November 15, 2023

Capstone Project: Learning/Living With Errors and Private Information Retreival

Aakanksha
aakanksha.agrawal_asp24@ashoka.edu.in
Ashoka University

Abstract

Can we design an encrypted version of Google's search engine that would enable users to search the Internet privately without revealing their queries to Google? Motivated by this goal, I replicate the paper **"One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval"** [1]. I first present the concept of Private Information Retrieval. Then, I review the Learning With Errors Problem introduced by Oded Regev. Then, I review a single server PIR scheme based on it through a toy example introduced by Regev in 2005. Finally, I review SimplePIR-the fastest known PIR scheme to date, from the paper. The paper also introduces Double PIR and applies it to certificate transparency, which I plan to do as part of my thesis.

Contents

Dedication

To Ma, Pops, and Nani.

Acknowledgement

"It's not hard, it's just complicated. It will take you a few weeks to figure it out" - Professor Debayan Gupta

I would like to thank my advisor **Professor Debayan Gupta**, for all his guidance throughout this challenging semester and for making me fall in love with cryptography over two years ago. His input and insight through all our countless conversations have been invaluable.

I would like to thank my **mother** and my **father** for always supporting me and being my strength, and **nani** for being my inspiration to choose this path. I would also like to thank **Cocoa K** for helping me work on math and being patient with me.

I would also like to thank my dearest friends **Sushi**, **Shivani**, **Gaurang**, **Soumya**, **Sid**, **Mini** and **Arshiya** for being there for me and helping me emotionally and physically while I tried completing this capstone. Their contribution to this thesis cannot be understated, and I would like to extend my deepest gratitude to them.

I would also like to thank **Professor Mahavir Jhawar** for his excellent course on cryptography.

1 Introduction

1.1 Private Information retreival

Private Information Retrieval (PIR) is a cryptographic protocol introduced in 1995 by Chor, Goldreich, Kushilevitz, and Sudan [2] in the information-theoretic setting and in 1997 by Kushilevitz and Ostrovsky [3] in the computational setting. It allows a client to retrieve a specific record from a database server without disclosing which record they are interested in. This has various applications in places where the client's query has sensitive information about the client itself. For example

1. Querying a medical database: The query would contain information about the condition the client has.
2. Querying a financial database: The query would contain information about the client's financial interests.
3. DNS: The query would contain the website the client wants to visit.

PIR has applications in various privacy enhancing technologies(PETs) such as private database searches [4], [5], metadata-protected messaging [6], [7], private media access [8], secure reporting of credential breaches [9], private contact discovery, privacy-conscious advertising [10], and confidential blocklist checks [11].

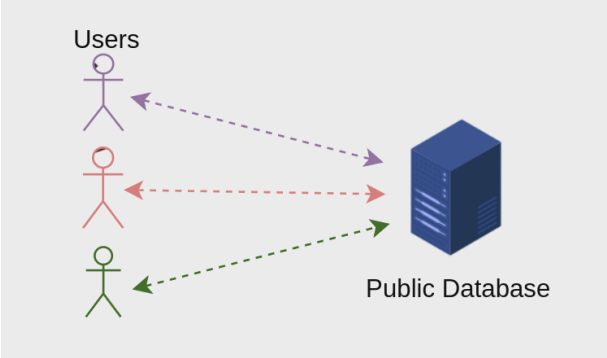


Figure 1: PIR protocol

1.2 PIR requirements

There are 2 main requirements for a PIR protocol, described here informally.
Correctness:
The client learns it's bit of interest always or with an overwhelmingly high probability.

Security:
(Malicious) server "learns nothing" about the client's desired bit.

1.3 Approaches to PIR

1.3.1 Naive Solution

The naive solution that fulfills both these requirements is that the client downloads the entire database, stores it locally, and queries it, never contacting the server again. This achieves perfect correctness and privacy. However, the downside is a significant communication cost, which grows with the size of the database. So we need a better approach.

1.3.2 Multi Server PIR

Informally, to achieve "secrecy," the key idea is to secret share the query between 2 or more non-colluding servers. This was also introduced by Chor, Goldreich, Kushilevitz and Sudan [3] in the same paper.

Consider two servers that are non-colluding. Syntactically, a multi-server PIR protocol consists of three efficient algorithms:

1. **Query**$(1^n, i) \to (q_0, q_1)$

The user runs the query algorithm. He inputs the desired index of the database he wants to read and the algorithm returns 2 queries, one for each of the servers.

The user sends those queries to each of the servers
2. **Answer**$(x, q) \to a$

Each of the two servers run the answer algorithm. The algorithm takes as input the database x and the query q, and outputs the answer.
The answer is sent back to the user.
3. **Reconstruct**$(a_0, a_1) \to x_i$

The user runs the reconstruct algorithm that takes as input both the server's answers and outputs the value of the desired bit.

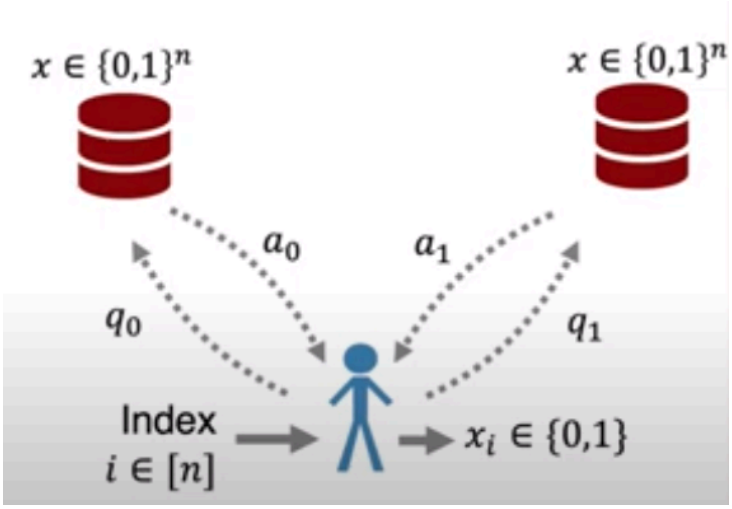


Figure 2: A multi-server PIR scheme [12]

1.3.2.1 Security(Formally)

We can now state the requirements formally. In a 2-server setting, we require:

Correctness:
Client learns it's bit of interest: $\forall n \in N, x \in \{0,1\}^n, i \in [n]$:

$$\Pr\{\text{Reconstruct}(a_0, a_1) = x_i : q_0, q_1 \leftarrow \text{Query}(1^n, i)$$
$$a_0 \leftarrow \text{Answer}(x, q_0)$$
$$a_1 \leftarrow \text{Answer}(x, q_1)\} = 1$$

Security:
Server learns nothing about the client's desired bit.

$$\forall n \in \mathbb{N}, i, j \in [n], s \in \{0,1\},$$

$$\{\boldsymbol{q_s} : q_0, q_1 \leftarrow \text{Query}(1^n, i)\} \underset{c}{\approx} \{\boldsymbol{q_s} : q_0, q_1 \leftarrow \text{Query}(1^n, j)\}$$

$$\{\text{View of server s when client reads bit i}\} \underset{c}{\approx} \{\text{View of server s when client reads bit j}\}$$

1.3.2.2 Drawbacks

Drawbacks for a multiserver PIR scheme are
1. cumbersome to deploy and requires non-colluding independent infrastructure providers
2. security relies on a non-colluding assumption, not cryptographic hardness

1.3.3 Single Server PIR

This problem was introduced in a single server setting by Kushilevitz and Ostrofsky in 1997 [3]. The key idea is to encrypt the query instead of secret sharing it. We use Linearly Homomorphic Encryption to encrypt it.

Linearly Homomorphic Encryption(LHE):
$\text{Enc}(k, m_1) + \text{Enc}(k, m_2) = \text{Enc}(k, m_1 + m_2)$

This LHE scheme can be built from a wide range of public key assumptions like Decisional Diffie Hellman, Quadratic Residuosity, Learning With Errors, etc.

Syntactically, a single-server PIR protocol consists of 3 steps:
1. $q = \text{Enc}(k, e_j)$

The client generates an encrypted query vector q, under the key k and sends it to the server
2. $a = Xq$

The server calculates the matrix-vector product between the database X and the query vector q, and outputs an encrypted response. 3.Output $\text{Dec}(k, a_i)$ The client extracts the i^{th} element(a_i) from the encrypted response and decrypts it under the key k

1.3.3.1 Security

The security of the protocol follows from the security of the assumption in the LHE scheme.

1.4 Simple PIR

In the SimplePIR scheme, the key insight is that using Regev's learning-with-errors-based encryption scheme, the server can preprocess the majority of the work involved in computing the matrix-vector product before receiving the client's query. This preprocessing, dependent only on the database and the public parameters of the encryption scheme, can be reused for multiple queries from various clients. The main drawback is the requirement for the client to download a hint about the database after preprocessing, which constitutes the majority of the communication cost in SimplePIR.

1.5 Double PIR

DoublePIR, an advanced concept from the Kushilevitz and Ostrovsky paper, builds upon SimplePIR. It involves the client downloading a hint and an encrypted vector but only using a small part of both for record retrieval. This method uses a recursive application of SimplePIR, significantly reducing communication costs. The technique is optimized to save computational resources, notably improving efficiency over basic designs.

I plan to do this as part of my thesis

1.6 Application to Certificate Transparency

The paper also explores the application of these PIR schemes to Certificate Transparency, specifically for auditing signed certificate timestamps. This process involves checking if a specific string (an SCT) is present in a set, while keeping the string confidential. The paper introduces a new data structure that improves the efficiency of private set-membership queries, allowing for a significant speedup compared to traditional methods.

Comparatively, Google's current method for SCT auditing in Chrome provides limited privacy, while the proposed solution offers full cryptographic privacy with more communication and computational requirements but significantly enhances privacy and data security.

I plan to do this as part of my thesis.

2 Learning with errors

The "Learning with errors" ("LWE") problem was introduced by Oded Regev [13] in 2005. It's an "encryption-enabling" analogue of the SIS problem. He then gave us a public key enryption system based on it. Ostrofsky used it to build a private information retrieval protocol based on it .

2.1.1 Search LWE

Say they give us A and the matrix-vector product b=As. Can we determine s?

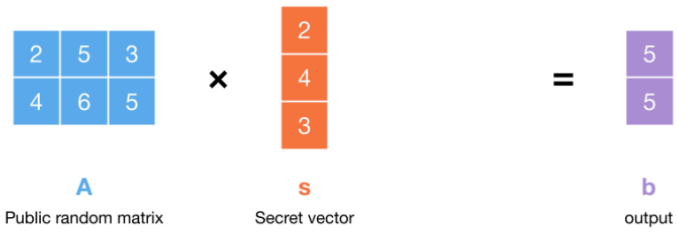


Figure 3: [14]

)

The answer is, with overwhelming probability, yes, it will be easy to determine s. We can compute the matrix inverse of A , and then multiply this by b. This will give us s since $A^{-1}A = I$ and

$$A^{-1}b = A^{-1}As = s$$

. The matrix inverse will exist as long as the determinant of A is not zero, which is almost certainly true, since A was chosen uniformly at random. Computing the matrix inverse is also not difficult, so this is not a 'hard' problem, and so it's not one we can base cryptography on.

Things get more interesting when we make a small change to the problem. Instead of sending us $b = As$, what if instead they make things harder by injecting some error into b? Specifically, if they send us $b = As + e$, for some error vector e, the problem becomes surprisingly hard.

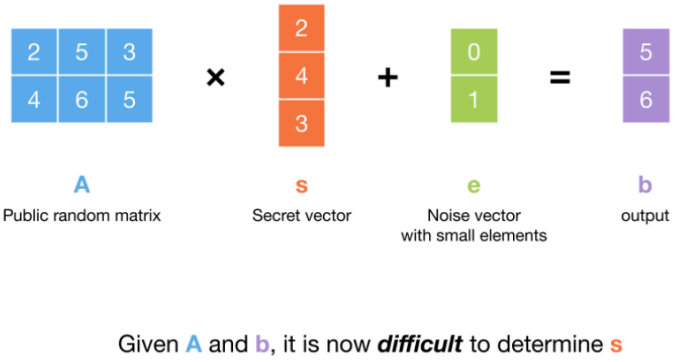


Figure 4: [14]

This is because multiplying by the matrix inverse doesn't work well in the presence of even a small amount of error. The error can be sampled in a number of ways, but for now, imagine that it is chosen uniformly over a small range (much less than q).

This problem is called **"learning with errors"**, or LWE since we are trying to learn s given some data that contains errors. Each row of the output can be called an "LWE sample", since it is a noisy 'observation' of the dot product of a row of A and s. The challenge is to determine s from at most m LWE samples. This is considered hard to solve, even for small parameters. Example: For n = 512, q = 3329, and error in [−3, 3], the problem is hard

The LWE challenge:

1. Generate random s and a vectors in range [0, 3328].
2. Choose random error e from {3326, . . . , 3}.
3. Compute b = a · s + e.
4. Output pair (a, b). Repeat for multiple samples.
5. Goal: Determine s from these samples.

2.1.2 Decision LWE

- Given (A, b), distinguish if b is from $b = As + e$ (LWE way) or randomly sampled.
- Cryptographers hypothesize that distinguishing correctly is extremely hard (probability < 2−128) for certain parameter choices.

```
def sample_true():
    A = random_matrix(n, n)
    s = random_vector(n)
    e = random_noise_vector(n)
    b = A * s + e
    return (A, b)

def sample_random():
    A = random_matrix(n, n)
    b = random_vector(n)
    return (A, b)
```

- It is "hard" to tell the difference between the outputs of **sample_true()** and **sample_random()**.

$\{A, (As + e)\} \underset{c}{\approx} \{A, r\}$.

2.2 Formal Definitions

LWE is parameterized by positive integers n and q, and an error distribution χ over $\mathbb{Z}$. [15]

2.2.1 Definition (LWE distribution)

For a vector $s \in \mathbb{Z}_n^q$ called the secret, the LWE distribution As,χ over $\mathbb{Z}_n^q \times \mathbb{Z}_q$ is sampled by choosing $a \in \mathbb{Z}_n^q$ uniformly at random, choosing $e \leftarrow \chi$, and outputting $(a, b = as + e \bmod q)$

There are two main versions of the LWE problem: *search*, which is to find the secret given LWE samples, and *decision*, which is to distinguish between LWE samples and uniformly random ones. We additionally parameterize these problems by the number m of available samples, which we typically take to be large enough that the secret is uniquely defined with high probability.

2.2.2 Definition(Search-LWE$_{n,q,\chi,m}$).

Given m independent samples $(a_i, b_i) \in \mathbb{Z}_n^q \times \mathbb{Z}_q$ drawn from A_s, χ for a uniformly random $s \in Z_q^n$ (fixed for all samples), find s.

2.2.3 Definition(Decision-LWE$_{n,q,\chi,m}$).

Given m independent samples $(a_i, b_i) \in \mathbb{Z}_n^q \times \mathbb{Z}_q$where every sample is distributed according to either: (1) As, χ for a uniformly random $s \in \mathbb{Z}_q^n$ (fixed for all samples), or (2) the uniform distribution, distinguish which is the case (with non-negligible advantage).

2.3 Secret-key Regev encryption

Regev [13] gives a secret-key encryption scheme that is secure under the LWE assumption. With LWE parameters (n, q, χ) and a plaintext modulus p the Regev secret key is a vector $s \leftarrow \mathbb{Z}_q^n$. The Regev encryption of a message $\mu \in \mathbb{Z}_p$ is

$$(a, c) = \left(a, a^t s + e + \left\lfloor \frac{q}{p} \right\rfloor \cdot \mu\right) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$$

,

for $e \leftarrow \chi$. To decrypt the ciphertext, anyone who knows the secret s can compute $c - a^t s \bmod q$ and round the result to the nearest multiple of $\lfloor \frac{q}{p} \rfloor$. Decryption succeeds as long as the absolute value of the error sampled from the error distribution $\chi < 12 \lfloor \frac{q}{p} \rfloor$. We say that a setting of the Regev parameters supports *correctness error* δ if the probability of a decryption error is at most δ (over the encryption algorithm's randomness).

Regev encryption is additively homomorphic, since given two ciphertexts (a_1, c_1) and (a_2, c_2), their sum $(a_1 + a_2, c_1 + c_2)$ decrypts to the sum of the plaintexts, provided again that the error remains sufficiently small

3 Private Information Retreival

3.1 Classic PIR/technical ideas

The simplest non-trivial PIR schemes take the following "square-root" approach [3]:

3.1.1 Setup Database

Given an N element database, the server stores this as a $\sqrt{N}$-by$\sqrt{N}$ square matrix.

$$D = \begin{pmatrix} d_{11} & d_{12} & \dots & d_{1\sqrt{n}} \\ d_{21} & d_{22} & \dots & d_{2\sqrt{n}} \\ \vdots & \vdots & \ddots & \vdots \\ d_{\sqrt{n}1} & \dots & \dots & d_{\sqrt{n}\sqrt{n}} \end{pmatrix}_{\sqrt{N} \times \sqrt{N}}$$

Toy Example:
Let's make a 2×2 matrix:
$D = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix}_{\sqrt{N} \times \sqrt{N}}$

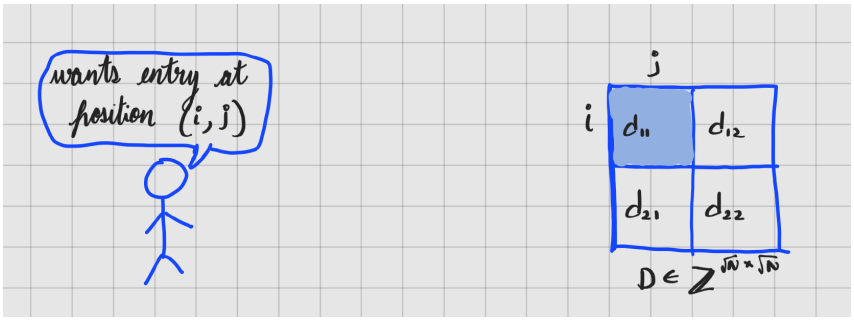


Figure 6: Given an N element database, the server stores this as a $\sqrt{N}$-by$\sqrt{N}$ square matrix.

3.1.2 Client: Build query vector

3.1.2.1 One hot encode the query

The client who wishes to query for a database entry at $i \in [N]$ decomposes i into a pair of coordinates $(i_{row}, i_{col}) \in \sqrt{N}$

Client builds a (one hot encoded) unit vector $\mu = u_{i_{col}} \in \mathbb{Z}_2^{\sqrt{N}}$(i.e, the vector of all zeroes with a single "1" at index i_{col}), element-wise encrypts it with a linearly homomorphic encryption scheme and sends this encrypted vector to the server.

We want to retrieve $x = d_{12}$, so $(i_{row}, i_{col}) = (1, 2)$

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}_{\sqrt{N}\times 1}$$

Let's element wise encrypt this with a linearly homomorphic scheme

3.1.2.2 Regev encryption-Sample LWE

The security of our scheme relies on the decesion varient of the LWE assumption[86]. The assumption is parametrized by the dimension of the LWE secret $n \in \mathbb{N}$, number of samples $m \in \mathbb{N}$, the integer modulus $q \geq 2$ and an error distribution χ over $\mathbb{Z}$

LWE(n, q, χ)

Given the size of database is N, set (number of samples) $\boldsymbol{m = \sqrt{N}}$

Sample: A matrix $A \leftarrow \mathbb{Z}_q^{\sqrt{N}\times n}$, a secret vector $s \leftarrow \mathbb{Z}_q^n$, an error vector $e \leftarrow \chi^m$
Output: $\left\{\left(\boldsymbol{A} \in \mathbb{Z}_q^{\sqrt{N}\times n}, (\boldsymbol{A}s + e) \in \mathbb{Z}_q^m\right)\right\}$

$(A, b, s) \leftarrow \pi_1.\text{KeyGen}\left(n, q, m = \sqrt{N}, \chi\right)$

Toy Example:

Sample: $A = \begin{pmatrix} a_1 \in \mathbb{Z}_p^{1\times n} \\ a_2 \in \mathbb{Z}_p^{1\times n} \end{pmatrix}_{\sqrt{N}\times n} = \begin{pmatrix} a_{11} & a_{12} & \dots & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & \dots & a_{2n} \end{pmatrix}_{\sqrt{N}\times n}$ $s = \begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ s_n \end{pmatrix}_{n\times 1}$ $e = \begin{pmatrix} e_1 \\ e_2 \end{pmatrix}_{\sqrt{N}\times 1}$

Compute: $b = \boldsymbol{A}s + e = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}_{\sqrt{N}\times 1} = \begin{pmatrix} a_1 s + e_1 \\ a_2 s + e_2 \end{pmatrix}_{\sqrt{N}\times 1}$

Output: $\left[A = \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}_{\sqrt{N}\times n}, b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}_{\sqrt{N}\times 1}\right]$

3.1.2.3 Regev encryption- Element Wise Homomorphically Encrypt μ

$\mathbb{Z}_q^{\sqrt{N}\times n}, \mathbb{Z}_q^{\sqrt{N}}, \mathbb{Z}_2^{\sqrt{N}} \rightarrow \mathbb{Z}_q^{\sqrt{N}\times n}, \mathbb{Z}_q^m$
$Enc(A, b, \mu) \rightarrow \left(A \in \mathbb{Z}_q^{\sqrt{N}\times n}, \boldsymbol{\mu_{enc}} \in \mathbb{Z}_q^m\right)$

$(A, \mu_{enc}) \leftarrow \pi_1.\text{Enc}(A, b, \mu)$

Toy example:

$$\mu_{enc} = \left[A_{\sqrt{N}\times n} \in \mathbb{Z}_p^{\sqrt{N}\times n}, c_{\sqrt{N}\times 1} \in \mathbb{Z}_p^{\sqrt{N}}\right]$$
$$= \left[A_{\sqrt{N}\times n}, c = \begin{pmatrix}\mu_1\\ \mu_2\end{pmatrix}_{enc}\right] = \left[A_{\sqrt{N}\times n}, c = \begin{pmatrix}0\\ 1\end{pmatrix}_{enc}\right]$$
$$= \left[A_{\sqrt{N}\times n}, c = \begin{pmatrix}b_1+\mu_1\\ b_2+\mu_2\end{pmatrix}\right]$$
$$= \left[A_{\sqrt{N}\times n}, c = \begin{pmatrix}a_1 s+e_1+\lfloor\frac{q}{p}\rfloor\mu_1\\ a_2 s+e_2+\lfloor\frac{q}{p}\rfloor\mu_2\end{pmatrix}_{\sqrt{N}\times 1}\right]$$

3.1.3 (Client → Server) Send $\mu_{enc} = (A, c) \in \left(\mathbb{Z}_p^{\sqrt{N}\times n}, \mathbb{Z}_q^{\sqrt{N}}\right)$ to server

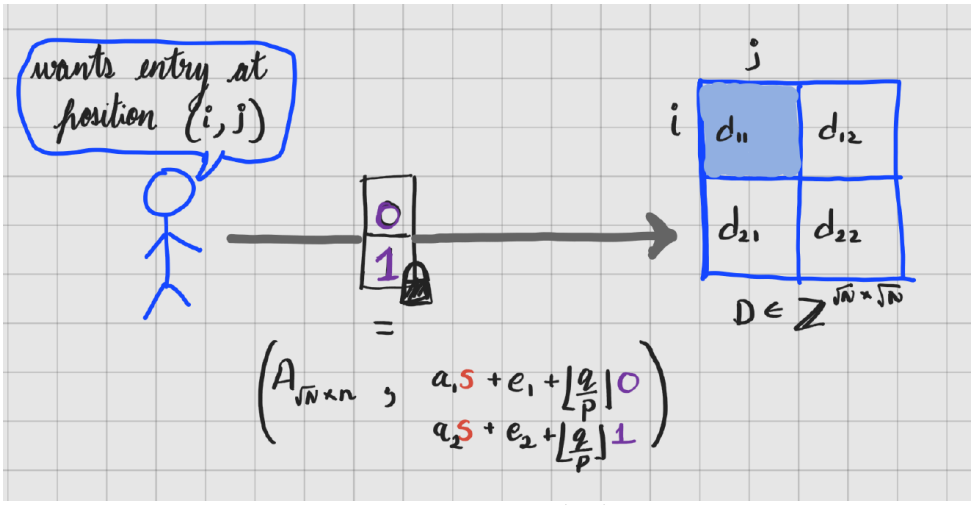


Figure 7: Client $\xrightarrow{\mu_{enc}=(A,c)}$ Server

Communication cost: Client uploads $\sqrt{N}n + \sqrt{N}$ elements in $\mathbb{Z}_p$

3.1.4 Server: Server computes $(D\mu)_{enc} = D\mu_{enc}$ and sends to client :

$D \cdot \mu_{enc} = \left[(D \cdot A) \in \mathbb{Z}_q^{\sqrt{N}\times n}, (D \cdot c) \in Z_q^m\right] = (D\mu)_{enc}$

Toy Example: Computing $(\boldsymbol{D\mu})_{\boldsymbol{enc}}$

$$D \cdot c = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix}_{\sqrt{N}\times\sqrt{N}} \cdot \begin{pmatrix} c_1 = a_1 s + e_1 + \lfloor \frac{q}{p} \rfloor \mu_1 \\ c_2 = a_2 s + e_2 + \lfloor \frac{q}{p} \rfloor \mu_2 \end{pmatrix}_{\sqrt{N}\times 1}$$

$$= \begin{pmatrix} d_{11}c_1 + d_{12}c_2 \\ d_{21}c_1 + d_{22}c_2 \end{pmatrix}_{\sqrt{N}\times 1 = \sqrt{N}\times 1}$$

$$D \cdot A = \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix}_{\sqrt{N}\times\sqrt{N}} \cdot \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}_{\sqrt{N}\times n}$$

$$= \begin{pmatrix} d_{11} & d_{12} \\ d_{21} & d_{22} \end{pmatrix}_{\sqrt{N}\times\sqrt{N}} \cdot \begin{pmatrix} a_{11} & a_{12} & \dots & \dots & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & \dots & \dots & a_{2n} \end{pmatrix}_{\sqrt{N}\times n}$$

$$= \begin{pmatrix} d_{11}a_{11} + d_{12}a_{21} & d_{11}a_{12} + d_{12}a_{22} & \dots & \dots & \dots & d_{11}a_{1n} + d_{12}a_{2n} \\ d_{21}a_{11} + d_{22}a_{21} & d_{21}a_{12} + d_{22}a_{22} & \dots & \dots & \dots & d_{21}a_{1n} + d_{22}a_{2n} \end{pmatrix}_{\sqrt{N}\times n}$$

Output : $(D\mu_{enc}) = \left(DA \in \mathbb{Z}_q^{\sqrt{N}\times n}, Dc \in \mathbb{Z}_q^{\sqrt{N}}\right)$

Computational costs:

DA:

1. Calculating one element a_{ij} of matrix-matrix multiplication requires $\sqrt{N}$ multiplication operations in Z_q and $\sqrt{N-1}$ addition operations in $\mathbb{Z}_q$ so $\left(\sqrt{N} + \sqrt{N} - 1\right) \approx 2\sqrt{N}$ total operations.
2. To calculate the entire row that has n elements, it would take $n\left(2\sqrt{N}\right)$ operations.
3. To calculate all the rows(number of rows $= \sqrt{N}$), it would take $\sqrt{N}\left(n2\sqrt{N}\right) \approx 2nN$ operations in $\mathbb{Z}_q$.

Dc:

1. To calculate the first element, it would take $\sqrt{N}$ integer multiplications in $\mathbb{Z}_q$, and $\sqrt{N} - 1 \approx \sqrt{N}$ additions in $\mathbb{Z}_q$.

2.Total number of operations to caculate one element $= \sqrt{N} + \sqrt{N} - 1) = 2\sqrt{N} - 1 \approx 2\sqrt{N}$,

3. To compute all rows (number of rows is $\sqrt{N}$), it would take $\sqrt{N}\left(2\sqrt{N}\right) = 2N$ operations in $\mathbb{Z}_q$

3.1.5 (Client ← Server) Server sends $(D\mu)_{enc}$ to client

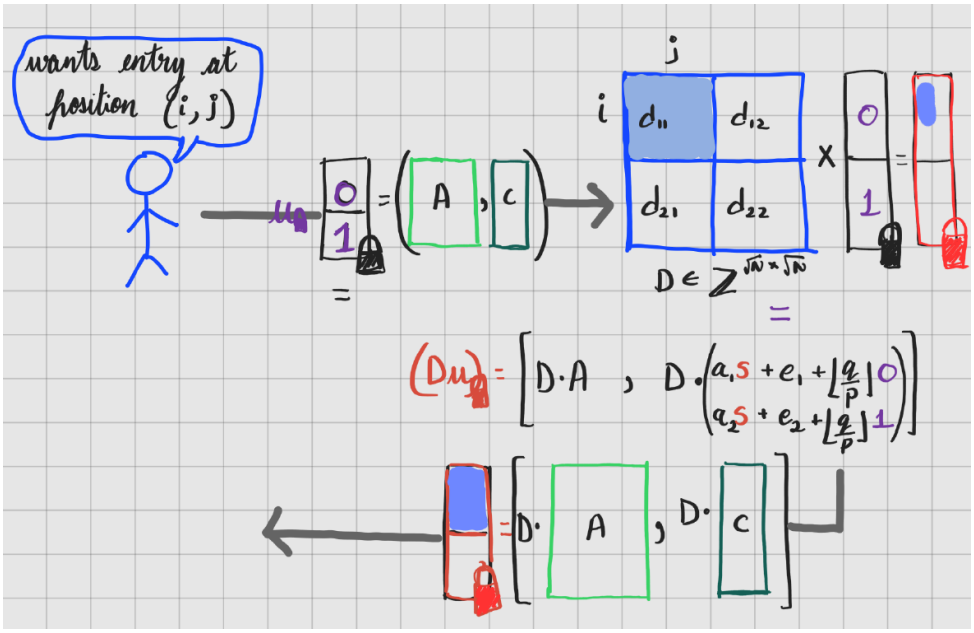


Figure 8:

$$\text{Client} \xleftarrow{(D\mu)_{enc}=(DA,Dc)} \text{Server}$$

Communication cost:

DA: Client receives $n\sqrt{N}$ elements in $\mathbb{Z}_q$

Dc: Client downloads $\sqrt{N}$ elements in $\mathbb{Z}_q$

3.1.6 Client: Client extracts $(D\mu)_{[i_{row}]}$ and decrypts with secret key s

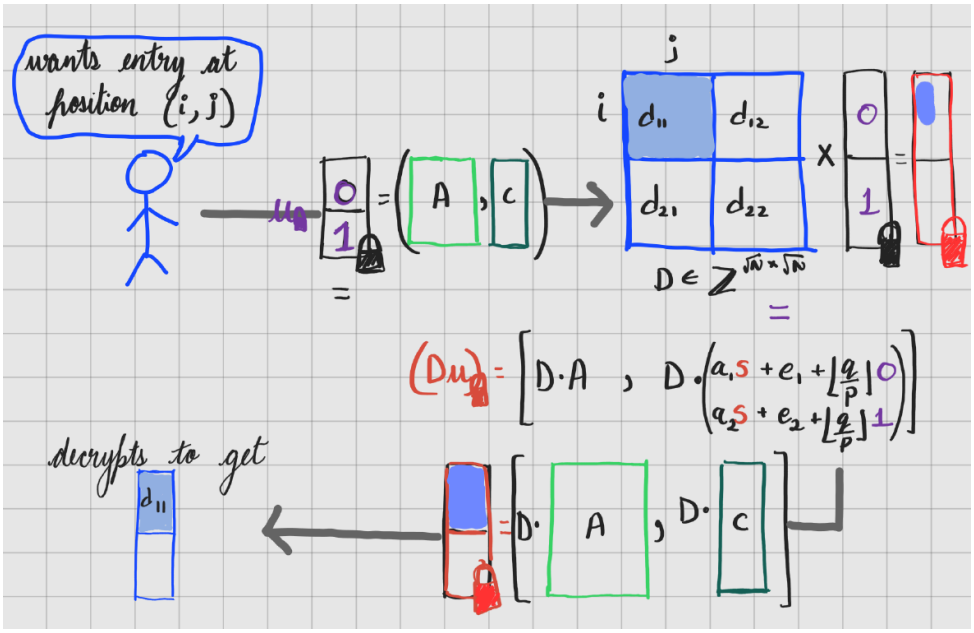


Figure 9: Client extracts $(D\mu)_1$ and decrypts with secret key s

$(D\mu)_{(enc)_1} = (DA)_1, (Dc)_1$ and compute $\hat{x} = (Dc)_1 - s(DA)_1$

1. What exacty is $(Dc)_{i_{row}}$?

$$
\begin{aligned}
(Dc)_1 &= (d_{11}c_1 + d_{12}c_2) \\
&= d_{11}\Big(a_1 s + e_1 + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_1\Big) + d_{12}(a_2 s) + e_2 + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_2) \\
&= \Big(d_{11}a_1 s + d_{11}e_1 + d_{11}\Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_1\Big) + \Big(d_{12}a_2 s + d_{12}e_2 + d_{12}\Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_2\Big) \\
&= (d_{11}a_1 + d_{12}a_2)s + d_{11}e_1 + d_{12}e_2 + \Big\lfloor \tfrac{q}{p} \Big\rfloor (d_{11}\mu_1 + d_{12}\mu_2) \\
&= \begin{pmatrix} d_{11} & d_{12} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} s + d_{11}e_1 + d_{12}e_2 + \Big\lfloor \tfrac{q}{p} \Big\rfloor (d_{11}\mu_1 + d_{12}\mu_2) \\
&= (\boldsymbol{DA})^{\boldsymbol{T}}_{\boldsymbol{1}} \cdot s + d_{11}e_1 + d_{12}e_2 + \Big\lfloor \tfrac{q}{p} \Big\rfloor (d_{11}\mu_1 + d_{12}\mu_2)
\end{aligned}
$$

In general,

$$
(Dc)_{i_{row}} = s \cdot (\boldsymbol{DA})^{\boldsymbol{T}}_{\boldsymbol{i_{row}}} + \sum_{v=1}^{m} d_{\boldsymbol{i_{row}}v} e_v + \left\lfloor \frac{q}{p} \right\rfloor \sum_{v=1}^{m} \left(d_{\boldsymbol{i_{row}}v} \mu_v \right)
$$

In genral, the matrix vector product $D \cdot c$ is:

$$
D \cdot c = \begin{pmatrix}
d_{11}\Big(a_1 s + e_1 + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_1\Big) + d_{12}\Big(a_2 s + e_2 + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_2 + \ldots + d_{1m}\Big(a_m s + e_m + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_m\Big)\Big) \\
d_{21}\Big(a_1 s + e_1 + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_1\Big) + d_{22}\Big(a_2 s + e_2 + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_2\Big) + \ldots + d_{2m}\Big(a_m s + e_m + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_m\Big) \\
\vdots \\
\vdots \\
d_{m1}\Big(a_1 s + e_1 + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_1\Big) + d_{m2}\Big(a_2 s + e_2 + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_2\Big) + \ldots + d_{mm}\Big(a_m s + e_m + \Big\lfloor \tfrac{q}{p} \Big\rfloor \mu_m\Big)
\end{pmatrix}_{\sqrt{N} \times 1}
$$

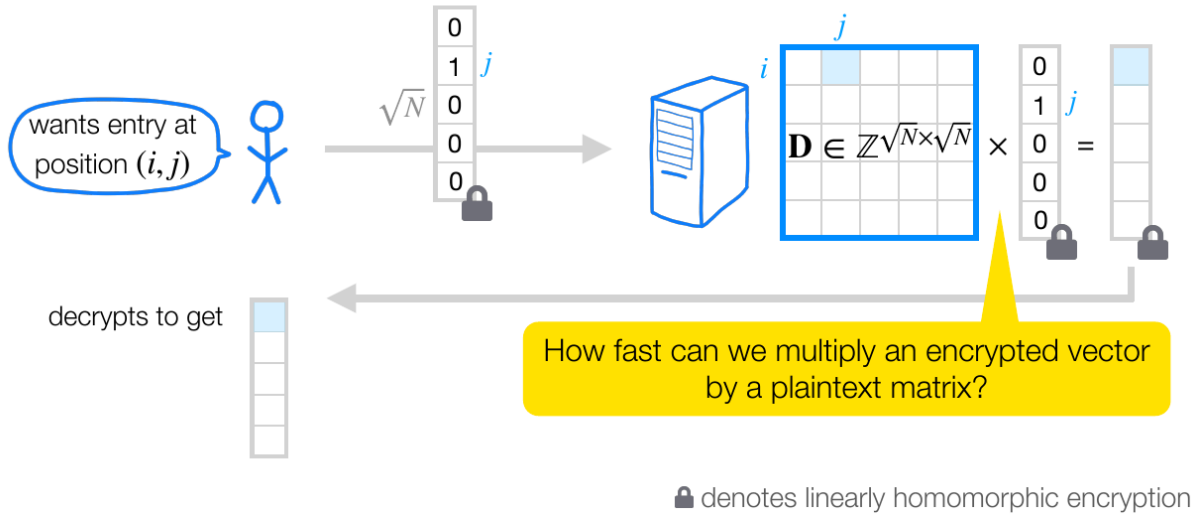
Decryption:

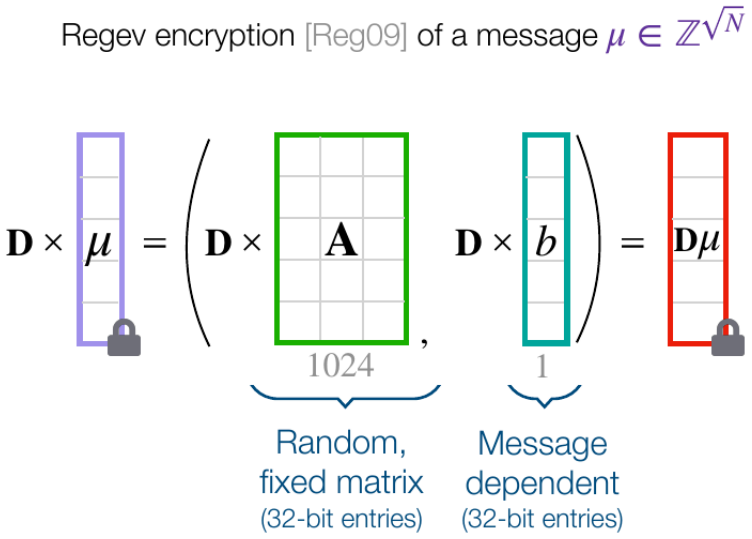
$$\triangle = \left\lfloor \frac{q}{p} \right\rfloor \in \mathbb{Z}$$

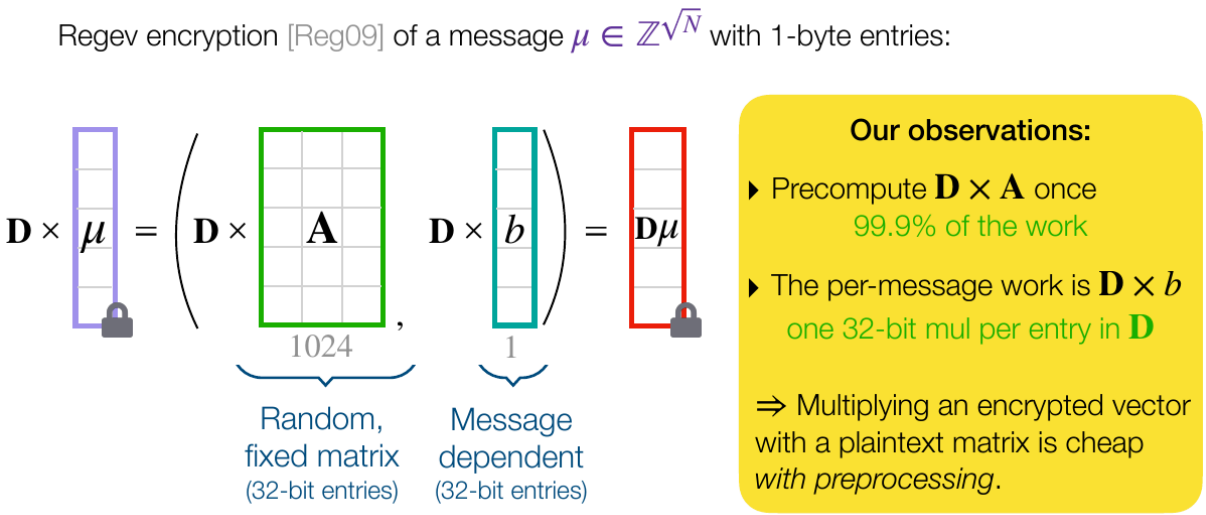
Compute $\hat{x} = \left[(Dc)_{i_{row}} - s(DA)^T_{i_{row}} \right]$ and Return $x = \text{Round}_{\triangle}(\hat{x}) / \triangle$

$$\begin{aligned}
\hat{x} &= (Dc)_1 - s(DA)_1 \\
&= \overbrace{\cancel{s \cdot (\boldsymbol{DA})^{\boldsymbol{T}}_{\mathbf{1}}} + d_{11}e_1 + d_{12}e_2 + \left\lfloor \frac{q}{p} \right\rfloor (d_{11}(\mu_1 = 0) + d_{12}(\mu_2 = 1))}^{(Dc)_1} - \cancel{s(DA)^T_1} \\
&= d_{11}e_1 + d_{12}e_2 + \left\lfloor \frac{q}{p} \right\rfloor d_{12} \\
x &= \text{Round}_{\triangle}\left(d_{11}e_1 + d_{12}e_2 + \left\lfloor \frac{q}{p} \right\rfloor d_{12} \right) / \left\lfloor \frac{p}{q} \right\rfloor \\
&= \left(\left\lfloor \frac{q}{p} \right\rfloor (d_{11}\mu_1 + d_{12}\mu_2) \right) / \left\lfloor \frac{q}{p} \right\rfloor \\
&= \left(\left\lfloor \frac{q}{p} \right\rfloor (d_{11}0 + d_{12}1) \right) / \left\lfloor \frac{q}{p} \right\rfloor) \\
&= \left(\left\lfloor \frac{q}{p} \right\rfloor d_{12} \right) / \left\lfloor \frac{q}{p} \right\rfloor) \\
&= d_{12}
\end{aligned}$$

3.2 Fast Linearly Homomorphic Encryption







3.3 Simple PIR

We(the paper [1]) make three crucial observations about Regev encryption:

1. A large part of the ciphertext- the matrix A, is independent of the encrypted message can be generated in an offline pre-processing phase.
2. Regev encryption remains secure even when the same matrix A is used to encrypt polynomially many messages, provided that each ciphertext uses an independent secret vector s and error vector e .
3. It is possible to take A as pseudorandom, rather than truly random, with only a negligible loss in security. Allowing us to represent A by a short random seed succinctly.

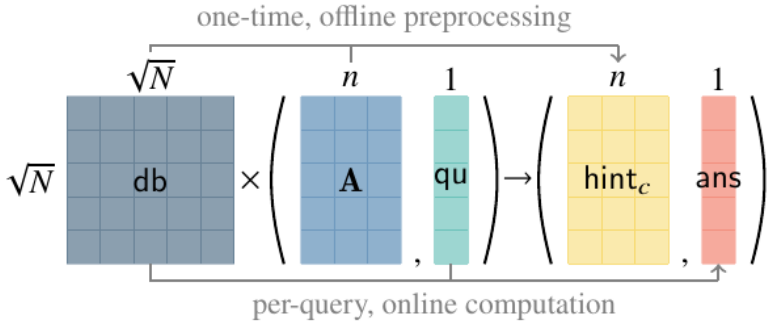
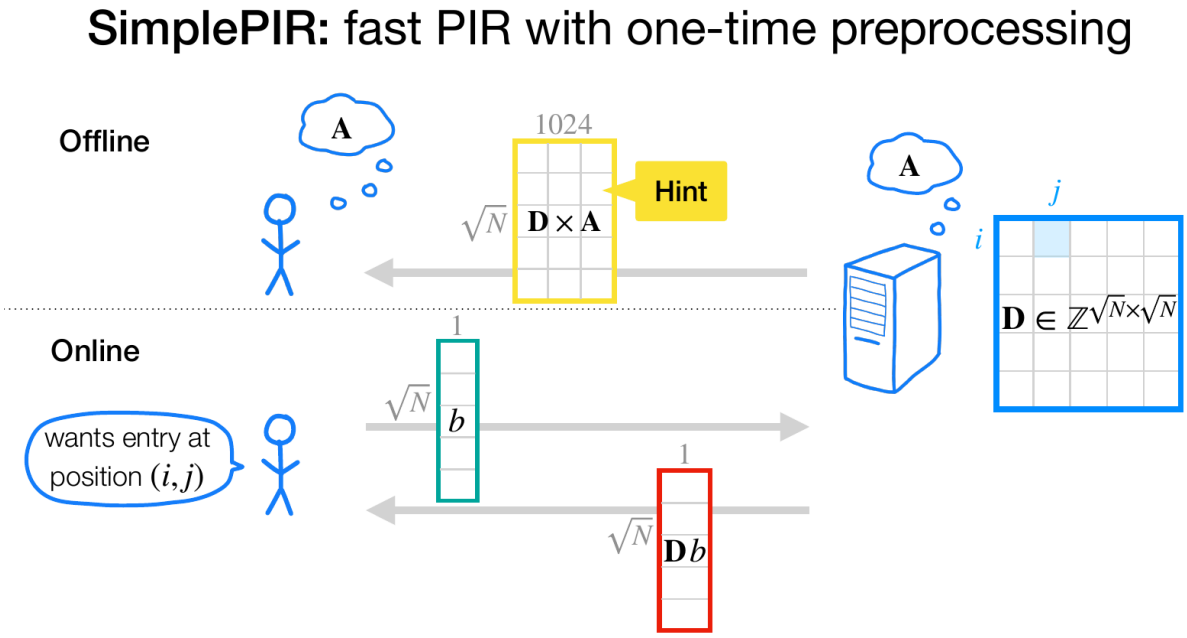


Figure 10: The server computation in SimplePIR. Each cell represents a $\mathbb{Z}_q$ element, and $\times$ denotes matrix multiplication. The server performs the bulk of its work in a one-time preprocessing step. Thereafter, the server can answer each client's query with a lightweight online phase.

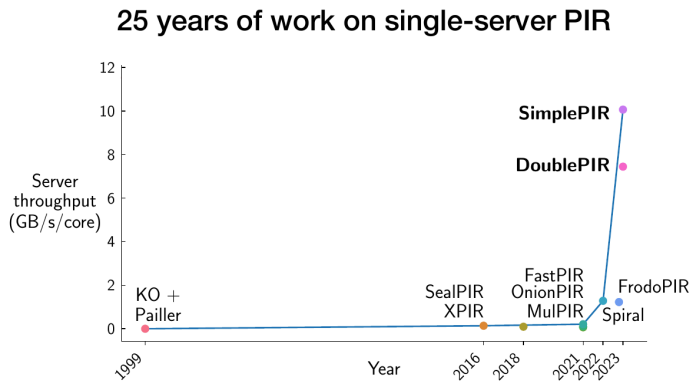
Now, we make the following modifications:

1. Preprocessing Phase:
   - Server computes $D \cdot A$ ahead of time, requiring 2nN operations in $\mathbb{Z}_q$.
   - To answer a query, only $2N$ operations in $\mathbb{Z}_q$ are needed for computing $D \cdot c$, instead of $2N + 2nN$
2. Shared Matrix A:
   - All clients use the same matrix A for their queries.
   - Server precomputes $D \cdot A$ once and shares it with all clients, amortizing the cost.
3. Compression via Pseudorandomness:
   - Compress A using a public hash function in counter mode.
   - Saves bandwidth and storage by communicating and storing only a small seed for A.



The security of the SimplePIR (appendix A) construction follows almost immediately from the security of Regev encryption with a reused matrix A, which in turn follows from the hardness

of LWE. SimplePIR's correctness follows from the correctness of Regev's linearly homomorphic encryption scheme and of Kushilevitz and Ostrovsky's "square-root" PIR template.



4 Appendix A

Construction: SimplePIR. The parameters of the construction are a database size N, LWE parameters (n, q, χ), a plaintext modulus $p \ll q$, and a LWE matrix $\mathbf{A} \in \mathbb{Z}_q^{\sqrt{N} \times n}$ (sampled in practice using a hash function). The database consists of N values in $\mathbb{Z}_p$, which we represent as a matrix in $\mathbb{Z}_p^{\sqrt{N} \times \sqrt{N}}$. Define the scalar $\Delta := \lfloor q/p \rfloor \in \mathbb{Z}$.

$\mathsf{Setup}(\mathsf{db} \in \mathbb{Z}_p^{\sqrt{N} \times \sqrt{N}}) \to (\mathsf{hint}_s, \mathsf{hint}_c)$.

- Return $(\mathsf{hint}_s, \mathsf{hint}_c) \leftarrow (\perp, \mathsf{db} \cdot \mathbf{A} \in \mathbb{Z}_q^{\sqrt{N} \times n})$.

$\mathsf{Query}(i \in [N]) \to (\mathsf{st}, \mathsf{qu})$.

- Write i as a pair $(i_{\text{row}}, i_{\text{col}}) \in [\sqrt{N}]^2$.
- Sample $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$ and $\mathbf{e} \xleftarrow{R} \chi^{\sqrt{N}}$.
- Compute $\mathsf{qu} \leftarrow (\mathbf{A}\mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{u}_{i_{\text{col}}}) \in \mathbb{Z}_q^{\sqrt{N}}$, where $\mathbf{u}_{i_{\text{col}}}$ is the vector of all zeros with a single '1' at index i_{col}.
- Return $(\mathsf{st}, \mathsf{qu}) \leftarrow ((i_{\text{row}}, \mathbf{s}), \mathsf{qu})$.

$\mathsf{Answer}(\mathsf{db} \in \mathbb{Z}_p^{\sqrt{N} \times \sqrt{N}}, \mathsf{hint}_s, \mathsf{qu} \in \mathbb{Z}_q^{\sqrt{N}}) \to \mathsf{ans}$.

- Return $\mathsf{ans} \leftarrow \mathsf{db} \cdot \mathsf{qu} \in \mathbb{Z}_q^{\sqrt{N}}$.

$\mathsf{Recover}(\mathsf{st}, \mathsf{hint}_c \in \mathbb{Z}_q^{\sqrt{N} \times n}, \mathsf{ans} \in \mathbb{Z}_q^{\sqrt{N}}) \to d$.

- Parse $(i_{\text{row}} \in [\sqrt{N}], \mathbf{s} \in \mathbb{Z}_q^n) \leftarrow \mathsf{st}$.
- Compute $\hat{d} \leftarrow (\mathsf{ans}[i_{\text{row}}] - \mathsf{hint}_c[i_{\text{row}}, :] \cdot \mathbf{s}) \in \mathbb{Z}_q$, where $\mathsf{ans}[i_{\text{row}}]$ denotes component i_{row} of ans and $\mathsf{hint}_c[i_{\text{row}}, :]$ denotes row i_{row} of hint_c.
- Return $d \leftarrow \mathsf{Round}_\Delta(\hat{d})/\Delta \in \mathbb{Z}_p$, which is $\hat{d}$ rounded to the nearest multiple of Δ and then divided by Δ.

Bibliography

[1] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan, "One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval". [Online]. Available: https://eprint.iacr.org/2022/949

[2] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan, "Private Information Retrieval", *J. ACM*, vol. 45, no. 6, p. 965, Nov. 1998, doi: 10.1145/293347.293350.

[3] E. Kushilevitz and R. Ostrovsky, "Replication is not needed: single database, computationally-private information retrieval", in *Proceedings 38th Annual Symposium on Foundations of Computer Science*, 1997, pp. 364–373. doi: 10.1109/SFCS.1997.646125.

[4] J. Reardon, J. Pound, I. Goldberg, and D. R. Cheriton, "Relational-Complete Private Information Retrieval", 2007. [Online]. Available: https://api.semanticscholar.org/CorpusID:394690

[5] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia, "Splinter: Practical Private Queries on Public Data", in *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, Boston, MA: USENIX Association, Mar. 2017, pp. 299–313. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/wang-frank

[6] S. Angel, H. Chen, K. Laine, and S. Setty, "PIR with compressed queries and amortized query processing". [Online]. Available: https://eprint.iacr.org/2017/1142

[7] S. Angel and S. Setty, "Unobservable Communication over Fully Untrusted Infrastructure", in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA: USENIX Association, Nov. 2016, pp. 551–569. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/angel

[8] T. Gupta, N. Crooks, W. Mulhern, S. Setty, L. Alvisi, and M. Walfish, "Scalable and Private Media Consumption with Popcorn", in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, Santa Clara, CA: USENIX Association, Mar. 2016, pp. 91–107. [Online]. Available: https://www.usenix.org/conference/nsdi16/technical-sessions/presentation/gupta-trinabh

[9] L. Li, B. Pal, J. Ali, N. Sullivan, R. Chatterjee, and T. Ristenpart, "Protocols for Checking Compromised Credentials", in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, in CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, p. 1387. doi: 10.1145/3319535.3354229.

[10] M. Backes, A. Kate, M. Maffei, and K. Pecina, "ObliviAd: Provably Secure and Practical Online Behavioral Advertising", in *2012 IEEE Symposium on Security and Privacy*, 2012, pp. 257–271. doi: 10.1109/SP.2012.25.

[11] D. Kogan and H. Corrigan-Gibbs, "Private Blocklist Lookups with Checklist", in *30th USENIX Security Symposium (USENIX Security 21)*, USENIX Association, Aug. 2021, pp. 875–892. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/kogan

[12] [Online]. Available: https://www.youtube.com/watch?v=JBVP3_PmbsI&t=160s

[13] O. Regev, "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography", *J. ACM*, vol. 56, no. 6, Sep. 2009, doi: 10.1145/1568318.1568324.

[14] [Online]. Available: https://blintzbase.com/posts/pir-and-fhe-from-scratch/

[15] C. Peikert, vol. 0, no. 2016, p.