |
| under 1 s | $2^{18}$ | $\leq$ 32 KB | $\leq$ 16 KB | $\leq$ 256 B |
| | $2^{19}$ | $\leq$ 16 KB | $\leq$ 8KB | $\leq$ 256 B |
| | $2^{20}$ | $\leq$ 8KB | $\leq$ 4 KB | None |
| under 5 s | $2^{18}$ | $\leq$ 128 KB | $\leq$ 64 KB | $\leq$ 4 KB |
| | $2^{19}$ | $\leq$ 64 KB | $\leq$ 32 KB | $\leq$ 1 KB |
| | $2^{20}$ | $\leq$ 16 KB | $\leq$ 16 KB | $\leq$ 1 KB |
| under 10 s | $2^{18}$ | $\leq$ 32 KB | $\leq$ 128 KB | $\leq$ 16 KB |
| | $2^{19}$ | $\leq$ 16 KB | $\leq$ 64 KB | $\leq$ 4 KB |
| | $2^{20}$ | $\leq$ 32 KB | $\leq$ 32 KB | None |
| under 20 s | $2^{17}$ | $\leq$ 256 KB | $\leq$ 256 KB | $\leq$ 64 KB |
| | $2^{19}$ | $\leq$ 256 KB | $\leq$ 256 KB | $\leq$ 8 KB |
| | $2^{20}$ | $\leq$ 256 KB | $\leq$ 256 KB | $\leq$ 4 KB |

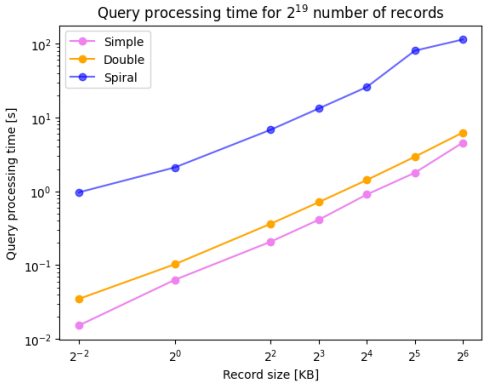


**Figure 5.4:** The left table shows the query processing time for various database configurations. The right plot shows the query processing time for $2^{19}$ records is displayed.

The response size depends on the number of rows, while the query size depends on the number of columns.

**Double** The difference in response size of Double and Simple is substantial, with Double reaching 3.7 GB for a $2^{19}\times$ 128 KB, while Simple only weighs in at 706 KB, yielding a 5000-fold difference. This correlation is in line with the findings in Table 4.5. The response size formula is devised by substituting fixed values from the paper, and we obtain 32 KB $\cdot\lceil r/\log p\rceil$, where $r$ is the record size, and $p$'s value is listed in Table 4.1. The minimum response size of Double is 32 KB and varies linearly in the number of field elements per record.

**Spiral** The query response is compacted using modulus switching. The size is determined by selecting $q_1$ and $q_2$, as seen in Table 4.5. The results show that the response size ranges from 11 KB for a $2^{14}\times 256$ B database to a maximum of 400 KB for database configurations $2^{14}-2^{17}\times 256$ KB.

### 5.2.5 Rate

We refer to the *rate* as the ratio between the plaintext record and the size of the (query) response. A high rate in a PIR scheme indicates efficient use of communication and storage resources. Conversely, a low rate indicates a significantly larger response than the record size, leading to higher communication costs.

Figure 5.5 shows the response sizes among the three protocols. It can be seen that as the record size increases, the rate also increases, which is the desired outcome. Our experimental results show that Double has a significantly low rate: the response size is at least $3\cdot 10^5$ times larger than the plaintext record. For a record size of 256 bytes, Double's rate is 100 times smaller than Simple

| *DB configuration* | Simple | Spiral |
|---|---|---|
| $2^{17}\times$ 256 KB | 0.27 | 0.64 |
| $2^{17}\times$ 128 KB | 0.143 | 0.64 |
| $2^{18}\times$ 8 KB | 0.04 | 0.4 |
| $2^{18}\times$ 128 KB | 0.139 | 0.64 |
| $2^{19}\times$ 256 B | 0.006 | 0.01 |
| $2^{19}\times$ 16 KB | 0.04 | 0.26 |
| $2^{20}\times$ 256 B | 0.004 | 0.01 |
| $2^{20}\times$ 8 KB | 0.02 | 0.13 |

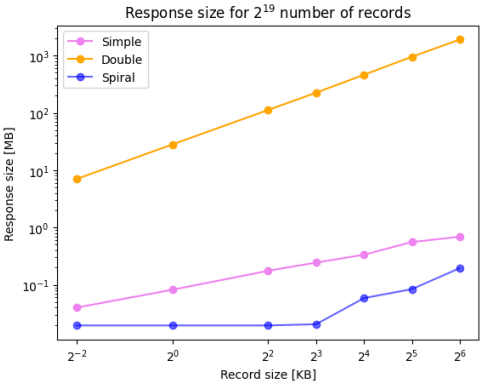


**Figure 5.5:** The left table presents the rates for Simple and Spiral. It should be noted that the rate for Double is fixed at $3 \cdot 10^{-5}$, which is why it was not included in the table.

or Spiral. Figure 5.2.5 also presents a table of rates of Simple and Spiral. It can be seen that the rate of Spiral is significantly higher than that of Simple; however, for both protocols, the rate is low for records smaller than 128 KB.

### 5.2.6 Decoding Time

We observe that the decoding time for all three protocols is very short and can be ignored. For Simple and Double, it is less than 1 microsecond, and for Spiral, it is less than 10 milliseconds, making it insignificant in terms of end-to-end latency.

## 5.3 Summary of Experimental Results

The hint size for Simple is typically in the range of hundreds of MB, while for Spiral, it is only in the tens of MB. These sizes are acceptable, making PIR a feasible option. However, the hint size for Double is always larger than the database size, measured in a few GB, making it impractical for use. The query sizes of all three protocols are sufficiently small for practice. The response sizes for both Simple and Spiral are similar to their query sizes, but Double's can reach substantial values, leading to a much lower rate compared with Simple and Spiral.

The generation of hints for Simple and Double leads to substantial query latency, making it the primary source of query latency for the first PIR query on a static database. The query processing time for Simple, Double, and Spiral is low when the record size is small, ranging from tens to hundreds of milliseconds. However, for larger record sizes above 1 KB, the processing time increases to minutes for Spiral or tens of seconds for Simple and Double.

In Chapter 6, we will examine how Simple and Double can compensate for the cost of their hint size and, in particular, their hint generation time in the presence of infrequent database updates and multiple clients.

Chapter 6

# Handling Database Updates and Multiple Clients

In the previous chapters, we evaluated the performance of Simple, Double, and Spiral with various database configurations, but the databases remained static. However, in many PIR applications, the database is subject to changes over time, and the frequency of these changes can vary depending on the specific use case. Two types of updates can occur: *modifications* to existing records in the database and the *addition or deletion* of records.

In this chapter, we examine how Simple, Double, and Spiral handle these changes and the resulting consequences. Furthermore, since multiple parties often access PIR applications simultaneously, we also investigate the scalability of these protocols in handling multiple clients.

## 6.1 Handling Database Updates

In both Simple and Double, any modification to an existing record requires a corresponding modification to the hint, as the hint is generated through the multiplication of a random LWE matrix and the database, as seen in Section 4.2.2.

In the case of the **Simple** protocol, the cost of recomputing the hint is proportional to the number of modified records, and the server only needs to send back the rows of the hint that have been modified rather than the entire hint, allowing the client to update the hint locally.

When considering a database with N records structured as a $\sqrt{N} \times \sqrt{N}$ matrix, with $n_d$ = 1024 as the lattice dimension, and $p$ as the plaintext modulus defined in Table 4.1, and assuming $c$ is the number of modified records, the cost of hint recomputation is $\sqrt{N}\lceil log(\frac{r}{\log(p)})\rceil \cdot n_d \cdot c$. However, it should be

noted that if $c \geq \sqrt{N}$, the size of the hint would be the same as in Table 4.5, requiring the entire hint to be recomputed.

For **Double**, the update of the client hint works similarly to Simple. The server needs to recompute the hint and send back a minimum of $c\lceil \log p / \log q \rceil \cdot n_d$ or $\lceil \log p / \log q \rceil \cdot n_d^2$ bits. However, if the number of modified records $c$ is greater than or equal to $n_d$, then the entire hint must be recomputed. Additionally, the server's stored hint must also be updated. The cost of recomputing the server's hint requires $c \cdot \sqrt{N\lceil \log(\frac{r}{\log(p)}) \rceil} \cdot n_d$ operations in $\mathbb{Z}_q$.

The addition and deletion of records are similarly handled in Simple and Double. Upon the addition of a record, the server will calculate the hint for the added record and send it to the client, who will then incorporate the added rows hint into the initial hint matrix. For a deletion, the client updates the corresponding row from the hint matrix.

For **Spiral**, as described in Section 4.2.2, the hint only depends on the dimensions of the database representation. As a result, modifications to records in the database do not require recomputing the hint. However, to handle record additions or deletions, since the dimension of the database representation changes, it is necessary for the client to recompute the hint.

## 6.2 Handling Multiple Clients

As discussed in Section 4.2.2, both Simple and Double require the server to compute the hint by multiplying a random LWE matrix $A$ with the database. When multiple clients use the application, the computational cost of generating the hint can be reduced by distributing the same matrix $A$ among the clients.

In contrast, Spiral requires the server to store a separate hint for each client, as it is based on individual keys. This can become impractical for the server when the number of clients grows, as it would require a large amount of storage to keep the individual hints. Although the size of the Spiral hint, as discussed in Section 5.1.1, is only in the tens of MB, it can still result in a significant overhead for a large number of clients. On the other hand, Simple and Double have larger hint sizes than Spiral, but since the hint is shared among clients, the overhead is reduced.

Chapter 7

# Applications

In this chapter, we delve into the practical applications of our experimental results of Simple, Double, and Spiral. Our focus is on four specific use cases that showcase the versatility and practicality of PIR.

The first use case we examine is a private article search on Wikipedia. Within this application, users can access Wikipedia articles without revealing to the server which articles they are accessing. The second use case we examine is a private DNS lookup service, allowing users to perform domain lookups without revealing the websites they are visiting. The third is a private search for Bitcoin balances, enabling users to search for a specific Bitcoin address's balances and recent transactions without revealing their identity or the address they are searching for. Finally, we investigate using PIR for Certificate Transparency (SCT auditing) from Simple and Double's paper [1]. This system enables users to verify the authenticity of TLS certificates without revealing for which certificates they are checking the signed certificate timestamp.

In Table 7.1, we present a comprehensive summary of the computational costs related to each protocol for each application. The cost of executing the first query (i.e the offline phase and the online phase) comprises the time for hint generation, query generation, and query processing. However, the cost of subsequent queries (i.e the online phase) only includes the time for query generation and processing. We also note that the decoding time is considered negligible.

## 7.1 Private Search on Wikipedia

A PIR-based Wikipedia article search system enhances privacy by keeping users' search queries and accessed articles confidential from the server.

The English version of Wikipedia is estimated to contain over 6 million

articles, which translates to approximately $2^{22}$. The size of an article on Wikipedia can vary, but an average text-only article is around 30 KB, resulting in a total database size of 120 GB.

As shown in Table 7.1, the cost of executing the first PIR query using Spiral is significantly faster than both Simple and Double. Spiral takes 31 seconds, while the other two take a few minutes. This cost can be divided among multiple clients in the setting described in Section 6.1. However, recomputing the hint may not be practical, resulting in long query latency for the subsequent queries, particularly for Spiral.

Hint size is also an important metric to consider. The hint for Double is massive, with a size of 460 GB, making it an unfeasible choice. On the other hand, the hint for Simple is 1 GB, which may be reasonable given the presence of multiple clients and limited database updates. Spiral's hint size is extremely small, at just 12 MB, providing a significant advantage in terms of efficiency and practicality.

## 7.2 Private DNS Lookup

A private DNS lookup service enhances privacy by allowing users to resolve domain names without revealing the websites they access.

Our estimation shows that a typical DNS server contains approximately 16,000 records, corresponding to $2^{14}$, with an average record size of 256 B. The query latency for all three protocols, Double, Simple, and Spiral is under 1 second. In terms of hint size, our research shows that Double has a significant size of 3 GB for a 4 MB database. Likewise, the hint sizes for Simple and Spiral are 7 MB and 8 MB, respectively, which are still larger than the database size. Therefore, Simple, Double, and Spiral are not practical for this specific application since the client could simply download the entire database and perform local queries.

## 7.3 Private Bitcoin Balance Search

A private Bitcoin balance search allows users to search for Bitcoin wallet addresses without revealing the addresses they search for to the server.

The number of active bitcoin addresses is estimated to be around 30 million. Each address is associated with recent transactions that have taken place within a limited time frame, which can be stored in a 512-byte record. The results of our analysis, presented in Table 7.1 indicate that the time for the first PIR query is relatively high for Simple and Double, taking several minutes, whereas Spiral only takes 2.4 seconds. However, as transactions occur in real-time, it is essential to evaluate the efficiency of these PIR protocols

under frequent database updates. This may, unfortunately, lead to a long querying time if the hint needs to be recomputed entirely.

Despite this, the online phase time of all protocols is close to 2 seconds, which is reasonable. In terms of hint size, we found in this particular database configuration, Double has a smaller hint size than the total database size, at 7.2 GB for a 16 GB database, while Simple's hint size is 484 MB, which is still 15 times smaller than that of Double. Meanwhile, the hint size for Spiral is estimated to be approximately 9 MB. Although this hint size is relatively small, it can still lead to substantial storage overhead if a large number of clients simultaneously use the application.

## 7.4 Certificate Transparency Private SCT Auditing

Certificate Transparency is a program that aims to provide a public log of public-key certificates issued by all certificate authorities. The certificates are submitted to log operators, who then provide a signed certificate timestamp (SCT), promising to log the certificate within a specified time. By verifying SCTs, clients can confirm that the certificate will eventually be logged by honest log operators.

Henzinger et al. [1] have demonstrated the effectiveness of Simple and Double for certificate auditing in a setting with 5 billion active SCTs, with 6 million being added or removed daily. Their approach incorporates Bloom filters [11] and uses PIR for private set membership, resulting in a database that stores only 1-bit entries indicating the presence of specific SCTs. The first PIR query computational time for Simple and Double is 4 and 12 seconds, respectively, which is acceptable considering it can be amortized across multiple clients. The authors recommend updating the hint periodically to reduce computation time and mention that the hint sizes for Simple and Double are 120 MB and 16 MB for a 1 GB database, respectively, with Double having a smaller client hint size than Simple. No results are presented for Spiral, as the parameter selection was made through an automatic tool, which resulted in a high variation of results in this case, making it impossible to make an accurate estimation.

| *Application* | | Simple | Double | Spiral |
|---|---|---|---|---|
| Private Wikipedia article search | 1st PIR query | 2 min | 4 min | 31 s |
| $2^{22}\times$ 30 KB | Following queries | 8 s | 9 s | 30 s |
| Private DNS lookup | 1st PIR query | 13 ms | 0.2 s | 0.4 s |
| $2^{14}\times$ 256 B | Following queries | 5 ms | 15 ms | 0.2 s |
| Private Bitcoin address search | 1st PIR query | 1 min | 2 min | 2.4 s |
| $2^{25}\times$ 512 B | Following queries | 1.8 s | 2.2 s | 2.3 s |
| Private SCT auditing | 1st PIR query | 3.7 s | 11.6 s | - |
| $2^{33}\times$ 1bit | Following queries | 0.1 s | 0.1 s | - |

**Table 7.1:** In the table, four applications of PIR are presented. The table displays the time for both the offline phase plus online phase, which is the time for the first PIR query and the online phase, which is the time for the following queries. It is important to note that Spiral does not have any results as it does not support databases larger than $2^{22}$.

Chapter 8

# Conclusion

In this work, we employed several key metrics to assess the efficiency of PIR schemes in terms of computational and communication costs. We evaluated the performance of three state-of-the-art single-server (offline-online) PIR protocols: Simple, Double, and Spiral, under various database configurations by changing the record size and the number of records. Our experimental results provide valuable insights into the efficiency and practicality of deploying these PIR schemes in real-world scenarios.

In terms of the *communication cost*, as analyzed in Chapter 5, we have evaluated the size of data transmitted between the client and the server. Our findings indicate that the hint size for Simple ranges from hundreds of MB, while the hint size for Spiral is only in tens of MB. These sizes are often small enough compared with the total database size considering the configurations where Simple's hint is smaller than the database size, making PIR a feasible option. On the other hand, the hint size for Double is always larger than the database size for the configurations we tested. Double's hint is at least measured in a few GB, making it unsuitable for practical use.

Regarding the *query size*, our results show that Simple varies between 7 KB for a $2^{14} \times$ 256 B database to 682 KB for a $2^{20} \times$ 32 KB database. Meanwhile, the query size for Spiral is fixed at 16 KB, and that of Double is fixed at 256 KB for most database configurations. These sizes are acceptable for most database configurations and applications. The *response size* follows a similar magnitude as the query size for both Simple and Spiral. However, for Double, it can reach significant sizes, such as 3.7 GB for a $2^{19} \times$ 128 KB database, resulting in a much lower rate for Double compared with Simple and Spiral.

In regards to the computational cost, we observed that generating hints for Simple and Double resulted in substantial query latency for some database configurations, but this may be tolerable in scenarios with infrequent database updates and when multiple clients are using the system, as the time can be

spread out. Query generation time only constitutes a small portion of the total latency. However, when it comes to query processing time, we have seen that Simple, Double, and Spiral have low processing times (in the range of tens to hundreds of milliseconds) when the record size is small ($\leq$ 1 KB), but for Spiral, processing time increases to minutes when the record size is larger than 1 KB. Meanwhile, Simple and Double reach tens of seconds of processing time.

In conclusion, our findings indicate that the practicality of PIR-based applications is primarily impacted by the query processing time and the hint exchange phase. Our experimental results and case studies demonstrate that for small record sizes less than 1 KB, Simple, Double, and Spiral perform well, even with large numbers of records, e.g., billions of 1-bit entries as in the SCT auditing for the Certificate Transparency application [1]. However, as record size increases, we observe that both Simple and Double scale better than Spiral, but still has significant query processing times. These results emphasize the need for further optimizations and advancements in PIR schemes to handle queries efficiently in a database with large records.

Appendix A

# Appendix

## A.1 Additional Algorithms

We present below the algorithms used in Section 4.3.1 to convert Regev scalars to Regev matrices (ScaltToMat) and to GSW matrices (RegevToGSW):

**ScalToMat**: This method allows us to expand a Regev encoding of a scalar $\mu$ into a Regev matrix encoding $\mu I_n \in R_q^{n \times n}$ and relies on the conversion key $W$ defined in Section 4.2.2. On input key $W$ from the client's public parameters, the packed Regev encoding $c = (c_0, c_1) \in R_q^2$ of a scalar and a gadget matrix $G_{n,z_{conv}}^{-1}$.

We compute $WG_{n,z}^{-1}(c_0 I_n) + \begin{bmatrix} 0^n \\ c_1 I_n \end{bmatrix}$

**RegevToGSW**: This method allows us to construct a GSW encoding of a message $\mu \in R_q$ with decomposition base $z_{GSW}$ from a collection of scalar Regev encodings, using the conversion keys $ck = (V, W, \Pi)$ defined in 4.2.2. On input of the conversion keys and $c_1, \dots c_{t_{GSW}} \in R_q^2$. For each $c_i$, $i$ in $[t_{GSW}]$, compute $C_i$ = ScalToMat($W$, $c_i$). Then, output $C = \left[Vg_{z_{conv}}^{-1}(\hat{C})|C_1|\dots|C_{t_{GSW}}\right] \cdot \Pi$, with $C = \left[c_1|\dots|c_{t_{GSW}}\right] \in R_q^{2 \times t_{GSW}}$.

# Bibliography


[1] Alexandra Henzinger, Henry Corrigan-Gibbs, Matthew M. Hong, Sarah Meiklejohn, Vinod Vaikuntanathan. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *Usenix Security*, 2023.

[2] Samir Jordan Menon and David J. Wu. Spiral: Fast, high-rate single-server PIR via FHE composition. In *IEEE S&P*, 2022.

[3] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, 2013.

[4] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In FOCS, 1995.

[5] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *CCS*, 2016.

[6] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In *EUROCRYPT*, 2014.

[7] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO*, 2012.

[8] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.* , 2012.

[9] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. In *IACR Cryptol. ePrint Arch.*, 2018.

[10] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: fast fully homomorphic encryption over the torus. *J. Cryptol.*, 33(1), 2020.

[11] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970.