Simple and Practical Single-Server Sublinear Private Information Retrieval

Muhammad Haris Mughees Ling Ren

University of Illinois Urbana-Champaign
{mughees, renling}@illinois.edu

1 Introduction

Private Information Retrieval (PIR) allows a client to fetch an entry from a public database on a server without revealing which entry the client is interested in [1]. Recent works in amortized sublinear PIR have made strides and demonstrated great potential. Despite the inspiring progress, the new paradigm is still faced with various challenges, including large client storage, expensive offline phases, high communication, and poor practical efficiency. Many schemes have to resort to heavy-weight theoretical tools, non-colluding servers, parallel instances, or restricted client query sequences. This paper proposes a simple and practical single-server sublinear PIR scheme with a small response overhead and without the aforementioned drawbacks.

Most amortized sublinear PIR schemes follow the blueprint of Corrigan-Gibbs and Kogan [3]. The client privately retrieves hints in an offline phase. Each hint involves a subset S of $\sqrt{N}$ random distinct indices within $[1, N]$ where N is the number of entries in the database. For each hint, the client stores the subset S and the corresponding parity $\sum_{i \in S} \mathsf{DB}[i]$ where $\mathsf{DB}[i]$ is the i-th entry of the database and the summation represents XOR. In the online phase, if the client wants to retrieve i-th entry, it finds a subset S that contains i. Since the client stores the parity of entries in S, ideally, it just needs to ask the server for the parity of $S \setminus \{i\}$, from which it can easily recover $\mathsf{DB}[i]$.

However, the above high-level strategy, the client always sends the server a subset that does not contain the queried index i. This is insecure because the server learns that the queried entry is not one of those in $S \setminus \{i\}$. To fix this problem, Corrigan-Gibbs and Kogan suggest that the client occasionally removes an index other than i from S. In this case, the client loses the ability to retrieve the queried entry i. To compensate for this loss of correctness, λ instances of their protocol are executed in parallel to achieve an exponentially small (in λ) failure probability. This blows up all efficiency metrics (communication, computation, client storage) by a factor of λ and renders the scheme impractical.

Lazzaretti and Papamathou [5] give a clever way to avoid the λ blowup using partition-based hints. Their scheme requires two non-colluding servers. Zhou et al. [7] adapt their scheme to a single server but can only handle client queries that are not adversarially influenced. While this restriction may be valid in certain scenarios, it does not always hold true and is a big departure from the standard PIR model. Another drawback of Lazzaretti and Papamathou (and inherited by Zhou et al.) is that the response blowup becomes $O(\sqrt{N})$, which would be prohibitive for databases with large entries. Though this problem could in theory be mitigated by invoking another regular single-server PIR scheme, that would not be efficient in practice.

In this paper, we propose new techniques in hint construction and usage and obtain a simple and lightweight single-server sublinear PIR. Our scheme has small amortized response and close to optimal online response, which is only twice that of simply fetching the desired entry without privacy. Our scheme does not require parallel instances and supports arbitrary client query sequences.

Table 1 gives a comparison with recent practical amortized sublinear PIR schemes in terms of asymptotic efficiency. We exclude schemes that require heavy theoretical tools such as CHK22 [2] and ZLTS23 [6] which require gate-by-gate FHE and privately programmable pseudorandom functions, respectively. Our scheme also enjoys good concrete efficiency. For a 128 GB database with 64-byte entries, each query consumes only 117 KB of communication and 7.5 milliseconds of computation, amortized.

Table 1: Comparison with recent amortized practical sublinear PIR schemes.

Scheme	Number of Servers	Amortized communication Request	Response	Storage Client	Amortized computation Client	Server
CK20 [3]	2	$O(\lambda\sqrt{N})$	$O(\lambda)$	$O(\lambda^2\sqrt{N})$	$O(\lambda\sqrt{N})$	$O(\lambda\sqrt{N})$
KC21 [4]	2	$O(\log N)$	$O(1)$	$O(\lambda\sqrt{N})$	$O(N)$	$O(\sqrt{N})$
LP23 [5]	2	$O(\log N)$	$O(\sqrt{N})$ [2]	$O(\lambda\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$
ZPSZ23 [3] [7]	1	$O(\sqrt{N})$	$O(\sqrt{N})$	$O(\lambda\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$
This paper	1	$O(\sqrt{N})$	$O(\sqrt{N}/\lambda)$	$O(\lambda\sqrt{N})$	$O(\sqrt{N})$	$O(\sqrt{N})$

[1] Request size and client computation are measured in words of size λ or $\log N$. Response, client storage, and server computation are measured in entry size (also making response a blowup over the insecure baseline).
[2] LP23 [5] can invoke a second PIR to reduce response, but a variant without this second PIR gives better practical efficiency and makes a more fair comparison.
[3] ZPSZ23 [7] requires client queries to have no adversarial influence, making it weaker than standard PIR.

2 Algorithm

The new hint. The key idea is a new type of hint that eliminates the information leakage due to the absence of the desired index. This immediately obviates the need for parallel instances. Applied to the partition-based hints [5], it avoids the large responses and works with a single server. For convenience, we will directly describe our idea on top of the partition framework. A database of size N is divided into $\sqrt{N}$ partitions each of size $\sqrt{N}$. For now, we assume $\sqrt{N}$ is an even integer.

Let $\mathcal{R}$ denote the following distribution: first select $\sqrt{N}/2+1$ random partitions (without replacement) out of the $\sqrt{N}$ total partitions; then pick one random index from each of these $\sqrt{N}/2+1$ partitions. In other words, a sample from $\mathcal{R}$ consists of $\sqrt{N}/2+1$ random indices from $\sqrt{N}/2+1$ random partitions, one index per partition.

A hint in our algorithm consists of a sample from $\mathcal{R}$ and its corresponding parity. Similar to previous works, the client needs to store M hints (M is specified later). For each $j = 1, 2, \ldots, M$, the client samples $S_j \leftarrow \mathcal{R}$ and stores S_j along with $\sum_{i \in S_j} \mathsf{DB}[i]$ as one hint. Usage of the hint resembles previous works in principle. When the client requests entry i, the client looks for a subset S_j that contains i. The client sends $S_j \setminus \{i\}$ to the server. The server returns the parity for $S_j \setminus \{i\}$. The client easily recovers $\mathsf{DB}[i]$ since it has been storing the parity for S_j. Similar to previous works, we need $M = \lambda\sqrt{N}$ where λ is a security parameter so that a subset containing the requested index can be found with overwhelming (in λ) probability.

Eliminating the leakage. Now we tackle the main challenge mentioned in the introduction. With the approach described so far, the subset sent by the client involves $\sqrt{N}/2$ random partitions and contains one random index from each of them. However, since the client always removes the requested index from the subset, the server now learns that the desired entry is definitely *not* in any of these $\sqrt{N}/2$ partitions.

Our main idea to address this leakage is for the client to additionally send a dummy subset that contains one random index from each of the other $\sqrt{N}/2$ partitions. The client also randomly permutes the two subsets, so the server cannot tell apart the real one from the dummy one. This perfectly hides which partition contains the desired entry. In fact, the client's request now reveals no information about the desired entry. The client sends two subsets, each covering $\sqrt{N}/2$ partitions. A random index is picked from each partition, so we only need to ensure that the groupings of the partitions leak no information. To this end, the dummy subset bundles the partition of interest with $\sqrt{N}/2-1$ other random partitions, and the real subset covers the remaining $\sqrt{N}/2$ partitions. This is indistinguishable from a purely random arrangement that would anyway group the partition of interest with $\sqrt{N}/2-1$ random partitions.

The following modifications to our protocol are natural from the above idea. The client now sends the two subsets, permuted, to the server. The server returns the two parities corresponding to the two subsets. The client discards the dummy parity and uses the real subset parity to recover the desired entry. As a result, the online response size of our scheme is already close to optimal.

Hint replenishment. After each online PIR request, the client needs to replenish one hint since it has consumed one. Moreover, the replenished hint must follow the same distribution as the one just consumed. This can be done similarly as in [2] using backup hints. In the offline phase, the client retrieves not only the $\lambda\sqrt{N}$ primary hints but also $\lambda\sqrt{N}$ backup hints. A backup hint contains one fewer index than a main hint in its subset. After the client makes a PIR query for entry i, it finds a backup hint that contains no index from i's partition. The client then adds i to the subset and $\mathsf{DB}[i]$ (which the client has just retrieved) to the parity. The new subset and parity now form a regular hint that follows the same distribution as the consumed one, i.e., has i in the subset. The client can make $\approx \lambda\sqrt{N}/2$ online queries between two offline phases. Even if the client keeps requesting entries from the same partition, it will not run out of backup hints that skip that partition, except for negligible (in $\sqrt{N}$) probability.

Further simplification to the hints. So far, we have been assuming that the subset of every hint has a size exactly $\sqrt{N}/2+1$. Sampling such a subset pseudorandomly can be done in theory but causes inconvenience for practice. We observe that the hint construction rule can be relaxed as follows. To construct a backup hint, for each partition, pick a random index from it or skip it with half-half probability independently. To get a main hint, pick one extra random index from a random skipped partition. Put in another way, a subset now has size $D+1$ where D is a binomial random variable, i.e., the number of selected partitions if each partition is selected independently with half probability.

With this change, the real subset has size D where D follows the aforementioned binomial distribution. The dummy subset's size is $\sqrt{N}-D$ and follows the same binomial distribution. Once permuted, the real subset and the dummy subset are indistinguishable. The security of the PIR scheme is thus unaffected.

Offline phase. In the offline phase, the client needs to retrieve main hints and backup hints in a private manner. This can be achieved in a few ways. The simplest and most practical way is perhaps to stream the entire database, one partition at a time. With the independent partition selection rule, we just need a few pseudorandom bits and numbers to determine, for each main or backup hint, which index, if any, should be drawn from the current partition. This streaming offline phase costs N communication and $O(\lambda N)$ computation; amortized over $\approx \lambda\sqrt{N}/2$ online queries, these give the results in Table 1.

Pseudocode of the complete algorithm is given in Algorithms 1 and 2.

3 Evaluation

We implement our algorithm in C++. We use AES as the pseudorandom function and use Intel's AES-NI instructions. A 128-bit PRF output provides randomness for up to eight hints. Our implementation uses a single thread. The parameter λ is set to 80. We evaluate our algorithm by running the client and simulating the server on an Intel Core i7-9750H CPU at 2.60 GHz. Table 2 gives a comparison with ZPSZ23 [7] in terms of concrete storage, communication, and computation costs.

Table 2: Comparison with ZPSZ23 [7]. "Am." stands for amortized.

Scheme	Database	Storage at Client	Communication Am. Offline	Communication Online	Computation Offline (Am.)	Computation Online
ZPSZ23	2 GB (2^{28} 8-byte)	75 MB	6.6 KB	128 KB	3369 s (10.6 ms)	9.0 ms
This paper	2 GB (2^{28} 8-byte)	30 MB	3.2 KB	30 KB	376 s (0.7 ms)	1.7 ms
ZPSZ23	100 GB ($2^{30.6}$ 64-byte)	719 MB	120.5 KB	900 KB	356 min (24.6 ms)	33.3 ms
This paper	128 GB (2^{31} 64-byte)	681 MB	51 KB	66 KB	77 min (2.2 ms)	5.3 ms

[1] ZPSZ23 results are taken from Table 1 in their paper [7], which also reports local evaluation results.
[2] We compare our results for a 128 GB database with their results for a 100 GB database.
[3] While offline communication is identical for the two schemes (streaming the database), our amortized offline communication is smaller because we support more online queries per offline phase.

Algorithm 1 The steaming offline algorithm

for each main or backup hint j **do**
  Set unique hint ID $h_j = j$
  Initialize parity $P_j = 0$
  For a main hint, set the extra index e_j to a random index from a random unselected partition
end for

for $k = 0 : \sqrt{N} - 1$ **do**
  Download $\mathsf{DB}[k\sqrt{N} : (k+1)\sqrt{N} - 1]$ from the server ▷ download partition k
  for each main or backup hint j **do**
    $(b, r) \leftarrow \mathsf{PRF}(h_j, k)$ ▷ b is a random bit, r is a random offset within 0 to $\sqrt{N} - 1$
    if $b == 1$ **then** ▷ partition k is selected
      $P_j = P_j \oplus \mathsf{DB}[r + k\sqrt{N}]$ ▷ index r is picked from partition k
    else if j is a main hint AND $\lfloor e_j/\sqrt{N} \rfloor == k$ **then** ▷ e_j is in partition k
      $P_j = P_j \oplus \mathsf{DB}[e_j]$
    end if
  end for
end for

Algorithm 2 The online algorithm

Input: requested index i

$\ell = \lfloor i/\sqrt{N} \rfloor$ ▷ ℓ is the partition that i belongs to
Find main hint j such that $e_j == i$ or $b \cdot (r + \ell\sqrt{N}) == i$ where $(b, r) \leftarrow \mathsf{PRF}(h_j, \ell)$ ▷ hint j contains i

Initialize $S = \emptyset$ and $S' = \emptyset$ ▷ S will be the real subset and S' will be the dummy subset
for $k = 0 : \sqrt{N} - 1$ **do**
  $(b, r) \leftarrow \mathsf{PRF}(h_j, k)$
  if $b == 1$ **then**
    $S = S \cup \{r + k\sqrt{N}\}$
  else if e_j belongs to partition k **then**
    $S = S \cup \{e_j\}$
  else
    $S' = S' \cup \{\mathsf{rand}() + k\sqrt{N}\}$ ▷ add a random index from partition k to the dummy subset
  end if
end for
$S = S \setminus \{i\}$ ▷ remove the requested index from the real subset
$S' = S' \cup \{\mathsf{rand}() + \ell\sqrt{N}\}$ ▷ add a random index from partition ℓ to the dummy subset

Send (S, S') or (S', S) to the server with half-half probability ▷ permute the real and dummy subsets
Receive the two subset parities P and P' from the server ▷ in the order S and S' are sent
Return $P \oplus P_j$ as $\mathsf{DB}[i]$

Find backup hint j' such that $b == 0$ where $(b, \_) \leftarrow \mathsf{PRF}(j', \ell)$ ▷ backup hint j' skips partition ℓ
$h_j = j'$
$e_j = i$
$P_j = P_{j'} \oplus \mathsf{DB}[i]$ ▷ promote backup hint j' to be main hint j by adding i
Delete backup hint j'

References

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

[2] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 3–33. Springer, 2022.

[3] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 44–75. Springer, 2020.

[4] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In *30th USENIX Security Symposium*. USENIX Association, 2021.

[5] Arthur Lazzaretti and Charalampos Papamanthou. Treepir: Sublinear-time and polylog-bandwidth private information retrieval from ddh. *Cryptology ePrint Archive*, 2023.

[6] Mingxun Zhou, Wei-Kai Lin, Yiannis Tselekounis, and Elaine Shi. Optimal single-server private information retrieval. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 395–425. Springer, 2023.

[7] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server pir with sublinear server computation. *Cryptology ePrint Archive*, 2023.