Exploring Private Information Retrieval

Henry Barthelemy

Written: Spring 2024

Contents

1 Introduction

This paper will be an overview of various works in the Private Information Retrieval (PIR) field. We aim to explore the main findings of each paper, lower-level primitives involved, as well as compare the differences between them.

1.1 Private Information Retrieval Problem

In private information retrieval (PIR) a client wishes to request data from an unencrypted server while hiding the index which is being queried from the server. The trivial $O(n)$ solution (where n represents the database size) in both communication and client-computation, would be to have the server send the entire database to the client and for the client to find the index they need locally. Through various forms of preprocessing of the database or generating hints the retrieval can be done more efficiently and with less communication.

2 Private Information Retrieval with Sublinear Online Time

Corrigan-Gibbs and Kogan's work achieves sublinear online time for PIR through the use of a second server for providing hints [1]. The scheme utilizes the primative of shifting puncturable pseudorandom sets, which is a shifting property built on-top of puncturable pseudorandom sets in order to minimize the client storage and communication cost of hints during the setup. Their work then uses fully homomorphic encryption to convert the two servers approach into a single-server one.

2.1 Primitive: Puncturable pseudorandom sets

Syntax

Let $\mathcal{K}$ be a key-space and $\mathcal{K}_p$ be a punctured-key space. Puncturable pseudorandom consists of the following three operations:

- $\text{Gen}(1^\lambda, n) \to sk \in \mathcal{K}$
- $\text{Punc}(sk, i) \to sk_p \in \mathcal{K}_p$
- $\text{Eval}(sk) \to S \subseteq [n]$

Where sk is a secret key. Note that $\text{Gen}(1^\lambda, n)$ is a randomized algorithm while $\text{Punc}(sk, i)$ and $\text{Eval}(sk)$ are deterministic algorithms.

Properties

Correctness

Let $s(n) : \mathbb{N} \to \mathbb{N}$ where $s(n) \leq n$ (we can think of the output of $s(n)$ to be the size of the punctured subset) and z denote the randomness of $\text{Gen}(1^\lambda, n) \to sk$. With probability $\frac{1}{z}$,

1. $\binom{S \in [n]}{s(n)}$
2. if $\text{Punc}(sk, i) \to sk_p$, then $\text{Eval}(sk_p) \to S/\{i\}$

The correctness of a puncturable pseudorandom set states that when we puncture a set with an index, that punctured set no longer contains the index it was punctuate with.

Security

We will analyse the security by calculating the probability that an adversary $\mathcal{A}$ can guess the index i of a punctured pseudorandom subset ψ with size $s(n) : \mathbb{N} \to \mathbb{N}$.

The guessing game will be played like such:
A challenger will use Gen(1^λ, n) to generate a secret key sk then Eval(sk) to generate a set S. The challenger then randomly select an x^* from S. With this x^* index, the challenger punctures the secret key using Punc(sk, x^*) to generate $sk_p \in \mathcal{K}_p$. The challenger sends both 1^λ and sk_p to the adversary $\mathcal{A}$. The adversary will win the game if they guess the integer $x \in [n]$ where $x = x^*$, we will denote this event $W_{\lambda,n}$.

We can describe the probability that $\mathcal{A}$ by guessing the index of ψ, which is the probability that the adversary wins the guessing game, as:

$$\text{PSAdv}[\mathcal{A}, \psi](\lambda, n) := \Pr[W_{\lambda,n}] - \frac{1}{n - s(n) + 1}$$

Since the adversary has $s(n) + 1$ elements in the resulting set, and thus has $n - s(n) + 1$ elements all with equal chance of being x^*.

2.2 Primitive: Shifting puncturable pseudorandom sets

Syntax

A shifting puncturable pseudorandom set is a built by encapsulating a puncturable pseudorandom set and adding GenWith and Shift capabilities. Let $\psi = (\text{Gen}, \text{Punc}, \text{Eval})$ be a pseudorandom punctured set, with associated key space $\mathcal{K}$ and punctured key space $\mathcal{K}_p$. Shifting puncturable pseudorandom sets will be defined with the operations

1. Gen$'(1^\lambda, n)$ samples a random subset of $[n]$, Δ, and outputs $sk \leftarrow (\psi.\text{Gen}.(1^\lambda, n), \Delta)$
2. Punc$'((sk, i), \delta)$ outputs $sk_p \leftarrow (\text{Punc}(sk, i - \Delta), \Delta)$
3. Eval$'((sk, \Delta)) \to S \subseteq [n]$ where $S \leftarrow \{i + \Delta : i \in Eval(sk)\}$
4. Shift$((sk, \Delta), j) \to sk'$ where $sk' \leftarrow (sk, \Delta + j \bmod n)$
5. GenWith$(1^\lambda, n, i)$ outputs Shift$(sk, i - i')$ where $sk \leftarrow \text{Gen}'(1^\lambda, n)$ and i' is randomly choosen from Eval(sk).

Note that the shifting puncturable pseudorandom set $\psi' = (\text{Gen}', \text{Punc}', \text{Eval}')$ will have the associated key space of $\mathcal{K} \times \mathbb{N}$ and punctured key space of $\mathcal{K}_p \times \mathbb{N}$.

Properties

Correctness

Not only does this follow from the puncturable pseudorandom set but we can additionally add that

1. For every $\lambda, n \in \mathbb{N}$, then the two are identitical

$$\left\{\begin{array}{l} i \leftarrow [n] \text{ randomly} \\ \text{sk} \leftarrow \text{GenWith}(1^\lambda, n, i) \\ \text{output}(i, \text{sk}) \end{array}\right\} \equiv \left\{\begin{array}{l} \text{sk} \leftarrow \text{Gen}'(1^\lambda, n) \\ S \leftarrow \text{Eval}'(sk) \\ i \leftarrow S \text{ randomly} \\ \text{output}(i, \text{sk}) \end{array}\right\}$$

Security

Follows from the security of the puncturable pseudorandom set. We let ψ be a puncturable pseudorandom set and ψ' be a shifting puncturable pseudorandom set. Let $\mathcal{A}$ be the adversary for ψ' and $\mathcal{B}$ be the adversary for ψ. Given the set up as before, set $sk'_p \leftarrow (sk_p, \Delta)$ where $\Delta \leftarrow [n]$ randomly. Then when running $\mathcal{A}$ on sk'_p, we get $j \in [n]$, and then have $\mathcal{B}$ output $i \leftarrow j - \Delta$. $\mathcal{B}$ wins with same probability as $\mathcal{A}$ thus the extension is secure.

2.3 Single query offline/online PIR Scheme

Syntax

Offline/online PIR is made up of 5 algorithms: Setup, Hint, Query, Answer, and Reconstruct

- $\text{Setup}(1^\lambda, n) \to (ck, q_h)$, takes the security parameter and database length to output a client key ck and a hint request q_h.
- $\text{Hint}(x, q_h) \to h$, takes in a database (a string of 0 and 1s), a hint request, and outputs a hint.
- $\text{Query}(ck, i) \to q$, takes in a client key ck and an index i, and outputs a query q
- $\text{Answer}^x(q) \to a$, takes in a query q and gets access to an oracle that takes an index j and outputs the j-th bit of a database x_j.
- $\text{Reconstruct}(h, a) \to x_i$ that takes a hint h and answer a to output a bit x_i.

Note that Setup and Query are randomized algorithms while Hint, Answer, and Reconstruct are deterministic algorithms.

Properties

Correctness

We need that for any $\lambda, n \in \mathbb{N}$, $x \in 0, 1$ (the database), and $i \in [n]$, the index, that:

$$\Pr\left[\text{Reconstruct}(h, a) = x_i\right] = 1$$

For h and a derived with where $(ck, q_h) \leftarrow \text{Setup}(1^\lambda, n)$, $h \leftarrow \text{Hint}(x, q_h)$, $q \leftarrow \text{Query}(ck, i)$, and $a \leftarrow \text{Answer}^x(q)$. Informally, when we query an index with a hint, the answer given (with the correct hint) should reconstruct to the bit at that index.

Security

The security of offline/online PIR is defined as the following. Let $\lambda, n \in \mathbb{N}$ where λ is the security parameter and n is the size of the database. Let $i, j \in [n]$ be indices for the distributions. We define a distribution $D_{\lambda,n,i}$ as:

$$D_{\lambda,n,i} := \left\{ q : \begin{matrix} (ck, q_h) \leftarrow \text{Setup}(1^\lambda, n), \\ q \leftarrow \text{Query}(k, i) \end{matrix} \right\}$$

We can describe the advantage for an adversary $\mathcal{A}$ as

$$\text{PIRadv}[\mathcal{A}, \Pi] := \max_{i,j \in [n]} \{\Pr[\mathcal{A}(1^\lambda, D_{\lambda,n,i}) = 1] - \Pr[\mathcal{A}(1^\lambda, D_{\lambda,n,j}) = 1]\}$$

For the security proof we analysis the differences in probability of winning given a difference in distribution of a single query.

2.4 Multi-querry offline/online PIR Scheme

Syntax

Offline/online PIR is made up of 5 algorithms: Setup, Hint, Query, Answer, and Reconstruct

- $\text{Setup}(1^\lambda, n) \to (ck, q_h)$, takes the security parameter and database length to output a client key ck and a hint request q_h.

- Hint$(x, q_h) \to h$, takes in a database (a string of 0 and 1s), a hint request, and outputs a hint.
- Query$(ck, i) \to (ck', q_{left}, q_{right})$, takes in a client key ck and an index i, and outputs an updated client key ck', query for left server q_{left}, and right server q_{right}.
- Answer$^x(q) \to a$, takes in a query q and gets access to an oracle that takes an index j and outputs the j-th bit of a database x_j.
- Reconstruct$(h, a_{left}, a_{right}) \to (h', x_i)$ that takes a hint h and answers from left and right server a_{left} and a_{right} to output a new updated hint h' and the bit x_i from index i of the database.

Properties

Correctness

For all requests $i_1, ..., i_T \in [n]^T$, the following holds true

$$\Pr\left[\text{Reconstruct}(h, a_{left}, a_{right}) = x_{it}\right] = 1$$

for $t = 1, ..., T$, where $(ck, q_{left}, q_{right}) \leftarrow Querry(ck, i_t)$, $a_{left} \leftarrow Answer^x(q_{left})$, and $a_{right} \leftarrow Answer^x(q_{right})$.

Security

We create two games, and we are secure if the advantage the adversary has is negligible. The one game servers for the (offline server / hint server) and the second game is for the right server (online server / query server). Note that this setup assumes no collusion (shared information) between the two servers. We define the advantage of the adversary $\mathcal{A}$ as

$$\text{PIRadv}[\mathcal{A}, \Pi](\lambda, n, T) := \max_{\sigma \in \{\text{right,left}\}} |\Pr[W^{\sigma}_{\lambda,n,T,0}] - \Pr[W^{\sigma}_{\lambda,n,T,1}]|$$

where $W^{\text{left}}_{\lambda,n,T,b}$ is the event that the adversary outputs $b' = 1$ on the left-server game, and $W^{\text{right}}_{\lambda,n,T,b}$ for the win on the right-server game.
We define the left server game as:

- (ck, q_h) $\leftarrow$ Setup(1^λ, n)
- st $\to \mathcal{A}(1^\lambda, q_h)$
- For $t = 1, ..., T$:
  (st, i_0, i_1) $\leftarrow \mathcal{A}$(st)
  (ck, q_{left}, q_{right}) $\leftarrow$ Query(ck, i_b)
  st $\leftarrow \mathcal{A}$(st, q_{left})
- $b' \leftarrow \mathcal{A}$(st)

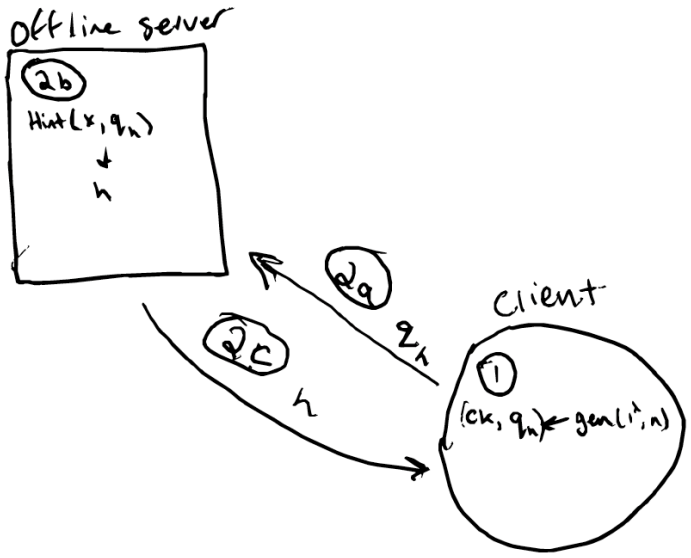
and the right server game as:

- (ck, q_h) $\leftarrow$ Setup(1^λ, n)
- st $\to \mathcal{A}(1^\lambda)$
- For $t = 1, ..., T$:
  (st, i_0, i_1) $\leftarrow \mathcal{A}$(st)
  (ck, q_{left}, q_{right}) $\leftarrow$ Query(ck, i_b)
  st $\leftarrow \mathcal{A}$(st, q_{right})
- $b' \leftarrow \mathcal{A}$(st)

2.5 Construction: Two-server PIR with sublinear online time

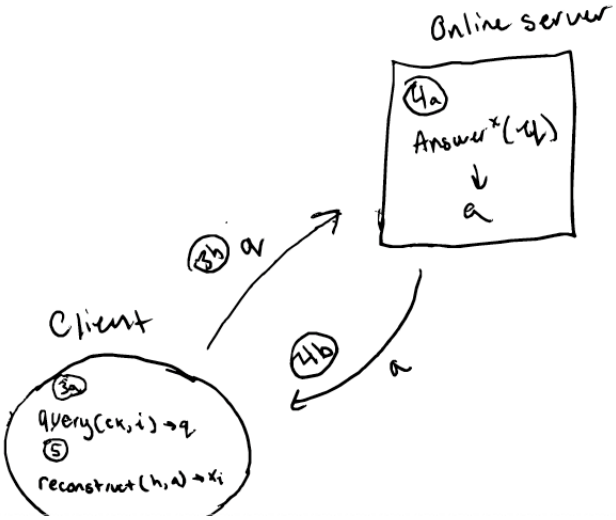
The scheme is defined as π= (Setup, Hint, Query, Answer, and Reconstruct). The construction for two-server PIR would be as follows.
In the **offline phase**, the following are run

- **Setup**(1^λ, n) which will initialize the client key and query. The client will run this to acquire ck and q_h. q_h will be the list of shifting puncturable pseudorandom set, randomly selected from $[n]$.
- To generate the hints, first the client will pass q_h to the hint server. The hint server will then run **Hint**(q_h, x) which takes in the database and q_h generated from the client, and uses this to generate a list of hints, which are parities of each of the subsets. In other words, given m shifting puncturable pseudorandom set, $(S_1, ..., S_m)$, hint will evaluate the parity of the given set keys, and return this as $(h_1, ..., h_m)$ where h_i corresponds to the parity of bits at indexes in S_i.



In the **online phase**, given the client would like bit at index i_{pir}, which we denote $x_{i_{pir}}$, the following happens:

- The client needs to generate a query to send to the online server. In order to do this, it takes i_{pir} and it's client key, this runs the probabilistic function **Query**(c_k, i_{pir}). With probability $1 - \frac{s-1}{n}$, where s is the size of the subsets, it will puncture the puncturable set key containing i, sq_q, by i, the resulting set key if evaluated now does not contain i. Moreover, with probability $\frac{s-1}{n}$, we will randomly select an element of the resulting set where $i' \neq i$. We can then output the resulting punctured set key, $q = Punc(sk_q, i_{punc})$ where $i_punc = i_{pir}$ or i' depending on where the probability took us.
- The client then sends this q to the online server. The online server will then run answer on punctured set key. During **Answer**(q), the server will evaluate this punctured set key and calculate the parity of the bits at the indexes of the resulting set. This parity, which we will denote a, is then sent back to the client.
- The client now uses a, the parity of the punctured subset and h_j, the parity of the non-punctured subset to calculate the bit $x_{i_{pir}}$. The client can now calculate using both h and a with **Reconstruct**(h, a), which calculates the difference in parity between h_j and a (mod 2) to output the bit $x_{i_{pir}}$.



This construction runs into the issue where with probability $\frac{s-1}{n}$, we will be puncturing the subset with $i_{punc} \neq i_{pir}$. Therefore the information given to us by the online server will not be useful in calculating $x_{i_{pir}}$. By setting the number of subsets, $m = nlog(n/s)$, we can drive this probability to $1/n$. By running the scheme $O(\lambda)$ times we then drive this probability of failure to negl(λ), bringing us computational security. This construction has a big downside though: we cannot achieve this security if we need to request a bit i' in the same S_j, where $i_{pir} \in S_j$, since the online server can then, with high probability, figure out that we are computing the parity of $S_j \setminus \{i_{pir}\}$ the first time and $S_j \setminus \{i'\}$ the second, then use this to realize the client wants $\{i_{pir}, i'\}$. This brings us to a construction adding a "refresh" feature

2.6 Arbitrary Queries Two-server PIR with sublinear amortized time

The scheme is a modified version of the above one. We set $m = \sqrt{n}logn$ and each subset is roughly size $\sqrt{n}$. Once the client uses S_j to get an index i where $i \in S_j$, we "refresh" the hints. The online server interactions with the clients remain the same as in the previous construction.

The offline server will have a refresh feature added to it, where it chooses a new set, S_{new}, to replace S_j such that it preserves the distribution of the sets. The problem becomes how to sample this random elements such that the distribution holds. Choosing a fully random set with random elements means that the amount of sets containing i will decrease. Instead, the set can be a random set with the constraint that one element is i and the other elements are randomly selected. This conserves the randomness while insuring that the distribution of elements remains the same.

3 PIANO

PIANO [2] also utilizes a hint prepossessing set-up. The hints are split up into three different data-structures: a primary table, replacement entries, and backup table. The prepossessing happens in chunks, allowing for lower communication costs. The replacement entries allow for perfect correctness and backup table allows for arbitrary number of queries.

3.1 Primitive: Puncturable Pseudorandom Function (PRF)

The scheme uses a combination of these chunks and PRFs in order to minimize client storage. The scheme utilizes PRFs to achieve client side storage of $\widetilde{O}(\sqrt{n})$.

Security

A PRF F is secure if it is indistinguishable from a truly random function. A truly random function is randomly selected from all possible functions from $F_{n\to h} : \{0,1\}^n \to \{0,1\}^h$. For a PRF h, $h(x) = y$ does not tell you about $h(x') = y'$. More formally, a PRF, F is secure if for all polytime adversaries, A

$$Adv_{PRF}[A, F] = |Pr[EXP(0) = 1] - Pr[EXP(1) = 1]| \leq negl$$

where the experiment is between a challenger and adversary A, the adversary gets to ask as many responses $(f(x))$ for any input x. A bit b is randomly assigned to 0 or 1, and if $b = 0$, the challenger will respond with $f(x)$ where f is a chosen from a subset/family of $F_{n\to h}$, otherwise if $b = 1$, f is chosen randomly from $F_{n\to h}$. The challenger using the outputs will attempt to distinguish whether b is 0 or 1. If the adversary is correct with a more than $\frac{1}{2}+$ negligible, or its advantage is more than negligible, than the PRF is not secure.

3.2 Single-Server Scheme

Given a server of size n, this scheme splits up the indices into $\sqrt{n}$ chunk, where the jth chunk contains the indices $\{j\sqrt{n}, ..., (j+1)\cdot\sqrt{n} - 1\}$. The scheme splits the hints into three tables/data: the primary table, the replacement entries, and backup tables.

Additionally, the client downloads in this chunk method to minimize the amount of local storage needed when calculating parity, this is accomplished since the primary table, replacement entry, and backup table only depend on the chunk.

Syntax

The syntax is split up into 3 methods:

1. setup(λ, κ)
2. query(i)
3. refresh

Setup creates the primary table, replacement entry, and backup table. Query computes the bit at the index provided, and refresh updates the primary table using the replacement entries.

Security Proof

We will be using a simulation proof, which works its way from an ideal simulation (random distribution) to the real simulation (what the scheme does) and proves that the view from the adversary remains the same through a series views involving small mutations from one another (called hybrids). We will start with the ideal simulation where D_n is the distribution of the random sets. Then we have Hyb_1, Hyb_2 and Real*.

Ideal:

- *Offline:* Nothing is done
- *Online:* For a query i, the client sends a random set from D_n, to $\mathcal{A}$

Hyb_1

- *Offline:* Client samples random sets $S_1, ..., S_{M_1} \leftarrow D_n^{[M_1]}$
- *Online:* For each online round (describing the number of queries) t, $\mathcal{A}$ chooses a query x_t. The client samples $S \leftarrow D_n$ such that $x_t \in S$ and a random index r from the chunk of x_t. The client then sends $(S/\{x_t\}) \cup \{r\}$

Hyb_2

- *Offline:* Client samples random sets $S_1, ..., S_{M_1} \leftarrow D_n^{[M_1]}$
- *Online:* For each online round (describing the number of queries) t, $\mathcal{A}$ chooses a query x_t.
    1. Client selects S_j such that $x_t \in S_j$ and j is smallest index in $[M_1]$ and denotes this set S^*. If no such S_j exists client samples a set randomly using D_n such that $x_t \in S^*$.
    2. Client samples r from chunk of x_t. Client sends $(S^*/\{x_t\}) \cup \{r\}$
    3. Client then sets up the refreshing by sampling a set S' randomly from D_n such that $x_t \in S'$, if the client used a set S_j for S^*, replace S_j by S'

Real*

- *Offline:* Client samples random sets $S_1, ..., S_{M_1} \leftarrow D_n^{[M_1]}$ and $\overline{S}_{i,j} \rightarrow D_n$, for $i \in \{0, 1, ..., \sqrt{n} - 1\}$, $j \in [M_2]$
- *Online:* For each online round (describing the number of queries) t, $\mathcal{A}$ chooses a query x_t.
    1. Client selects S_j such that $x_t \in S_j$ and j is smallest index in $[M_1]$ and denotes this set S^*. If no such S_j exists client samples a set randomly using D_n such that $x_t \in S^*$.
    2. Client samples r from chunk of x_t.
    3. Client sends $(S^*/\{x_t\}) \cup \{r\}$
    4. Client then sets up the refreshing by consuming an unconsumed set of $\overline{S}_{i^*,1}, \overline{S}_{i^*,2}, ..., \overline{S}_{i^*,M_2}$, and replaces the $j-th$ set in the local sets with $S' = (\overline{S}_{i^*,j}/\overline{S}_{i^*,j[i^*]}) \cup \{x^t\}$ for the first unconsumed set $\overline{S}_{i^*,j}$. If no such set exists, then refresh sampling a set S' randomly from D_n such that $x_t \in S'$. If the client used a set S_j for S^*, replace S_j by S'

From the ideal simulation to the first hybrid one, the view from the adversary is the same, since the sampling of sets does not change between the chunking or a random set S_j, since there are the same amount of elements in S_j each with same probability based on D_n distribution by the replacement of i with and random index r (since it is possible that $i = r$).

From Hyb$_\mathbf{1}$ to Hyb$_\mathbf{2}$ we tackle choosing the set based on the chunk rather than sampling a random set and replacing an index. We also tackle refreshing the sets but without use of the backup table (and replacing this with randomness)

From Hyb$_\mathbf{2}$ to Real* we add in the refresh through the backup table rather than through randomness.

The only difference between Real* and the Real simulation is that the Real simulation uses PRF's rather than random sets. Proving these are equivalent is trivial by the definition of pseudo randomness and will be left out. Additionally, this scheme is secure under collusion of the servers too, and does not need to be run in parallel like the PIR scheme to achieve negligible security advantage.

4 PANDAS

PANDAS [3] expands upon PIR by creating a multi-client PIR scheme that allows bounded-collusion between the clients and provides anonymity. Additionally, this paper introduces and utilizes concepts from oblivious RAM (ORAM) and doubly-efficient PIR (DEPIR).

4.1 Oblivious RAM (ORAM)

The oblivious RAM problem is about hiding access patterns. It is used when there is untrusted ram running a series of instructions. An ORAM scheme has the syntax of:

- Setup(1^λ, DB) $\rightarrow$ ck, State

- Access(op, addr, val) which outputs a value if the op is read, otherwise if op is write then it writes at the specified addr with the value inputted

An ORAM scheme is considered secure when

$$Pr[b' = b] = \frac{1}{2} + negl(\lambda)$$

For the game below between a challenger and PPT adversary:

- Adversary sends the challenger a datebase, DB
- Challenger runs the setup on the DB to obtain a server state State, client state ck, and sends the server state State to the adversary, and picks a random bit $b \leftarrow \{0, 1\}$.
- For some poly in the security parameter number of times:
  - Adversary sends two instructions (op, addr_1, val_1) and (op, addr_2, val_2) to the challenger. Note that for each instruction in the series, the operation stays consistent between the two.
  - Challenger executes the operations at the specified address (and writes the value if its a write operation) and sends the access pattern to the adversary
- Adversary outputs bit b'

4.2 Doubly Efficient PIR (DEPIR)

DEPIR is sometimes referred to as PIR with preprocessing. With the preprocessing, the scheme is able to achieve sub-linear computational work on the server. The doubly part of the efficiency comes from the sub-linear computation efficiency on both the server and client side.

4.3 Primitive: Locally Decodable Codes (LDC)

We formally define Locally Decodable Codes as

Enc: $\Sigma^N \rightarrow \Sigma^M$ deterministically maps message input of size N (referred to as the message length) into code word of size M (referred to as the block length). Where $M > N$.

Query: Is a probabilistic algorithm which on input $i \in [N]$ will generate k indices, $j_1, ..., j_k \in [M]$ and a decoding state st

Dec: Is a probabilistic algorithm which on input $C_{j_1}, ..., C_{j_k} \in [M]$ and st, outputs m_i

These locally decodable codes can be utilized with Reed-Muller codes and used to build a bounded PIR construction. Due to the linearity of underlining LDC's, this is only secure for a bounded B number of queries.

4.4 Possible (wrong) Attempts

There are four previous attempts before the scheme for PANDAS was reached.

1. All clients share 1 key, this gave an efficient run time, but there were no collusion protections at all. If one client colluded the server would be able to figure out which information was requested. This did however provide anonymity. By sharing 1 key between clients, there is no distinguishable information for a single client.
2. The clients each have separate keys. The information gets accessed privately with the clients keys. This did reach the efficiency wanted with privacy, but provided no anonymity. The server could tell which client accessed the encrypted information since the key it used was attached to that client. Each client had a distinct key and therefore servers could tell clients apart from their queries.

3. All clients have their own key and we introduce dummy access to hide which client is accessing. This does lead to both privacy and anonymity; however, the approach has an inefficient linear runtime.

4. Client keys over several servers. This provides us with good runtime, privacy, and anonymity, but does not hold up to the single-server set up.

There was another closer attempt which was also unsuccessful. It utilized concepts from ORAM, obscuring access and the LDCs. The issue that arose from this attempt was the need for bogus calls in order to maintain anonymity which again resulted in too high of a runtime.sla

4.5 PANDAS Scheme

Bounded Access PANDAS combines LDC + PRP, which has easy bogus reads for anonymity. This allows for anonimity while maintaining a lower runtime. The downside is the need to bound the numbers of reads; Since PRP is limited we need to refresh after a certain number of calls.

Unbounded access is possible since PANDA server is stateful. PRF needed to generate a new PRP, and FHE is needed for a refresh with this PRP after "B" reads (since then we can make sure the server does not learn PRP/PRF keys). This is effectively unbinding the number of queries by refreshing the security after the bound is reached. This also needs a leveled FHE assumption which comes from an learning with error (LWE) assumption to correctly "refresh" the server without prior information on the previous PRF.

4.6 Read only Security Game

Security game overview:

1. Adversary generates two sequences of reads, R^0, R^1 (where each read specifies client, j, and an address, addr).

2. We also make sure that if a client is corrupt that the read at that index is the same in both sequences (preventing use of corrupt client to figure out which sequence is chosen).

3. The challenger chooses randomly, b, and selects R^b, one of the two read sequences. It sends the adversary view of the servers and corrupted clients while executing the read operations.

4. The adversary outputs b' and wins if it correctly guesses which sequence of reads the challenger executed.

The winning advantage of the adversary is described as

$$\text{Adv}_{\mathcal{A}}(\lambda) = |Pr[b = b'] - \frac{1}{2}|$$

4.7 Public Encoding PANDAS

The idea for this is that we simply encodes a new database $\tilde{DB}$. Allowing us to have arbitrary requests.
Syntax:

- KeyGen(1^λ, 1^n, 1^t, 1^L) $\rightarrow$ (pk, ck_j) generates the public key for encoding and secret client keys for reads.
- Encode(pk, DB, lab) $\rightarrow$ $(\tilde{DB}^1 ... \ \tilde{DB}^j)$ generates a newly encoded DB for every permutation of the data.
- Query(ck_j, addr, lab) $\rightarrow$ $(q_1, ..., q_k)$ generates a list of points at the address for the given labeled database
- Recover(ck_j, lab, $(\tilde{DB}_{q_1} ... \ \tilde{DB}_{q_k})$) $\rightarrow$ val. Recover takes a series of points at the database (gotten from query), and determines the corresponding value).

Note that KeyGen and Query are probabilistic algorithms while Encode and Recover are deterministic ones.

4.8 Public-writes PANDA

Public-writes PANDA considers the setting of a public database where any client can preform read/write. Although the values being written are public, the clients should remain anonymous. The scheme for public-writes PANDA has write operations, but only grantees privacy for read operations. A malicious client can just read the database after every operation, learning what was written. Using PANDA with public encoding would be too inefficient as it would require (at least) linear time to update the encoded database. The construction rather takes a similar form to that of hierarchal ORAM. That is to say the database is stored in a sequence of Log L levels, where $L = |DB|$. Each level l_i consists of seperate instance of a read-only PANDA with public encoding, and will contain at most $L_i = 2^{l_i}$ database values. That is to say each level contains 2 times more than the above it (if $l_i = 0$ is the top level).

4.9 Secret-writes PANDA

Secret-writes PANDA builds upon Public-writes PANDA and utilizes a prf for each client in order to ensure that writes remain private. When writing at some location j in a database, a client can use her unique PRF to write at $\text{PRF}(j, c)$ and when reading some location j she then uses her perf to read at the largest count where there exists a value at $\text{PRF}(j, c)$ and then writes at $\text{PRF}(j, c + 1)$. This approach does have a downside of the server storage growing with the total number of writes, rather than the data size as the database cannot delete old copies.

5 Towards DEPIR

This [4] work by Canetti, Holmgren, and Richelson explores developing a single-server PIR scheme where the client has no update-able state, and the per query work of the client and server is sublinear in database size.

This paper sets up three types of preprocessing: public preprocessing (randomness of preprocessing is public), public client (long-term key is public, but not preprocessing), and designated client (where the client's key is hidden from the server). In all three cases, the client has no updateable state. They then provide a DEPIR construction for the designated client case and show the preprocessing is doomed to fail for the public client and public preprocessing case.

5.1 Formal DEPIR Definition

Syntax

- $\text{Keygen}(1^\lambda) \to$ a key k
- $\text{Process}(k, DB \in \Sigma^N) \to \tilde{DB}$
- $\text{Query}(k, i \in [N]) \to$ query q, and local state st
- $\text{Resp}(q, \tilde{DB}) \to$ answer a
- $\text{Dec}(k, st, a) \to DB_i \in \Sigma$

Correctness

For all $DB \in \Sigma^N$ and $i \in [N]$

$$Pr[Dec(st, a) \neq DB_i] \leq \epsilon$$

Where $K \leftarrow \text{Keygen}(1^\lambda)$, $\text{Process}(k, DB \in \Sigma^N) \to \tilde{DB}$, $q, st \leftarrow \text{Query}(k, i)$, $a \leftarrow \text{Resp}^{\tilde{DB}}(q)$. If $\epsilon = 0$ then we have perfect correctness. The security depends on which case we are in and is defined below accordingly.

5.2 Designated Client

The results of this paper and constructions are for the designated client case where key is only known to the client. There are two variants: the server does not know the database, and the server might know database but not queried location.

Formal Designated Client Security Definition

A scheme π = (Keygen, Process, Query, Resp, Dec) is a Designated Client if for all PPT adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ where $\text{REAL}_{\mathcal{A},\pi} \approx \text{IDEAL}_{\mathcal{A},S}$. We define $\text{REAL}_{\mathcal{A},\pi}$ as:

(1) $\mathcal{A} \to DB$, $\tilde{DB} \leftarrow$Process(K, DB), $K \leftarrow$ Keygen(λ, N)

(2) Repeat until $\mathcal{A}$ generates index $i \in [N]$ where $q \leftarrow$ Query(K, i)

We define $\text{IDEAL}_{\mathcal{A},S}$ as

(1) $\mathcal{A}$ outputs a database DB and obtains an encoded database from the simulator $\tilde{DB} \leftarrow$ S(λ, DB).

(2) Repeat until $\mathcal{A}$ generates index $i \in [N]$ where $q \leftarrow$ S()

In other words, the view of the adversary does not change between a the simulation and with a private key.

Trivial Scheme

Trivial scheme is to have $\tilde{DB}$ to be $\tilde{DB}(i) = DB(\pi(i))$, where $K = \pi$ is a permutation of the data. This is similar to a one-time-pad but for PIR. We have one-round perfect correctness and security, that has $O(n)$ preprocessing and $O(1)$ server work for the query.

5.3 Public Client & Public Preprocessingt

Public Client

The key is public, but the randomness used to generate the key remains private. This allows multiple clients to query the server at the same time since it is a stateless setup. There are two variants again: one where k is known before the database, and one where it is database is based on k. There is also another variant where a private key is generated but only for use in databse processing step.

Public Preprocessing

In this case, the key and randomness to generate the key is public.

Formal Public Client & Public Preprocessing Security Definition

A scheme π = (Keygen, Process, Query, Resp, Dec) is a Designated Client if for all PPT adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ where $\text{REAL}_{\mathcal{A},\pi} \approx \text{IDEAL}_{\mathcal{A},S}$. We define $\text{REAL}_{\mathcal{A},\pi}$ as:

(1) $\mathcal{A}(K) \to DB$ where $K \leftarrow$ Keygen(λ, N) and $\mathcal{A}$ obtains $\tilde{DB} \leftarrow$Process(K, DB),

(2) Repeat until $\mathcal{A}$ generates final output: index $i \in [N]$ where $q \leftarrow$ Query(K, i)

We define $\text{IDEAL}_{\mathcal{A},S}$ as

(1) $\mathcal{A}(K) \to DB$ where $K \leftarrow$ S(λ, N) and $\mathcal{A}$ obtains $\tilde{DB} \leftarrow$S(DB),

(2) Repeat until $\mathcal{A}$ generates final output: $\mathcal{A}$ generates index $i \in [N]$ where $q \leftarrow$ S()

The above is a security definition for public preprocessing if Process uses no additional randomness other than K. The above a security definition for public client with secret preprocessing key if Keygen generates an additional key that is used by process (and otherwise remains unknown).

5.4 Primitive: LDC

This scheme similar to PANDAS utilizes LDCs and thus the protocol is only secure found a bounded B number of queries from the linearity of LDCs.

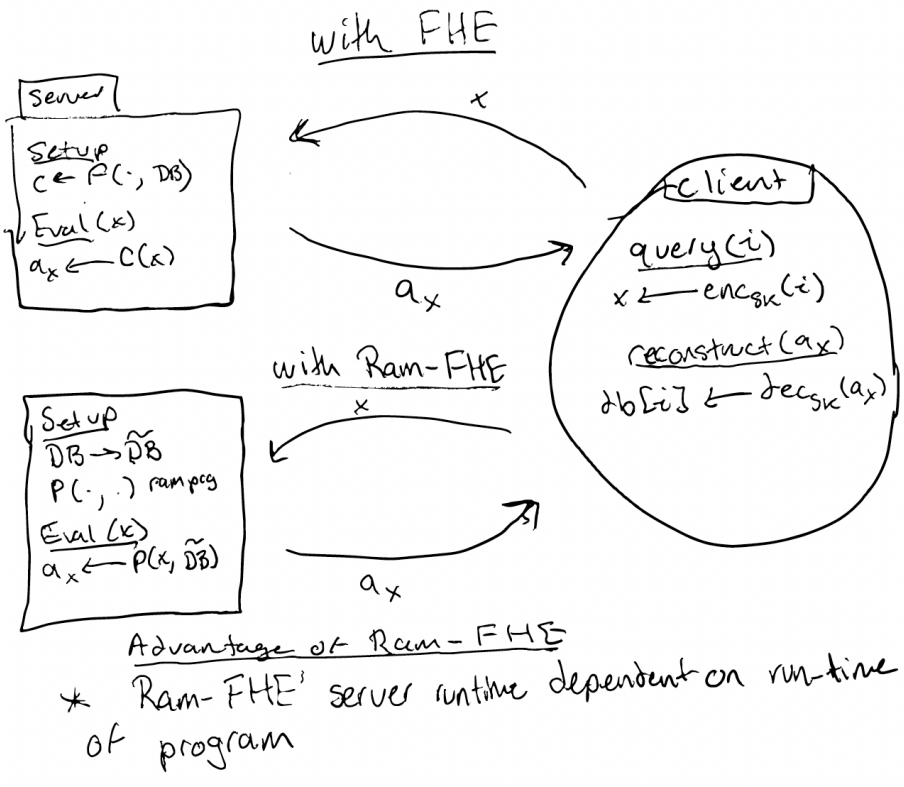
5.5 Assumption: HPN

This assumption hidden permutation with noise problem (HPN) helps bring the designated client case from bounded to unbounded number of queries. Addition to LDC but with added noise to the clients queries. This prevents the attacks that were possible on the original scheme. The downside is with the addition of noise, the security can no longer be perfect, and will be at maximum computational.

6 Doubly Efficient Private Information Retrieval and Fully Homomorphic RAM Computation from Ring LWE

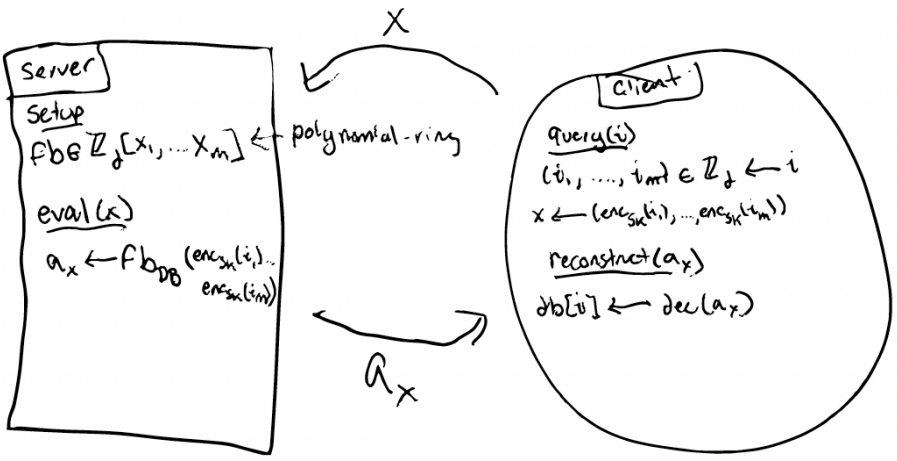
This [5] paper by Wei-Kai Lin, Ethan Mook, and Daniel Wichs explores the DEPIR problem by utilizing the RingLWE assumption and prior work by Kedlaya and Umans to preprocess polynomials for fast evaluation of queries. This not only speeds up the preprocessing step, but also allows for a more used assumption than HPN from Towards DEPIR [4].

6.1 Constructions: FHE vs Ram-FHE



The primary issue with FHE is the evaluation time, since it is done by converting to circuits, the runtime ends up being large than the $|DB|$. On the other hand, the runtime of Ram-FHE is dependent on the runtime of the program.

6.2 Construction: ASHE (Algebraic SHE)



The figure above depicts the working of SHE (Somewhat homomorphic encryption) to construct a DEPIR setup. By picking specific parameters, the work of Kedlaya and Umans is utilized in a preprocessing step to achieve $N^{1+\epsilon}$ setup and polylog(N) evaluation runtime. The issue with SHE is we can't preprocess for encrypted inputs. This is worked through by utilizing ASHE, where we lift the polynomial to the ring (i.e. reinterpret $u \in Z_d$ as an element of the ring R).

6.3 Updatable DEPIR

Uses same syntax as DEPIR except introduces update defined where:
$\tilde{DB}' \leftarrow Update(\tilde{DB}, i, b)$: Takes as input a location $i \in [N]$ and a bit b. Uses random-access to the data structure $\tilde{DB}$ and updates it to $\tilde{DB}'$.
The update in construction works by having a hierarchy of $L = log(N)$ levels where each level l contains a pre-processed database, $\tilde{DB}_l$. In order to read, the client would make a query at each level of $\tilde{DB}_l$.
The construction utilizes binary search and has each level padded to insure that there is always worst case searching (thus the run time does not reveal anything), and is based on hierarchal ORAM

Round Optimal Updatable DEPIR

To achieve Round Optimal Updatable DEPIR we rely on Ram-FHE and have the server run binary search of the preprocessed database. By combining DEPIR and Ram-FHE we can round-optimal writing DEPIR.

7 Comparison

We will compare the differences each papers construction. For all comparisons, assume N is the size of the database λ is the security parameter, and O_λ hides polylogarithmic λ factors.

	Sublinear PIR [1]	PIANO [2]	PANDAS [3]	Towards DEPIR [4]	RingLWE DEPIR [5]
Servers	1 & 2	1	1	1	1
Comm. Complexity	O($\sqrt{N}$)	O($\sqrt{N}$)	O_λ(polylog$N$)	O_λ(polylog$N$)	O_λ(polylog$N$)
Server Work	O(N)	O($\sqrt{N}$)	$O_\lambda(N^\epsilon \cdot \text{polylog}N)$	O_λ(polylog$N$)	O_λ(polylog$N$)
Clients	Unlimited	Unlimited	Bounded	Unlimited	Unlimited
State	No	No	Yes	No	No
Assumption	FHE	OWF	FHE	HPN	RingLWE
Access	R	R	R/W	R	R/W

Runtime

Both Sublinear PIR [1] and Piano [2] have a $\sqrt{n}$ amortized runtime for the server computation. But unlike [4] and [5], [1] and [2] are not sublinear in both the client and server work as they require a large client preprocessing (of the whole database) in order to generate the hints (i.e. do not meet DEPIR efficiency).

Memory

[1] and [2] require the client to hold onto hints: Sublinear PIR [1] requires $\sqrt{N}$, PIANO [2] requires a little more with a small constant multiplied by $\sqrt{N}$ (to store a backup table and replacement entries). Piano has this extra memory due to its use of 3 different data structures which includes the backup table and replacement entries. The primitives used to compress the database storage is pseudorandom functions for PIANO and shifting pseudorandom puncturable sets in Sublinear PIR.

Collusion

The Sublinear PIR assumes no collusion in the offline/online server setup. If the servers colluded, the online server could calculate the requested index by seeing which hint the request subset of indices came from. In the single server setting, the FHE assumption protects against 'self-colusion'. Therefore all the other constructions work trivially under collusion since it is single server (thus lacks another server to collude with).

Syntax

The syntax of the Sublinear PIR involves a setup, hint, query, answer and reconstruct, which breaks down requesting an index into creating a correct query, and reconstructing the provided answer to the query. On the other hand, PIANO just has a setup, query, and refresh in it's syntax. Query handles both building the query and construction the answer based on the servers answer. The syntax of both methods separate refreshing into its own function. All the DEPIR constructions have a processing step to process the database on the server side in order to obtain the double-efficiency in the per request computation work.

Security

The security games are either tackled through guessing games, where adversary has to guess the correct index, and we show that the distribution of guesses remains the same regardless of what index the user requests, or through simulations, where we argue that through small changes from the real situation to our constructed ideal situation called hybrids, the adversary has the same view.

References

[1] H. Corrigan-Gibbs and D. Kogan, *Private information retrieval with sublinear online time*, Cryptology ePrint Archive, Paper 2019/1075, `https://eprint.iacr.org/2019/1075`, 2019. [Online]. Available: `https://eprint.iacr.org/2019/1075`.

[2] M. Zhou, A. Park, E. Shi, and W. Zheng, *Piano: Extremely simple, single-server pir with sublinear server computation*, Cryptology ePrint Archive, Paper 2023/452, `https://eprint.iacr.org/2023/452`, 2023. [Online]. Available: `https://eprint.iacr.org/2023/452`.

[3] A. Hamlin, R. Ostrovsky, M. Weiss, and D. Wichs, *Private anonymous data access*, Cryptology ePrint Archive, Paper 2018/363, `https://eprint.iacr.org/2018/363`, 2018. [Online]. Available: `https://eprint.iacr.org/2018/363`.

[4] R. Canetti, J. Holmgren, and S. Richelson, *Towards doubly efficient private information retrieval*, Cryptology ePrint Archive, Paper 2017/568, `https://eprint.iacr.org/2017/568`, 2017. [Online]. Available: `https://eprint.iacr.org/2017/568`.

[5] W.-K. Lin, E. Mook, and D. Wichs, *Doubly efficient private information retrieval and fully homomorphic ram computation from ring lwe*, Cryptology ePrint Archive, Paper 2022/1703, https://eprint.iacr.org/2022/1703, 2022. [Online]. Available: https://eprint.iacr.org/2022/1703.