Zelda: Efficient Multi-server Preprocessing PIR with Unconditional Security

Ashrujit Ghoshal[1], Mingxun Zhou[2], Bo Peng[3], and Elaine Shi * [4]

[1]Indian Institute of Technology Madras
[2]The Hong Kong University of Science and Technology
[3]Peking University
[4]Carnegie Mellon University

December 4, 2025

Abstract

Private Information Retreival (PIR) schemes without preprocessing are known to incur linear server computation per client query. Several recent works have shown that by relying on a one-time preprocessing phase, we can get around this barrier, and achieve sublinear computation per query without relying on any cryptographic assumptions. Beimel et al. (CRYPTO'00) first showed a family of schemes whose bandwidth and computation per query scale as fast as $n^{O(1/S)}$ where S denotes the number of servers and n denotes the database size. Unfortunately, their schemes are not practical partly because the servers must each store an encoded version of the database, and the encoding length grows sharply as we increase S. The recent work of Singh et al. (TCC'24) showed how to achieve similar bandwidth scaling but without the server space blowup. To get this, they rely on a different type of preprocessing called client-specific preprocessing, where the stateful client stores some hints and the servers store only the original database. Unfortunately, Singh et al.'s result is completely impractical due to the reliance on Dvir and Gopi's PIR as a building block.

We propose Zelda (short for ZEro-Leakage Data Access), the *first* concretely efficient, information-theoretic multi-server PIR scheme with sublinear computation. Our work makes both theoretical and practical contributions. On the theoretical front, we devise a unified framework for constructing multi-server PIR with client-specific preprocessing. This gives us a parametrizable family of schemes that asymptotically outperform all prior constructions in the same setting, including Singh et al. (TCC'24) and Ishai et al. (CRYPTO'24). On the practical front, Zelda is conceptually simple, self-contained, and does not rely on any underlying PIR as a building block. We implemented Zelda and open sourced our code. We compared the concrete performance of Zelda with a state-of-the-art PIR scheme called QuarterPIR (Eurocrypt'24), which relies on pseudorandom functions for security. Experimental results show that Zelda outperforms QuarterPIR in terms of online response time and client space (assuming typical fiber optical links), at the price of increased costs for offline maintenance operations.

1 Introduction

Private Information Retrieval (PIR) was introduced in the seminal work of Chor, Goldreich, Kushilevitz and Sudan [CGKS95]. It is a cryptographic mechanism that allows a client to retrieve

*Randomized Author Ordering. Email: ashrujit@cse.iitm.ac.in, mingxunz@ust.hk, bo.peng@stu.pku.edu.cn, rshi@andrew.cmu.edu .

the value of a public database $\mathsf{DB} \in \{0,1\}^n$ at an index $\mathsf{idx} \in \{0,1,\ldots,n-1\}$, from one or more non-colluding servers, *without* the server(s) learning what idx is. PIR has many promising applications like private web search [HDCG+23], private DNS [Fea], private contact discovery [DRRT18] etc., and hence has been studied over the course of more than two decades.

Classical PIR schemes [CGKS95, Cha04, GR05, CMS99, CG97, KO97, Lip09, OS07, Gas04, BFG03, SC07, OG11, MCG+08, MG07, HHCG+22, MW22] do not use any preprocessing. Unfortunately, Biemel et al. [BIM00] proved that for any classical PIR scheme, a client query *must* incur total server computation that is linear in the size of the database. This linear computation barrier holds even in the multi-server setting, and it reveals a fundamental efficiency bottleneck for scaling PIR to realistic data sizes.

A paradigm shift: preprocessing PIR. Pioneered by the work of Beimel et al. [BIM00] and Corrigan-Gibbs and Kogan [CK20], our community has made a significant recent push to overcome the linear server computation barrier using preprocessing [WY05, PPY18, CHR17, BIPW17, SACM21, CHK22, ZLTS23, LMW23, LP23a, ZPSZ24, LP23b, RMS24, GZS24, ISW24, SWZ24]. In its most general form, PIR with preprocessing allows the server and each (stateful) client to engage in a *one-time preprocessing* protocol, and afterwards each client may make an unbounded (i.e., a-priori unknown) number of queries. Henceforth, if each client performs a separate preprocessing with the server, we call the model *client-specific preprocessing* [CK20]. If the preprocessing does not involve the clients such as the model considered by Beimel et al. [BIM00], we call it *public preprocessing*. Recent works showed that both preprocessing models allow us to overcome the linear computation barrier [WY05, CHR17, BIPW17, SACM21, CHK22, ZLTS23, LMW23, LP23a, ZPSZ24, LP23b, RMS24, GZS24, ISW24, SWZ24], and moreover, information theoretic security is possible in the multi-server setting [BIM00, WY05, GLM+24, LLFP24].

Information-theoretic preprocessing PIR was first explored in the public preprocessing setting by Biemel et al. [BIM00], Woodruff and Yekhanin [WY05], and several subsequent works [GLM+24, LLFP24]. These works [BIM00, WY05, GLM+24] showed for any $S = O(1)$, how to construct an S-server PIR scheme with $O(n^{(1+\epsilon)/S})$ per-server computation and bandwidth while consuming a polynomial amount of server space, where $\epsilon \in (0,1)$ is an arbitrarily small constant. Although it is desirable for the computation and bandwidth costs to scale as fast as $O(n^{(1+\epsilon)/S})$ as the number of server S increases, this line of work suffers from a major drawback — the server space is a large polynomial $n^{\Omega(S^{1/\epsilon})}$ which grows sharply as we increase the number of servers. Such prohibitively large server space makes these schemes completely impractical for moderate to large databases.

In comparison, information-theoretic PIR in the client-specific preprocessing setting has received relatively little attention. To the best of our knowledge, only a couple very recent works have explored this setting. Ishai, Shi, and Wichs [ISW24] considered the special case of having exactly *2 servers*, and they showed how to get a scheme with $\widetilde{O}(n^{2/3})$ server computation, $\widetilde{O}(n^{1/2})$ client computation, and $O(n^{1/3})$ bandwidth per query, while consuming $\widetilde{O}(n^{2/3})$ client space.

A subsequent work by Singh et al. [SWZ24] explored *multi-server* client-specific preprocessing PIR. For $4 \le S \le \log n$ and S divisible by 4, they show how to get an S-server scheme that achieves $\widetilde{O}(\sqrt{n})$ client and server computation and $\widetilde{O}(n^{2/S+o(1)})$ bandwidth per query, while consuming $\widetilde{O}(n^{1/2+2/S})$ client space. In other words, Singh et al.'s elegant result shows that with client-specific preprocessing, we can achieve similar bandwidth scaling as public preprocessing, but without the excessive blowup in server space as S grows. Despite the desirable asymptotics, their scheme suffers from a couple major drawbacks. First, it employs the 2-server PIR scheme of Dvir and Gopi [DG16] as a building block, and thus is completely impractical. Second, they need at least 8 servers to get a non-trivial result — otherwise, the client space would be at least linear in the database size, and the client might as well just store the entire database itself. Singh et al. [SWZ24] additionally

Table 1: **Comparison:** we unify and strictly improve over prior works [ISW24, SWZ24].

Scheme	Compute		BW	Client space	# servers	Concrete eff.
	server	client				
Ours	$\widetilde{O}(n^{\epsilon})$	$\widetilde{O}(n^{1-\epsilon}+n^{\epsilon/S})$	$\widetilde{O}(n^{\epsilon/S})$	$\widetilde{O}(n^{1-\epsilon+\epsilon/S})$	$S=O(\log n)$	✓
[ISW24]	$\widetilde{O}(n^{2/3})$	$\widetilde{O}(n^{1/2})$	$\widetilde{O}(n^{1/3})$	$\widetilde{O}(n^{2/3})$	2	✗
Ours ($\epsilon=\frac{2}{3}$)		$\widetilde{O}(n^{1/3})$				✓
[SWZ24]	$\widetilde{O}(n^{1/2})$	$\widetilde{O}(n^{1/2})$	$n^{2/S+o(1)}$	$\widetilde{O}(n^{1/2+2/S})$	$S=O(\log n)$	✗
[SWZ24]			$\widetilde{O}(n^{1/2})$	$\widetilde{O}(n^{1/2+1/S})$		✗
Ours ($\epsilon=\frac{1}{2}$)			$\widetilde{O}(n^{1/2S})$	$\widetilde{O}(n^{1/2+1/2S})$		✓

✓ means that the scheme has plausible concrete efficiency but not supported by implementation and experimental results. $\widetilde{O}(\cdot)$ hides polylogarithmic factors.

show a variant that avoids Dvir and Gopi's PIR and reduces the number of servers by a factor of 2, but at the price of $\widetilde{O}(\sqrt{n})$ bandwidth per query, meaning that the resulting scheme no longer scales w.r.t. the number of servers S in any dimension.

Finally, Ishai et al. [ISW24] and Singh et al. [SWZ24] rely on rather different algorithmic techniques, and we are not aware of any direct way to unify these seemingly disparate techniques to obtain non-trivial improvements.

In this paper, we revisit information-theoretic multi-server PIR with client-specific preprocessing, and ask the following natural questions:

- Can we get a *concretely efficient* scheme whose bandwidth scaling matches or improves over the best known public-preprocessing scheme, but without the blowup in server space?
- Can we have a *unified framework* for constructing PIR schemes in this setting such that we can generalize and improve over previous results [ISW24, SWZ24]?

1.1 Our Results and Contributions

We answer the above questions affirmatively. We construct Zelda (short for ZEro-Leakage Data Access), a family of multi-server PIR schemes in the client-specific preprocessing setting, without relying on any cryptographic assumptions. Zelda asymptotically outperforms the prior works by Ishai et al. [ISW24] and Singh et al. [SWZ24] as shown in Table 1. Not only so, Zelda is *conceptually simple*, self-contained, and does not rely on any underlying PIR as a building block. To the best of our knowledge, Zelda *is the first information-theoretic multi-server PIR scheme shown to be concretely efficient.* All prior works in both the public and client-specific preprocessing settings are of a theoretical nature.

Our result, in its most general form, is presented in the following theorem.

Theorem 1.1 (Our main result)**.** *Fix an arbitrary $\epsilon \in (0,1)$. Suppose n is reasonably large such that $1+\frac{\epsilon}{S}\cdot\log_2 n < n^{\epsilon/2S}$. Let $\alpha(n)$ be any function of n such that $\alpha(n)=\omega(S/\epsilon)$. Then, there exists an information-theoretic S-server PIR scheme that satisfies non-adaptive correctness, with the following performance bounds. The one-time preprocessing incurs $O(\epsilon\cdot n(\log n/S)\cdot\alpha(n))$ server computation and $O(\epsilon\cdot n^{1-\epsilon+\epsilon/S}\log n\cdot\alpha(n))$ communication. Afterwards, every query incurs $O(S\cdot n^{\epsilon/S}\cdot\alpha(n))$ online and offline bandwidth, $O(n^{\epsilon-\epsilon/S}\cdot\alpha(n))$ online server computation, $O(n^{\epsilon}\cdot$*

$\alpha(n))$ *offline server computation,* $O((\epsilon \cdot n^{1-\epsilon} \log n + S^2 \cdot n^{\epsilon/S}) \cdot \alpha(n))$ *online client computation, and* $O(S \cdot n^{\epsilon/S} \cdot \alpha(n))$ *offline client computation. Further, the client space is bounded by* $O(\epsilon \cdot n^{1-\epsilon+\epsilon/S} \log n \cdot \alpha(n))$.

The above Theorem 1.1 is stated with general choices of ϵ and S. As shown in Table 1, choosing $S = 2$ and $\epsilon = 2/3$ gives an instantiation that strictly outperforms Ishai et al. [ISW24]; and choosing $\epsilon = 1/2$ gives a family of schemes parametrized by S that strictly outperforms Singh et al. [SWZ24]. In other words, our result unifies and strictly improves earlier works [ISW24, SWZ24].

In Theorem 1.1, *the tradeoff between the client space and online server computation is also nearly optimal* barring polylogarithmic factors. In particular, earlier works [CK20, CHK22] proved the following lower bound result: assuming that the servers do not store any extra data besides the original database, then the product of the client space and online server computation must be at least $\Omega(n)$.

We also propose a new technique for upgrading Theorem 1.1 to achieve adaptive correctness, as stated in the following corollary. The upgrade incurs only an S-fold cost in all dimensions.

Corollary 1.2 (Informal: adaptive correctness)**.** *We can upgrade the scheme in Theorem 1.1 to achieve adaptive correctness with an* S*-fold multiplicative blowup in all costs.*

Concrete performance. We implemented the 2-server version of Zelda and made the source code open source at https://github.com/p-b-p-b/Zelda. We also compare our costs relative to a state-of-the-art 2-server PIR scheme called QuarterPIR [GZS24] which relies on pseduorandom functions (PRFs):

- *Better online response time over fiber optic links.* Our evaluation results suggest that over a typical fiber optic link of 1Gbps bandwidth and a 10ms ping latency , Zelda achieves 35ms response time for a moderately sized database of 128GB with entry size 1 kilobyte, while QuarterPIR requires 55ms. Specifically, Zelda's online computation $O(n^{1/3} \cdot \alpha(n))$ is asymptotically better than QuarterPIR's $O_\lambda(\sqrt{n})$ where λ is the computational security parameter. Although Zelda's $O(n^{1/3} \cdot \alpha(n))$ communication cost (i.e., amount of data transmitted) is asymptotically worse than QuarterPIR's $O_\lambda(n^{1/4})$, when running the scheme over fiber optic links, the end-to-end ping latency and computation time are dominating, not the transmission time due to bandwidth.

- *Reduction in client space.* Further, Zelda achieves 51% concrete reduction in client space relative to QuarterPIR for a moderately large database as mentioned above, despite being slightly worse asymptotically (Zelda's client space is $\widetilde{O}(n^{2/3})$ as opposed to QuarterPIR's $\widetilde{O}(n^{1/2})$). This is because Zelda can benefit a concrete optimization that balances the payload and metadata parts of the storage cost for long entries, whereas QuarterPIR inherently cannot benefit from such an optimization.

- *Better preprocessing costs.* Zelda's one-time preprocessing is $2\times \sim 6\times$ faster than that of QuarterPIR since our preprocessing is free of cryptographic operations whereas QuarterPIR's preprocessing needs to perform PRF evaluations.

- *Limitation: increased offline costs.* The above benefits come at the price of increased offline costs relative to QuarterPIR, both asymptotically and concretely. However, since the offline costs are for background maintanence operations, they do not factor into the response time perceived by the user.

1.2 Technical Highlights

Novel multi-way privately puncturable random set. One of our core contributions is a new way to construct an information theoretic object called Multi-Way Privately Puncturable Random Set (mPPRS). An mPPRS allows us to sample a random subset of indices from $\{0, 1, \ldots, n-1\}$ according to some distribution, summarized by a succinct key denoted msk. Later, we can remove a particular element idx from the set by puncturing the key into S punctured keys denoted $\mathsf{sk}_0, \ldots, \mathsf{sk}_{S-1}$, and each punctured key reveals no information about the element that is removed. Each punctured key sk_s defines a list of subsets, and we can recover the original set minus the element idx by choosing one subset from each of the S lists.

Singh et al. [SWZ24] first defined mPPRS and used it to construct a multi-server PIR scheme. However, their scheme is inefficient. Their punctured key may generate up to $\sqrt{n}$ subsets, and when the mPPRS is used to construct a PIR, it means that each server may need to send up to $\sqrt{n}$ parities of different subsets of the database to the client. To avoid this excessive bandwidth blowup, they then use Dvir and Gopi [DG16] to replicate each server twice, such that the client can fetch the desired parity without revealing which one is desired.

Our new mPPRS construction is not only conceptually simple, but also asymptotically improves the efficiency of Singh et al. [SWZ24]. Specifically, the number of subsets generated by all punctured keys are bounded by $n^{1/2S}$ where S is the number of servers (assuming that $\epsilon = 1/2$). This way, we can afford to have each server directly send back $n^{1/2S}$ parities, and we need not rely on Dvir and Gopi's PIR [DG16].

Partially broken hint technique. Besides our new mPPRS scheme, we also propose a new "partially broken hint" technique that makes non-blackbox usage of our specific mPPRS, allowing us to save a logarithmic factor in our PIR scheme in comparison with the standard "broken hint" technique proposed in the recent literature on client-specific preprocessing PIR [GZS24, ISW24]. Specifically, in the PIR scheme, the client stores a table of hints, and for each query, it finds a hint that matches the present query. Once the hint is consumed, for privacy reasons, it is important to replace with a fresh random hint subject to matching the current query. However, because this refreshed hint is generated on the fly by the client on its own, the client does not have the appropriate advice obtained from preprocessing to make this hint functional. Such a hint is tagged as *broken*, and if consumed later, will result in an error. The standard literature [GZS24, ISW24] then relies on parallel repetition to boost the correctness probability to all but negligible. In our new "partially broken hint" technique, a refreshed hint is not completely broken for all future queries, it is only broken if some specific queries are made, and it is unpredictable which queries it will break. This trick allows us to significantly reduce the error probability due to encountering a broken hint, and allows us to reduce the number of repetitions from super-logarithmic to only super-constant.

Achieving adaptive correctness. *Adaptive* correctness requires that even when a semi-honest adversary controlling all but one server can adaptively choose the queries for the client, the client's queries are guaranteed to return a correct answer with all but negligible probability. To the best of our knowledge, for the multi-server and client-specific preprocessing setting, all known schemes [RMS24, LP23b, ISW24, SWZ24, GZS24, ZPSZ24] with *non-trivial efficiency*, even computationally secure ones [RMS24, LP23b, GZS24, ZPSZ24], achieve only *non-adaptive* correctness, i.e., correctness is only guaranteed when the choice of the queries does not dynamically depend on

the views of the servers[1]. In the above, non-trivial efficiency means that the multi-server PIR's scheme cannot be matched by a single-server PIR scheme in the client-preprocessing setting, since we know techniques for achieving adaptive correctness in the single server setting [RMS24, ISW24]. Unfortunately, the adaptive correctness techniques for the 1-server setting are not applicable to multi-server schemes (with non-trivial efficiency), because the 1-server schemes [ISW24] perform a streaming pass for preprocessing, resulting in asymptotically higher offline bandwidth.

We propose a new technique for achieving adaptive correctness even when the (semi-honest) adversary controls all but one server. To get this result, we first construct a scheme that satisfies adaptive correctness as long as the preprocessing server is not corrupted. Then, in Section 10.1,we suggest an upgrade for achieving adaptive correctness when *any* of the $S-1$ servers are controlled by the semi-honest adversary. Our upgrade can also be used to make some of the earlier 2-server or multi-server constructions [RMS24, LP23b, SWZ24] adaptively correct.

Additional optimizations. We also devise additional concrete optimizations. For example, in Section 8.1, we devise a technique such that the client only needs to download super-constant number of entries from each server to compute the response. In Section 8.3, we carefully choose the parameters for our mPPRS to balance the payload and metadata parts of the client space.

1.3 Additional Related Work

We now review some additional related works.

PIR with server-side preprocessing. The more recent work of Ghoshal et al. [GLM+24] showed how to reduce the server space of the schemes of Beimel et al. [BIM00] and Woodruff and Yekhanin [WY05] by polynomial factors, while still enjoying the $n^{O(1/S)}$ bandwidth scaling. However, despite their significant asymptotic improvement, the server space increases rapidly with more servers, making these schemes impractical. In the same work, Ghoshal et al. [GLM+24] also showed how to let the bandwidth, computation, and server space scale equally as S grows, resulting in a scalable family of schemes with $n^{O(1/\log S)}$ bandwidth and computation per query, and $n^{1+O(1/\log S)}$ server space. The bandwidth scaling of their scalable family of schemes is worse than Zelda, and the concrete performance of their schemes remains unsubstantiated. Another recent work of Lazzaretti et al. [LLFP24] showed how to achieve $n^{o(1)}$ bandwidth and computation per query with $n^{1+o(1)}$ server space, assuming polylogarithmically many servers. Their work is of a theoretical nature and completely impractical due to the usage of a polynomial data structure [KU11, BGKM22] that incurs large concrete parameters.

If we allow computational assumptions, the ground-breaking work of Lin, Mook, and Wichs [LMW23] showed how to construct a single-server PIR schme with polylogarithmic bandwidth and computation per query, while consuming $n^{1+\epsilon}$ server space. Their scheme relies on server-side preprocessing, and its security relies on standard lattice-based assumptions. Unfortunately, subsequent works [OPPW23] have shown that the Lin, Mook, Wichs construction is nowhere near practical.

PIR with client-side preprocessing. PIR with client-side preprocessing was studied in a line of works [CK20, ZPSZ24, SACM21, CHK22, ZLTS23, LP23a, LP23b, HPPY24, GZS24, ISW24, SWZ24], beginning the elegant work of Corrigan-Gibbs and Kogan [CK20]. It is known that in the two-server setting, if we allow public-key cryptography or lattice-based assumptions, we can

[1]Although the work of Ren et al. [RMS24] claims to achieve adaptive correctness in the 2-server setting, upon careful examination, their adaptive correctness claim holds only if the corrupted server is the one that handles the queries. Adaptive correctness is not guaranteed if their preprocessing server is corrupted— see Section 10 for more discussions.

achieve polylogarithmic bandwidth and $\widetilde{O}(\sqrt{n})$ computation per query, while incurring $\widetilde{O}(\sqrt{n})$ client space [SACM21, LP23b].

A more recent line of works [ZPSZ24, RMS24, GZS24, ISW24, SWZ24, HPPY24, WR25], has focused on constructing single-server or multi-server PIR with client-specific preprocessing, while minimizing the cryptographic assumptions. Partly, the hope is that by minimizing the cryptographic assumptions, we can get better practical efficiency. Several works [ZPSZ24, RMS24, HPPY24, WR25] showed a 1-server scheme that relies only on pseudorandom functions (PRFs), achieving $\widetilde{O}(\sqrt{n})$ bandwidth and computation per query while consuming $\widetilde{O}(\sqrt{n})$ client space. The recent work of Ishai et al. [ISW24] proved that in the single-server setting, if we restrict ourselves to symmetric-key cryptography (or information-theoretic security), then the total bandwidth per query multiplied by the client space must be $\Omega(n)$. However, schemes in the multi-server setting is not subject to this limitation.

Both Plinko [HPPY24] and Wang et al. [WR25] focus on tight complexity, that is, removing the $\mathsf{poly}(\lambda, \log n)$ factors or obtaining tight tradeoff for parameter regimes other than $\tilde{O}(\sqrt{n})$. Plinko further supports fast database updates. However, due to their use of small-domain PRPs, their results are theoretical in nature, and they do not implement the schemes.

Ren et al. [RMS24] has an efficient implementation that achieves concretely practical performance. Piano [ZPSZ24] has similar asymptotic performance as Ren et al., but with slightly worse concrete performance. We compare with Ren et al. in the experiments section (Section 7), where we show that increasing the number of servers leads to reduced bandwidth and client storage in Zelda.

QuarterPIR [GZS24] showed that in the 1-servers setting, the online bandwidth (i.e., bandwidth on the critical path of obtaining the answer) can be improved to $\widetilde{O}(n^{1/4})$ while still relying on PRFs, but the offline bandwidth is still $\widetilde{O}(\sqrt{n})$. Besides the 1-server setting, QuarterPIR also showed a two-server scheme, based only from PRFs, with $\widetilde{O}(n^{1/4})$ total bandwidth and $\widetilde{O}(\sqrt{n})$ computation per query, while incurring $\widetilde{O}(\sqrt{n})$ client space — we use this scheme as a baseline of comparison for studying the concrete performance of Zelda in this work.

2 Technical Overview

Our work is inspired by Singh et al. [SWZ24]. For constructing a multi-client preprocessing PIR scheme, they propose a core information-theoretic object called multi-way privately puncturable random set (mPPRS).

Notation. In our work, it will be more convenient to start indexing at 0. Therefore, we define the notation $[\![k]\!] := \{0, 1, \ldots, k-1\}$.

2.1 Background: PIR from Multi-Way Privately Puncturable Random Set

Multi-way privately puncturable random set (mPPRS). An mPPRS scheme, parametrized by n, S, and ϵ, provides a way to succinctly encode random subsets of indices drawn from a particular distribution over $[\![n]\!]$. Each set, of size n^ϵ, is represented by a key denoted msk, and the set represented by msk is denoted $\mathsf{Set}(\mathsf{msk})$. Besides having a succinct key, the main additional property we need from an mPPRS is private S-way puncturability. This entails the following.[2]

[2]For convenience, our formulation of mPPRS is a slight modification of the description of [SWZ24].

- Let idx be an element of $\mathsf{Set}(\mathsf{msk})$. There is an efficient function $\mathsf{Puncture}$ such that $\mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})$ produces S *punctured keys* denoted $\mathsf{sk}_0, \ldots, \mathsf{sk}_{S-1}$.
- There is a function $\mathsf{Subsets}$ such that for $s \in [\![S]\!]$, $\mathsf{Subsets}(s, \mathsf{sk}_s)$ outputs a list of sets denoted List_s, and moreover, there exist indices $y_0, \ldots, y_{S-1}$ (which can be efficiently computed from msk and idx), such that
$$\triangle_{s \in [\![S]\!]} \mathsf{List}_s[y_s] = \mathsf{Set}(\mathsf{msk}) \triangle \{\mathsf{idx}\} ,$$
where $\triangle$ denotes the set symmetric difference operation, i.e., for sets A, B we have $A \triangle B = (A \cup B) \setminus (A \cap B)$.
- The main privacy requirement is that for any index idx and any $s \in [\![S]\!]$, the punctured key sk_s reveals no information about idx.

From mPPRS to multi-server PIR. Given an (n, S, ϵ)-mPPRS scheme, we can construct an $(S+1)$-server PIR scheme for $Q = \Omega(n^{1-\epsilon})$ queries and with constant error probability as follows.

- *Preprocessing.* A separate preprocessing server samples $O(n^{1-\epsilon})$ keys using the mPPRS scheme. For each msk sampled, it computes $\mathsf{Parity}(\mathsf{Set}(\mathsf{msk})) := \bigoplus_{j \in \mathsf{Set}(\mathsf{msk})} \mathsf{DB}[j]$ (we always treat database entries as binary strings), and sends msk along with the parity $\mathsf{Parity}(\mathsf{Set}(\mathsf{msk}))$ to the client. The client stores a hint table containing all the keys and their corresponding parities.
- *Query for* idx. To query $\mathsf{DB}[\mathsf{idx}]$, the client and server engage in the following protocol:
  1. The client finds some msk in its hint table such that $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$. If not found, the client samples a random msk subject to containing idx. It then computes the punctured keys $\mathsf{sk}_0, \ldots, \mathsf{sk}_{S-1} \leftarrow \mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})$, and sends sk_s to server s.
  2. Each server s calls $\mathsf{Subsets}(s, \mathsf{sk}_s)$ and obtains a list of sets List_s. Then, it responds to the client with $\mathsf{Parity}(\mathsf{List}_s[y])$ for every $y \in [\![|\mathsf{List}_s|]\!]$.
  3. If a matching msk was found earlier, the client computes $y_0, \ldots, y_{S-1}$ from msk and idx. Let p_s be the y_s-th parity returned by server s, corresponding to the set $\mathsf{List}_s[y_s]$. The client outputs the answer $\mathsf{Parity}(\mathsf{Set}(\mathsf{msk})) \oplus \left(\bigoplus_{s \in [\![S]\!]} p_s\right)$ where $\mathsf{Parity}(\mathsf{Set}(\mathsf{msk}))$ is stored in the client's hint table along with msk, and $\oplus$ denotes the xor operation.
  4. If a matching msk was found earlier, the client replaces the msk in the hint table with a freshly sampled key subject to containing idx, and sets the corresponding parity bit to 0. This refreshed key is considered *broken*, and if consumed later will lead to an incorrect answer.

Using standard techniques [ZPSZ24, ISW24], we can then upgrade the above scheme to support an unbounded number of queries, and boost the correctness probability to all but negligible.

The performance of the resulting PIR scheme relates directly to the underlying mPPRS scheme. Concretely, the client storage depends on the size of msk, the client computation depends on given idx, how fast a msk can be found in the hint table satisfying $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$. The server computation depends on the efficiency of the $\mathsf{Subsets}$ function, and the bandwidth depends on both the size sk and the number of sets output by $\mathsf{Subsets}$.

2.2 Our Efficient mPPRS Construction

Our core contribution is to devise a new, efficient mPPRS scheme. To illustrate our ideas, we first describe the scheme for the special case where $S = 3$ and $\epsilon = 1/2$, and $n^{\epsilon/S}$ is a power of 2. In our subsequent sections, we will present a version with generalized parameters.

An initial strawman. We start off by presenting an initial attempt at an mPPRS construction which does not support efficient testing of whether some $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$ given msk. In this strawman construction, msk is a $3 \times n^{1/6}$ matrix where each entry of the matrix is an element of $[\![n]\!]$ sampled uniformly at random. We use the notation $\mathsf{msk}[s][:]$, to denote the s-th row of msk. Given msk, we can expand it to the following set of indices where $\oplus$ denotes bit-wise xor:

$$\mathsf{Set}(\mathsf{msk}) = \left\{ \mathsf{msk}[0][x_0] \oplus \mathsf{msk}[1][x_1] \oplus \mathsf{msk}[2][x_2] : x_0, x_1, x_2 \in [\![n^{1/6}]\!] \right\}$$

It is not hard to show that for a randomly sampled msk and any $\mathsf{idx} \in [\![n]\!]$, the probability that $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$ is $\Theta(n^{-1/2})$.

Suppose $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, i.e., there exist $x_0^*, x_1^*, x_2^* \in [\![n^{1/6}]\!]$ such that $\mathsf{msk}[0][x_0^*] \oplus \mathsf{msk}[1][x_1^*] \oplus \mathsf{msk}[2][x_2^*] = \mathsf{idx}$. We can puncture msk at idx as follows, resulting in $\mathsf{sk}_0, \mathsf{sk}_1$, and sk_2 (see also Figure 1):

$$\mathsf{sk}_0 = \begin{pmatrix} \mathsf{msk}[0][:]/(x_0^* \leftarrow \$) \\ \mathsf{msk}[1][:] \\ \mathsf{msk}[2][:] \end{pmatrix}, \quad \mathsf{sk}_1 = \begin{pmatrix} \mathsf{msk}[0][x_0^*] \\ \mathsf{msk}[1][:]/(x_1^* \leftarrow \$) \\ \mathsf{msk}[2][:] \end{pmatrix}, \quad \mathsf{sk}_2 = \begin{pmatrix} \mathsf{msk}[0][x_0^*] \\ \mathsf{msk}[1][x_1^*] \\ \mathsf{msk}[2][:]/(x_2^* \leftarrow \$) \end{pmatrix}$$

where the notation $\mathsf{msk}[s][:]/(x_s^* \leftarrow \$)$ means replace the x_s^*-th element in the s-th row $\mathsf{msk}[s][:]$ with a randomly sampled element from $[\![n]\!]$. Observe that in sk_1, row 0 has only one element from $[\![n]\!]$, and in sk_2, both rows 0 and 1 have only one element from $[\![n]\!]$. It is also easy to see that each of $\mathsf{sk}_0, \mathsf{sk}_1$, and sk_2 individually leaks almost nothing about idx.

Given a punctured key sk, we use the notation $\mathsf{sk}\langle * \backslash \{x_0\}, *, * \rangle$ to mean remove the x_0-th element from row 0 (thus reducing the length of row 0 by one). The notations $\mathsf{sk}\langle *, * \backslash \{x_1\}, * \rangle$ and $\mathsf{sk}\langle *, *, * \backslash \{x_2\} \rangle$ can be similarly defined. For a matrix sk where not all rows may have the same length, we naturally extend the Set notation as follows where $\mathsf{rlen}_i := |\mathsf{sk}[i][:]|$.

$$\mathsf{Set}(\mathsf{sk}) = \left\{ \mathsf{sk}[0][x_0] \oplus \mathsf{sk}[1][x_1] \oplus \mathsf{sk}[2][x_2] : x_0 \in [\![\mathsf{rlen}_0]\!], x_1 \in [\![\mathsf{rlen}_1]\!], x_2 \in [\![\mathsf{rlen}_2]\!] \right\}$$

Now, the $\mathsf{Subsets}(s, \mathsf{sk}_s)$ function returns the following lists $\{\mathsf{List}_s\}_{s \in [\![S]\!]}$:

$$\begin{aligned}
\mathsf{List}_0 &:= \left\{ \mathsf{Set}\big(\mathsf{sk}_0\langle * \setminus \{x_0\}, \quad *, \quad\quad *\rangle\big) : x_0 \in [\![n^{1/6}]\!] \right\} \\
\mathsf{List}_1 &:= \left\{ \mathsf{Set}\big(\mathsf{sk}_1\langle \quad *, \quad * \setminus \{x_1\}, \quad *\rangle\big) : x_1 \in [\![n^{1/6}]\!] \right\} \\
\mathsf{List}_2 &:= \left\{ \mathsf{Set}\big(\mathsf{sk}_2\langle \quad *, \quad\quad *, \quad * \setminus \{x_2\}\rangle\big) : x_2 \in [\![n^{1/6}]\!] \right\}
\end{aligned}$$

As illustrated in Figure 1, it is not hard to see that

$$\mathsf{List}_0[x_0^*] \triangle \mathsf{List}_1[x_1^*] \triangle \mathsf{List}_2[x_2^*] = \mathsf{Set}(\mathsf{msk}) \triangle \{\mathsf{idx}\}$$

Efficiency analysis. Suppose we use the above strawman mPPRS to build an 4-server as described in Section 2.1. We now analyze the efficiency of this PIR scheme — for now, we will focus on the scheme for a bounded $Q = O(n^{1/2})$ number of queries and with constant error probability, since the standard upgrades [ZPSZ24, ISW24] for supporting an unbounded number of queries and boosting the correctness probability incurs only a super-logarithmic factor blowup. We will highlight in blue the undesirable parts of the costs which we want to improve:

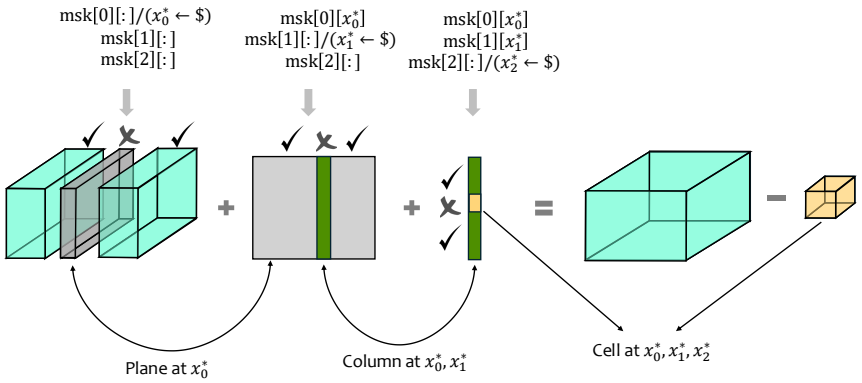


Figure 1: Puncturing msk at (x_0^*, x_1^*, x_2^*)

- *Preprocessing cost.* The preprocessing server needs to sample $n^{1-\epsilon} = n^{1/2}$ keys, and send the keys and their respective parities to the client. This requires $O(n^{1/2} \cdot n^{1/6}) = O(n^{2/3})$ communication, as well as $O(n^{1/2} \cdot n^{1/2}) = O(n)$ server computation. Amortized to $Q = O(n^{1/2})$ queries, the offline bandwidth is $O(n^{1/6})$ and the offline server computation is $O(n^{1/2})$ per query.
- *Query costs.*
  - *Online bandwidth.* During each query, the client sends one punctured key of size $O(n^{1/6})$ to each server. Each server responds with the parities of a list of $n^{1/6}$ sets. Thus, the online bandwidth to each server is $O(n^{1/6})$.
  - *Online server computation.* If the server computes all the responses naïvely, its computation would be rather expensive. However, with some algorithmic tricks, we can bound server 0's online computation by $O(n^{1/2})$, server 1's online computation by $O(n^{1/3})$, and server 2's online computation by $O(n^{1/6})$. Specifically, server 0 could first compute the parities for $\mathsf{Set}(\mathsf{sk}_0\langle x_0, *, *\rangle)$ for every $x_0 \in [\![n^{1/6}]\!]$, where $\mathsf{sk}_0\langle x_0, *, *\rangle$ is obtained by copying rows 1 and 2 of sk_0 and extracting only the x_0-th element from row 0. With these intermediate parities, the server can compute all $n^{1/6}$ parities in the response in $O(n^{1/2})$ time. A similar argument holds for server 1 and server 2.
  - *Online client computation.* The client needs to find a key in its hint table that matches the desired idx. Unfortunately, our strawman mPPRS scheme does not provide an efficient membership test procedure. To test if $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$ given some msk, the naïve approach is to expand the entire $\mathsf{Set}(\mathsf{msk})$ which takes $O(n^{1/2})$ time. Therefore, the client's total online computation is $O(n)$.
- *Client space.* The client needs to store $O(n^{1/2})$ keys each of size $O(n^{1/6})$, along with one parity bit per key. In total, we need $O(n^{2/3})$ client space.

Improving the efficiency. As shown above, there are two major inefficiencies with the strawman scheme.

1. First, during the query phase, server 0's online work is significantly larger than the other two servers. We propose an optimization that leverages our mPPRS scheme in a *non-blackbox manner* to get rid of server 0 by pushing the work to the preprocessing phase. *The resulting scheme requires only* 3 *servers rather than* 4 *servers.* Specifically, the preprocessing server can send back all $n^{1/6}$ parities corresponding to the sets $\{\mathsf{Set}(\mathsf{msk}\langle x_0, *, *\rangle)\}_{x_0}$ for every msk

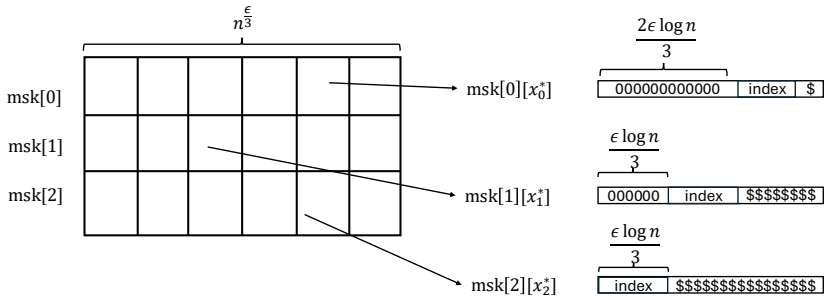


Figure 2: Structure of each entry in msk, when $S = 3$

sampled. This way, during any query, the client no longer needs to contact server 0 since it already knows

$$\mathsf{Parity}\left(\mathsf{Set}(\mathsf{msk})\right) \oplus \underbrace{\mathsf{Parity}\left(\mathsf{Set}(\mathsf{msk}\langle * \backslash \{x_0^*\}, *, \ldots, * \rangle)\right)}_{\text{returned by server 0 before}} = \underbrace{\mathsf{Parity}(\mathsf{Set}(\mathsf{msk}\langle \{x_0^*\}, *, \ldots, * \rangle))}_{\text{now obtained during preprocessing}}$$

This modification does not asymptotically increase the preprocessing costs or client space.

2. The second inefficiency is more challenging. Because our mPPRS scheme does not support an efficient membership test, the client's online computation is linear in n. To overcome this efficiency bottleneck, we need a non-trivial change to our mPPRS scheme to support efficient membership testing. We describe our ideas below.

Making membership testing efficient. In order to solve the aforementioned issue, we modify our initial construction. Instead of sampling the entries of the key matrix uniformly at random from $[\![n]\!]$, we make the following modification (also shown in Figure 2):

- *Row 2:* for each $x_2 \in [\![n^{1/6}]\!]$, the leading $\frac{1}{6} \cdot \log_2 n$ bits of $\mathsf{msk}[2][x_2]$ are fixed to the binary representation of x_2 (called "index bits" in Figure 2), and the remaining bits of $\mathsf{msk}[2][x_2]$ are random;
- *Row 1:* for each $x_1 \in [\![n^{1/6}]\!]$, the leading $\frac{1}{6} \cdot \log_2 n$ bits of $\mathsf{msk}[1][x_1]$ are fixed to 0, the next $\frac{1}{6} \cdot \log_2 n$ bits of $\mathsf{msk}[1][x_1]$ are fixed to the binary representation of x_1 (called "index bits" in Figure 2), and the remaining bits of $\mathsf{msk}[1][x_1]$ are random;
- *Row 0:* for each $x_0 \in [\![n^{1/6}]\!]$, the leading $\frac{1}{3} \cdot \log_2 n$ bits of $\mathsf{msk}[0][x_0]$ are fixed to 0, the next $\frac{1}{6} \cdot \log_2 n$ bits of $\mathsf{msk}[0][x_0]$ are fixed to the binary representation of x_0 (called "index bits" in Figure 2), and the remaining bits of $\mathsf{msk}[0][x_0]$ are random.

With this modification, it is not hard to see that for a randomly sampled msk, the probability that any fixed $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$ is exactly $n^{-1/2}$. Puncturing is performed in the same way as before, except that for the entry we resample in each punctured key, we only resample the random bits — the fixed bits of the entry remain untouched. Clearly, each punctured key still satisfies privacy.

The advantage of this modification is that it now supports membership test in $O(S)$ time — for the special case $S = 3$, the membership test takes $O(1)$ time. To determine whether $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, we can use the leading $\frac{1}{6} \log n$ bits of idx to determine x_2. Then, the next $\frac{1}{6} \log n$ bits of $\mathsf{idx} \oplus \mathsf{key}[2][x_2]$ determines x_1, and finally the next $\frac{1}{6} \log n$ bits of $\mathsf{idx} \oplus \mathsf{key}[1][x_1] \oplus \mathsf{key}[2][x_2]$ determines x_0. Thus, membership testing can be done in $O(1)$ time.

Putting it together and generalized parameters. Putting everything together, we can get a 3-server PIR scheme with $O(n^{1/6})$ bandwidth, $O(n^{1/3})$ online server computation, $O(n^{1/2})$ offline server computation, $O(n^{1/2})$ online client computation, $O(n^{1/6})$ offline client computation, while consuming $O(n^{2/3})$ client space. The resulting scheme supports only $Q = O(n^{1/2})$ queries and has constant error probability. However, as mentioned, we can upgrade it to a scheme that supports an unbounded number of queries and with all but negligible failure probability, with only a super-logarithmic factor blowup in all costs [ZPSZ24, ISW24].

More generally, to get an S-server PIR, we can employ an (n, S, ϵ)-mPPRS scheme. Henceforth let $\ell := \frac{\epsilon}{S} \cdot \log_2 n$, and we shall assume that ℓ is a positive integer for convenience, which implies that $S = O(\log n)$. The msk is a $S \times n^{\epsilon/S}$ matrix, and entries in the matrix are sampled from the following distributions:

- *Row* $S-1$*:* for each entry $\mathsf{msk}[S-1][x_{S-1}]$, the leading ℓ bits are fixed to the binary representation of x_{S-1}, and the remaining bits are random;
- *Row* $S-2$*:* for each entry $\mathsf{msk}[S-2][x_{S-2}]$, the leading ℓ bits are fixed to 0, the next ℓ bits fixed to the binary representation of x_{S-2}, and the remaining bits are random;
- ...
- *Row* 0: for each entry $\mathsf{msk}[0][x_0]$, the first $(S-1)\cdot\ell$ bits are fixed to 0, the next ℓ bits fixed to the binary representation of x_0, and the remaining $(1-\epsilon)\log_2 n$ bits are random.

To puncture some msk at $\mathsf{idx} \in [\![n]\!]$, suppose $\mathsf{idx} = \bigoplus_{i\in[\![S]\!]} \mathsf{msk}[i][x_i^*]$. The s-th punctured key is composed as follows: for each $i \in \{0, \ldots s-1\}$, row i is fixed to a single value $\mathsf{msk}[i][x_i^*]$; the s-th row is copied from msk but resampling the entry $\mathsf{msk}[s][x_s^*]$ from the appropriate distribution; and all remaining rows are directly copied from msk.

The resulting S-server PIR scheme (for bounded queries and with constant error probability) enjoys the following performance:

- *Preprocessing cost.* In the preprocessing phase, the server computation is $O(n)$, and communication is $O(Sn^{1-\epsilon+\epsilon/S})$. Amortized to a window of $Q = O(n^{1-\epsilon})$ queries, the offline server computation is $O(n^{\epsilon})$ and the offline bandwidth is $O(Sn^{\epsilon/S})$ per query.
- *Query costs.* The online bandwidth is $O(S\cdot n^{\epsilon/S})$. The online per-server computation is bounded by $O(n^{\epsilon-\epsilon/S})$. The online client computation is $O(S \cdot n^{1-\epsilon} + S^2 \cdot n^{\epsilon/S})$.
- *Client space.* The client space is bounded by $O(S \cdot n^{1-\epsilon+\epsilon/S})$.

Again, we can upgrade it to support unbounded queries and have all but negligible correctness probability at only super-logarithmic blowup in performance.

2.3 Comparison with [ISW24, SWZ24]

Both [ISW24, SWZ24] and our work use a matrix to represent a hint, that is, a set of indices. We summarize their constructions below, where n is assumed to be a power of 2^6 for simplicity.

- [ISW24] only considered 2-server PIR. Each hint is a matrix with 3 rows. For each row, every number in $[n^{1/3}]$ is included with probability $n^{-1/6}$, resulting in an expected row length of $n^{1/6}$. This is similar to sampling exactly $n^{1/6}$ elements uniformly at random from $[n^{1/3}]$. [3] The entire set is defined as

$$\mathsf{Set}(\mathsf{msk}) = \left\{ \mathsf{msk}[0][x_0] + \mathsf{msk}[1][x_1]\cdot n^{1/3} + \mathsf{msk}[2][x_2]\cdot n^{2/3} : x_0 \in [\![\mathsf{rlen}_0]\!], x_1 \in [\![\mathsf{rlen}_1]\!], x_2 \in [\![\mathsf{rlen}_2]\!] \right\}.$$

[3] However, due to technical issues, the technique of [ISW24] is not directly applicable to this modification.

- [SWZ24] considered $2t$-server PIR for all $t \geq 2$. For ease of comparison, we take $t = 3$, in which case each hint is a matrix with 3 rows and $n^{1/6}$ columns. The entries in the matrix are sampled uniformly at random from $[n^{1/2}]$. The entire set is defined as

$$\mathsf{Set}(\mathsf{msk}) = \Big\{ x_0 \cdot n^{1/2} + x_1 \cdot n^{2/3} + x_2 \cdot n^{5/6} + (\mathsf{msk}[0][x_0] \oplus \mathsf{msk}[1][x_1] \oplus \mathsf{msk}[2][x_2]) : \\ x_0 \in [\![\mathsf{rlen}_0]\!], x_1 \in [\![\mathsf{rlen}_1]\!], x_2 \in [\![\mathsf{rlen}_2]\!] \Big\}.$$

Until now the similarity seems to merely arise from the use of matrices. We point out that by using random bits and index bits, both approaches can be expressed in a unified way, as in Figures 3 and 4. With this representation, both approaches can be rewriten as

$$\mathsf{Set}(\mathsf{msk}) = \Big\{ \mathsf{msk}[0][x_0] \oplus \mathsf{msk}[1][x_1] \oplus \mathsf{msk}[2][x_2] : x_0 \in [\![\mathsf{rlen}_0]\!], x_1 \in [\![\mathsf{rlen}_1]\!], x_2 \in [\![\mathsf{rlen}_2]\!] \Big\}.$$

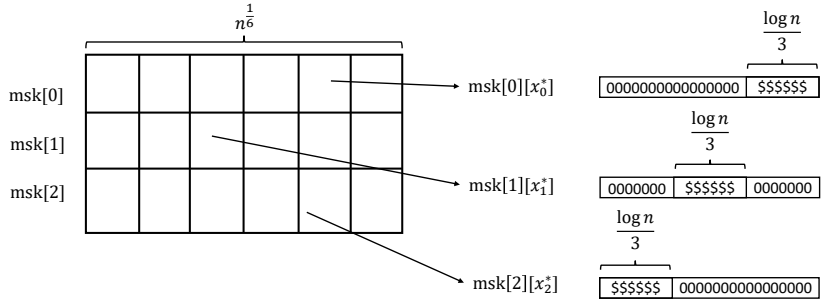


Figure 3: Structure of each entry in [ISW24]'s hint, approximately

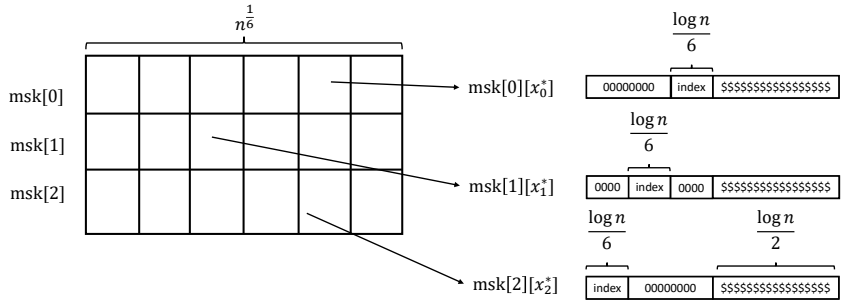


Figure 4: Structure of each entry in [SWZ24]'s hint, for $t = 3$

After deciding on the hint structure, the next step is to construct an appropriate puncturing algorithm. Our puncturing algorithm is much simpler than those in [ISW24, SWZ24]. In [ISW24], the authors used the inclusion-exclusion principle; In [SWZ24], while the algorithm itself was conceptually simple, it required at least 4 non-colluding servers to be non-trivial. Both of these schemes have efficiency worse than our scheme.

Our simplicity stems from the fact that the index bits in i-th row of our hint matrix are masked by the random bits in the $(i+1)$-th row. Therefore, when puncturing msk at $\mathsf{idx} = \bigoplus_{i=0}^{S-1} \mathsf{msk}[i][x_i^*]$, we can leak $\mathsf{msk}[i][x_i^*]$ without compromising privacy as long as $\mathsf{msk}[i+1][x_{i+1}^*]$ is rerandomized. In contrast, in [ISW24, SWZ24], leaking $\mathsf{msk}[i][x_i^*]$ always compromises privacy, so they had to compensate by using a more sophisticated puncturing algorithm or more servers and computation.

We finally remark that in our scheme, the msk has at most 2 rows for a 2-server PIR scheme, while [ISW24] managed to use 3 rows, which has the potential for lower client space. It remains an open question to find the best combination of hint structures and puncturing algorithms.

3 Preliminaries

3.1 Notation

Let $\mathbb{N}^+ = \{0, 1, \ldots, \}$. For every $n \in \mathbb{N}^+$, we use $[\![n]\!]$ to represent the set $\{0, \ldots, n-1\}$. For sets A and B we denote using $A \triangle B$ the symmetric difference of the two sets, i.e., $A \triangle B = (A \cup B) \setminus (A \cap B)$. We denote using $\oplus$ the bitwise xor operation. We use $\leftarrow$ to denote assignment. We let $x \xleftarrow{\$} D$ denote sampling x according to the distribution D. If D is a set, we overload notation and let $x \xleftarrow{\$} D$ denote uniformly sampling from the elements of D. For a bit-string (or a list) v, we use $|v|$ to denote the number of bits (or elements) in v.

3.2 Multi-Server Private Information Retrieval

S-server PIR without preprocessing. An S-server PIR scheme with preprocessing consists of the following sub-protocols between a client and S non-colluding servers numbered $0, 1, \ldots, S-1$.

- **One-time preprocessing.** Each server's input is a database $\mathsf{DB} \in \{0,1\}^n$. The client has no input. The client and the servers engage in a protocol at the end of which the (stateful) client stores some local state henceforth called a *hint table*. In this paper, we assume that the servers do not need extra space beyond storing the original DB.
- **Queries.** After the one-time preprocessing, the client can issue an unbounded number of queries. During each query, the client receives an index $\mathsf{idx} \in [\![n]\!]$ as input, and the server receives no additional input. At the end of the protocol, the client outputs an answer denoted ans.

Non-adaptive correctness. We require that for any polynomial functions $S(\cdot)$ and $T(\cdot)$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for any n, any $\mathsf{DB} \in \{0,1\}^n$, any sequence of $T(n)$ queries denoted $\mathsf{idx}_0, \ldots, \mathsf{idx}_{T-1} \in [\![n]\!]$, with all but $\mathsf{negl}(n)$ probability, the following holds: if we execute the honest preprocessing between a client and $S(n)$ servers, followed by the honest query protocol for $\mathsf{idx}_0, \ldots, \mathsf{idx}_{T-1}$, it must be that for any $t \in [\![T]\!]$, $\mathsf{ans}_t = \mathsf{DB}[\mathsf{idx}_t]$ where ans_t denotes the client's answer for the query idx_t.

Adaptive correctness. We say that an S-server PIR scheme satisfies adaptive correctness in the presence of at most $S' < S$ colluding servers, iff for any polynomial function $\mathsf{poly}(\cdot)$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any n, no (even computationally unbounded) adversary that corrupts up to S' servers can win the following game except with $\mathsf{negl}(n)$ probability:

- The adversary outputs a database $\mathsf{DB} \in \{0,1\}^n$, and the experiment initializes an honest client as well as S servers. The client and the servers run the honest preprocessing protocol where all

servers receive DB as input, and the corrupt servers' views (including their random coins and the messages they receive) are given to the adversary.

- For each iteration $t = 0, 1, 2, \ldots, \mathsf{poly}(n)$: the adversary adaptively outputs some index $\mathsf{idx}_t \in [\![n]\!]$. The client and the servers invoke an honest query protocol where the client uses idx_t as input. The corrupt servers' views are given to the adversary. Let ans_t be the answer output by the client in the t-th iteration.
- The adversary is said to win the game if there exists t such that $\mathsf{ans}_t \neq \mathsf{DB}[\mathsf{idx}_t]$.

Privacy. We say that an S-server PIR scheme is *perfectly private* if for any $s \in [\![S]\!]$, any $n \in \mathbb{N}^+$, any $\mathsf{DB} \in \{0,1\}^n$, for any (even computationally unbounded) adversary $\mathcal{A}$ controlling server s, there exists a simulator Sim such that for any sequence of indices $\mathsf{idx}_0, \ldots, \mathsf{idx}_{T-1}$, the following distributions are identical.

- **Real execution.** Instantiate an honest client and $S-1$ honest servers. Execute the preprocessing protocol where all servers' inputs are DB, and then for $t \in [\![T]\!]$, execute the query protocol where the client's input is idx_t. Output the adversary's view including DB, all of its random coins, as well as all protocol messages it receives.
- **Ideal execution.** Output $\mathsf{Sim}(T)$.

Note that in the above definition, the simulator Sim may depend on n, DB, s, the adversary's algorithm $\mathcal{A}$, and moreover, it is given the number of queries T. Importantly, the simulator does not know the queries $\mathsf{idx}_0, \ldots, \mathsf{idx}_{T-1}$.

4 Multi-way Privately Puncturable Random Set

4.1 Formal Definitions

Syntax. An (n, S, ϵ)-Multi-way Privately Puncturable Random Set (mPPRS) is defined by the following tuple of algorithms $(\mathsf{Gen}, \mathsf{Set}, \mathsf{Test}, \mathsf{Puncture}, \mathsf{Subsets}, \mathsf{GetIndex})$.

- $\mathsf{msk} \leftarrow \mathsf{Gen}(n, S, \epsilon)$: a randomized algorithm that samples and outputs a key msk.
- $\mathcal{I} \leftarrow \mathsf{Set}(\mathsf{msk})$: a deterministic algorithm that takes as input a key msk, and outputs a set of indices $\mathcal{I} \subseteq [\![n]\!]$.
- $b \leftarrow \mathsf{Test}(\mathsf{msk}, \mathsf{idx})$: a deterministic algorithm that takes as input a key msk and an element $\mathsf{idx} \in [\![n]\!]$, and outputs a bit $b \in \{0,1\}$ indicating whether $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$.
- $\mathsf{sk}_0, \ldots, \mathsf{sk}_{S-1} \xleftarrow{\$} \mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})$: a deterministic algorithm that takes as input a key msk, an index $\mathsf{idx} \in [\![n]\!]$ such that $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, and outputs S punctured keys $\mathsf{sk}_0, \ldots, \mathsf{sk}_{S-1}$.
- $\mathsf{List}_s \leftarrow \mathsf{Subsets}(s, \mathsf{sk}_s)$: a deterministic algorithm that takes as input $s \in [\![S]\!]$ and a punctured key and outputs a list of sets denoted List_s.
- $y_0, \ldots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$: a deterministic algorithm that takes as input a key msk and an element $\mathsf{idx} \in [\![n]\!]$, and outputs S indices $y_0, \ldots, y_{S-1}$.

Correctness. An (n, S, ϵ)-mPPRS is correct if it satisfies the following conditions.

- **Correctness of membership test.** We require that for all natural number n and S, for any $0 < \epsilon < 1$, any $\mathsf{idx} \in [\![n]\!]$, the following holds with probability 1: let $\mathsf{msk} \leftarrow \mathsf{Gen}(n, S, \epsilon)$, $b \leftarrow \mathsf{Test}(\mathsf{msk}, \mathsf{idx})$, then it must be that $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk}) \Leftrightarrow b = 1$.

- **Correctness of puncturing.** We require that for all natural numbers n, S, any $0 < \epsilon < 1, \mathsf{idx} \in [\![n]\!]$, the following holds with probability 1: sample $\mathsf{msk} \leftarrow \mathsf{Gen}(n, S, \epsilon)$ subject to $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, let $\mathsf{sk}_0, \ldots, \mathsf{sk}_{S-1} \leftarrow \mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})$, let $\mathsf{List}_s \leftarrow \mathsf{Subsets}(s, \mathsf{sk}_s)$ for $s \in [\![S]\!]$, let $y_0, \ldots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$, then, it must be that $\mathsf{Set}(\mathsf{msk}) \triangle \{\mathsf{idx}\} = \triangle_{s \in [\![S]\!]} \mathsf{List}_s[y_s]$.

Privacy. An (n, S, ϵ)-mPPRS is perfectly private if it satisfies the following condition. There exists a simulator Sim such that for all natural numbers n, S, any $0 < \epsilon < 1$, $\mathsf{idx} \in [\![n]\!]$, $s \in [\![S]\!]$, the following distributions are identical:

- Sample $\mathsf{msk} \leftarrow \mathsf{Gen}(n, S, \epsilon)$ subject to $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, let $\mathsf{sk}_0, \ldots, \mathsf{sk}_{S-1} \leftarrow \mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})$, and output sk_s.
- Output $\mathsf{Sim}(n, S, \epsilon, s)$.

4.2 mPPRS Construction

Notation. Let $B_i = n^{1-\epsilon} \cdot \left(n^{\epsilon/S}\right)^i$ for $i \in [\![S]\!]$. For a (punctured) key sk, let $\mathsf{sk}[i][:]$ denote the i-th row of sk, and let $\mathsf{sk}[i][:]/(x_i^* \leftarrow \$)$ denote the i-th row of sk with the x_i^*-th element rerandomized, i.e., the least significant $\log_2 B_i$ bits of the x_i^*-th element are sampled uniformly at random. For a (punctured) key sk, for sets $X_0 \subseteq [\![|\mathsf{sk}[0][:]|]\!], \cdots, X_{S-1} \subseteq [\![|\mathsf{sk}[S-1][:]|]\!]$, we use $\mathsf{sk}\langle X_0, \cdots, X_{S-1}\rangle$ to denote the key obtained by keeping only the elements at $\mathsf{sk}[i][x]$ where $i \in [\![S]\!]$ and $x \in X_i$, and removing the rest. We use $*$ to denote $[\![|\mathsf{sk}[i][:]|]\!]$ when i is clear from the context. For example, $\mathsf{sk}\langle *, *, \ldots, *, *\backslash\{0\}\rangle$ means remove the 0-th element from the last row of sk.

Construction. In Figure 5, we construct an (n, S, ϵ)-mPPRS where $2 \le S \le \log n$ and $0 < \epsilon < 1$. We assume n is a power of 2, and $(\epsilon/S)\log_2 n$ is an integer. Later in Section 9, we discuss how to relax the parameter constraints that n must be a power of 2, and $(\epsilon/S)\log_2 n$ must be integral.

Intuition and alternative view. Let $\ell = (\epsilon/S) \cdot \log_2 n$. In $\mathsf{msk}[i][x]$, the leading $(S - i - 1) \cdot \ell$ bits are 0, the next ℓ bits are the binary representation of x, the remaining bits are random (see also Figure 2 for the special case when $S = 3$). In the algorithms Test and $\mathsf{GetIndex}$, we are using the leading ℓ bits of idx to determine x_{S-1}^*, the next ℓ bits of $\mathsf{idx} \oplus \mathsf{msk}[S-1][x_{S-1}^*]$ to determine x_{S-2}^*, the next ℓ bits of $\mathsf{idx} \oplus \mathsf{msk}[S-1][x_{S-1}^*] \oplus \mathsf{msk}[S-2][x_{S-2}^*]$ to determine x_{S-3}^*, and so on. Using this view, it is easy to see the following facts:

Fact 4.1. *The following facts hold for our* mPPRS *scheme in Figure 5.*

- *In the algorithms* Test *and* GetIndex*, in every iteration, the x_i^* computed falls within the correct range $[\![n^{\epsilon/S}]\!]$.*
- *If* $\mathsf{Test}(\mathsf{msk}, \mathsf{idx})$ *returns* 1*, then the following holds:* $\mathsf{idx} = \bigoplus_{i \in [\![S]\!]} \mathsf{msk}[i][x_i^*]$*, where* $\{x_i^*\}_{i \in [\![S]\!]}$ *are the indices computed in all the iterations in the* $\mathsf{Test}(\mathsf{msk}, \mathsf{idx})$ *algorithm.*

Efficiency. We now analyze the efficiency of our mPPRS construction.

- Gen: runs in time $O(S \cdot n^{\epsilon/S})$.

Construction of mPPRS

$\underline{\mathsf{Gen}(n, S, \epsilon)}$:

- Sample msk as an $S \times n^{\epsilon/S}$ matrix; where for $i \in [\![S]\!]$, $x \in \{0, \ldots, n^{\epsilon/S} - 1\}$, $\mathsf{msk}[i][x]$ is sampled uniformly at random from $\{x \cdot B_i, x \cdot B_i + 1, \cdots, (x+1) \cdot B_i - 1\}$.
- Return msk.

$\underline{\mathsf{Set}(\mathsf{msk})}$: *// This algorithm can also take in a punctured key* sk *as input.*

- Let $\mathsf{rlen}_i := |\mathsf{msk}[i][:]|$, for $i \in [\![S]\!]$.
- Return $\left\{\bigoplus_{i=0}^{S-1} \mathsf{msk}[i][x_i] : x_i \in [\![\mathsf{rlen}_i]\!]\right\}$.

$\underline{\mathsf{Test}(\mathsf{msk}, \mathsf{idx})}$:

- For i from $S-1$ down to 0:
  - Let $x_i^* \leftarrow \lfloor \mathsf{idx}/B_i \rfloor$.
  - Let $\mathsf{idx} \leftarrow \mathsf{idx} \oplus \mathsf{msk}[i][x_i^*]$.
- If $\mathsf{idx} = 0$, return 1; otherwise, return 0.

$\underline{\mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})}$:

- For i from $S-1$ down to 0:
  - Let $x_i^* \leftarrow \lfloor \mathsf{idx}/B_i \rfloor$.
  - Let $\mathsf{idx} \leftarrow \mathsf{idx} \oplus \mathsf{msk}[i][x_i^*]$.
- Return $x_0^*, \cdots, x_{S-1}^*$.

$\underline{\mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})}$:

- Let $y_0, \cdots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$.
- Return $\mathsf{sk}_0, \cdots, \mathsf{sk}_{S-1}$ where for $s \in [\![S]\!]$,

$$\mathsf{sk}_s = \begin{pmatrix} \mathsf{msk}[0][y_0] \\ \vdots \\ \mathsf{msk}[s-1][y_{s-1}] \\ \mathsf{msk}[s][:]/(y_s \leftarrow \$) \\ \mathsf{msk}[s+1][:] \\ \vdots \\ \mathsf{msk}[S-1][:] \end{pmatrix}$$

$\underline{\mathsf{Subsets}(s, \mathsf{sk}_s)}$:

- Return $\left\{\mathsf{Set}\big(\mathsf{sk}_s \langle \underbrace{*, \cdots, *}_{s}, * \setminus \{x_s\}, \underbrace{*, \cdots, *}_{S-s-1} \rangle\big) : x_s \in [\![n^{\epsilon/S}]\!]\right\}$.

Figure 5: Construction of mPPRS

- Set: runs in time $O(n^\epsilon)$.
- Test: runs in time $O(S)$.

- $\mathsf{Puncture}$: runs in time $O(S^2 \cdot n^{\epsilon/S})$.
- $\mathsf{Subsets}$: using a naïve approach, the algorithm $\mathsf{Subsets}(s, \mathsf{sk}_s)$ runs in time $O(n^{\epsilon(S-s+1)/S})$ for outputting all $n^{\epsilon/S}$ sets, each of size $O(n^{\epsilon(S-s)/S})$. However, we point out that any two of those sets differ by only $n^{\epsilon(S-s-1)/S}$ elements. Later, we will exploit this fact in our PIR scheme for better efficiency — see Section 5.

We prove the correctness of our mPPRS construction in the following section, and the privacy of our mPPRS construction in Section 4.4.

4.3 Correctness Proof of Our mPPRS

Lemma 4.2. *The following facts hold for our* mPPRS *construction.*

1. *For any* $\mathsf{idx} \in [\![n]\!]$, $\Pr\left[\mathsf{msk} \xleftarrow{\$} \mathsf{Gen}(n, S, \epsilon) : \mathsf{idx} \in \mathsf{Set}(\mathsf{msk})\right] = n^{\epsilon-1}$.
2. *For any* $\mathsf{idx}, \mathsf{idx}' \in [\![n]\!]$ *such that* $\mathsf{idx} \neq \mathsf{idx}'$,

$$\Pr\left[\mathsf{msk} \xleftarrow{\$} \mathsf{Gen}(n, S, \epsilon) \ s.t. \ \mathsf{idx} \in \mathsf{Set}(\mathsf{msk}) : \mathsf{idx}' \in \mathsf{Set}(\mathsf{msk})\right] \leq n^{\epsilon-1}.$$

Proof. We begin by proving the first fact. Let $x^*_0, \ldots, x^*_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$. It is easy to see that $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$ iff the least significant $(1-\epsilon)\log_2 n$ bits of $\bigoplus_{i \in [\![S-1]\!]} \mathsf{msk}[i][x^*_i]$ are equal to the least significant $(1-\epsilon)\log_2 n$ bits of idx. The probability that this happens is $n^{\epsilon-1}$ — note that the $\mathsf{GetIndex}$ function does not look at the least significant $(1-\epsilon)\log_2 n$ bits of the entries in msk.

For the second fact, conditioned on $\mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx}) \neq \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx}')$ and $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, then the probability that $\mathsf{idx}' \in \mathsf{Set}(\mathsf{msk})$ is exactly $n^{\epsilon-1}$ due to a similar argument as above. Conditioned on $\mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx}) = \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx}')$ and $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, then the probability that $\mathsf{idx}' \in \mathsf{Set}(\mathsf{msk})$ is 0. ∎

Correctness of membership test is easy to see using the alternative view in Section 4.2.

We now show correctness of puncturing. Fix arbitrary $\mathsf{msk} \xleftarrow{\$} \mathsf{Gen}(n, S, \epsilon)$ and $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$. Let $\mathsf{sk}_0, \cdots, \mathsf{sk}_{S-1} \leftarrow \mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})$, $\mathsf{List}_s \leftarrow \mathsf{Subsets}(s, \mathsf{sk}_s)$, and $y_0, \cdots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$. Then we know that

$$\begin{aligned}\mathsf{List}_s[y_s] &= \mathsf{Set}\big(\mathsf{sk}_s\langle\underbrace{*, \cdots, *}_{s}, * \setminus \{y_s\}, \underbrace{*, \cdots, *}_{S-s-1}\rangle\big) \\ &= \mathsf{Set}\big(\mathsf{msk}\langle\{y_0\}, \cdots, \{y_{s-1}\}, * \setminus \{y_s\}, *, \cdots, *\rangle\big)\end{aligned}$$

Therefore, we have

$$\begin{aligned}&\triangle_{s \in [\![S]\!]} \mathsf{List}_s[y_s] \\ &= \triangle_{s \in [\![S]\!]} \mathsf{Set}\big(\mathsf{msk}\langle\{y_0\}, \cdots, \{y_{s-1}\}, * \setminus \{y_s\}, *, \cdots, *\rangle\big) \\ &= \mathsf{Set}(\mathsf{msk}) \setminus \mathsf{Set}(\mathsf{msk}\langle\{y_0\}, \cdots, \{y_{S-1}\}\rangle) \\ &= \mathsf{Set}(\mathsf{msk}) \setminus \{\mathsf{idx}\} \ .\end{aligned}$$

A graphical illustration of the above proof is provided in Figure 1 for the special case $S = 3$.

4.4 Privacy Proof for Our mPPRS

We next prove privacy assuming that n, S, and ϵ satisfy the aforementioned parameter constraints. Fix an arbitrary $\mathsf{idx} \in [\![n]\!]$, and an arbitrary $s \in [\![S]\!]$. We want to show that the following two distributions are identically distributed:

- Real: sample msk from $\mathsf{Gen}(n, S, \epsilon)$ subject to $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, return the s-th punctured key sk_s output by $\mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})$.
- Ideal: return the output of some simulator $\mathsf{Sim}(n, S, \epsilon, s)$ that is unaware of idx.

Block notation. Below, let $\ell = (\epsilon/S) \cdot \log_2 n$. Given a number $k \in [\![n]\!]$ with $\log_2 n$ bits, we divide the bits into S blocks, where all blocks have length ℓ except the last block which has length $\ell + (1-\epsilon)\log_2 n$:

- block 0 contains the most significant ℓ bits,
- block 1 contains the next ℓ bits,
- ...,
- block $S-2$ contains the next ℓ bits, and
- finally, block $S-1$ contains the remaining $\ell + (1-\epsilon)\log_2 n$ bits.

Henceforth, we use the notation $\mathsf{Block}^i(k)$ to denote the i-th block of bits in the binary represeantation of k.

We prove privacy through a sequence of equivalent rewrites of the distribution. At the end, we reach a distribution that can be sampled without knowing idx, and this is exactly our simulator construction.

Experiment Hyb_1. First, we rewrite Real distribution as the following distribution Hyb_1.

- For $i = S-1$ down to s,
  - sample row i like in Real;
  - let $x_i^* = \mathsf{Block}^{S-1-i}(\mathsf{idx})$;
  - let $\mathsf{idx} \leftarrow \mathsf{idx} \oplus \mathsf{sk}[i][x_i^*]$. // *blocks* $0, \ldots, S-1-i$ *of* idx *are now 0*
- For $i = s-1$ downto 0:
  - let $x_i^* = \mathsf{Block}^{S-1-i}(\mathsf{idx})$;
  - sample row i to contain a single number $\mathsf{sk}[i][0]$ where blocks $0, 1, \ldots, S-2-i$ are 0, the next block encodes x_i^*, and the remaining bits are random.
  - $\mathsf{idx} \leftarrow \mathsf{idx} \oplus \mathsf{sk}[i][0]$; // *blocks* $0, \ldots, S-1-i$ *of* idx *are now 0*
- Resample $\mathsf{sk}[s][x_s^*]$ at random from the same distribution as before.
- Output the resulting sk.

Experiment Hyb_2. Hyb_2 first samples row $S-1, S-2, \ldots, s$ just like in Hyb_1, then it samples the remaining rows (which all contain a single number) block by block, as follows:

- For $i = S-1$ down to s,
  - sample row i like in Real;
  - let $x_i^* = \mathsf{Block}^{S-1-i}(\mathsf{idx})$;
  - let $\mathsf{idx} \leftarrow \mathsf{idx} \oplus \mathsf{sk}[i][x_i^*]$.
- Let $x_i^* = 0$ for $i \in \{s-1, \ldots, 0\}$.
- For $i = s-1$ downto 0: sample the $(S-1-i)$-th blocks of all of $\mathsf{sk}[s-1][0], \ldots, \mathsf{sk}[i][0]$ such that

$$\mathsf{Block}^{S-1-i}\left(\bigoplus_{j \in \{S-1, S-2, \ldots, i\}} \mathsf{sk}[j][x_j^*]\right) = \mathsf{Block}^{S-1-i}(\mathsf{idx}) \tag{1}$$

- Resample $\mathsf{sk}[s][x_s^*]$ at random from the same distribution as before.
- Output the resulting sk where all the unsampled bits are set to 0 by default.

Clearly, Hyb_2 is identically distributed as Hyb_1.

Experiment Ideal. Ideal is otherwise the same as Hyb_2, except that when we sample the rows $s-1, s-2, \ldots, 0$ block by block, we no longer need to respect the constraint Equation (1), and we no longer resample the entry $\mathsf{sk}[s][x_s^*]$ at the end. It is easy to see that Ideal identically distributed as Hyb_2 because in Hyb_2, resampling the entry $\mathsf{sk}[s][x_s^*]$ essentially makes the constraint of Equation (1) go away.

- For $i = S-1$ down to s, sample row i like in Real;
- For $i = s-1$ downto 0: sample the $(S-1-i)$-th blocks of all of $\mathsf{sk}[s-1][0], \ldots, \mathsf{sk}[i][0]$ at random.
- Output the resulting sk where all the unsampled bits are set to 0 by default.

Observe that Ideal can be sampled without knowing idx, so it essentially gives a simulator construction.

5 Multi-Server Preprocessing PIR from mPPRS

In this section, we present a construction of an S-server PIR scheme from our mPPRS scheme.

5.1 Construction

Basic scheme with constant correctness probability, and bounded queries. In Figure 6. we present a multi-server PIR scheme with constant correctness probability, and supporting up to $Q = n^{1-\epsilon}/10$ queries after the preprocessing phase. We will upgrade the scheme to support an unbounded number of queries and with all but negligible correctness. In this section, we focus on asymptotic analysis so we first describe a scheme that is crafted for conceptual simplicity and ease of understanding. Then, in Section 6, we describe new optimizations that enable a logarithmic factor improvement in bandwidth and computation. Finally, in Section 8, we describe several practical optimizations including how to tighten the constants, leading to the scheme that we actually implemented.

S-server PIR with constant correctness probability, supporting $n^{1-\epsilon}/10$ queries

// Let $\mathsf{mPPRS} = (\mathsf{Gen}, \mathsf{Set}, \mathsf{Test}, \mathsf{GetIndex}, \mathsf{Puncture}, \mathsf{Subsets})$ *be the scheme in Figure 5.*

Preprocessing.

- The preprocessing server (i.e., server 0) repeats the following procedure for $4n^{1-\epsilon}$ times:
  - Sample $\mathsf{msk} \xleftarrow{\$} \mathsf{Gen}(n, S, \epsilon)$, send msk to the client.
  - For each $x \in [\![n^{\epsilon/S}]\!]$, let $v_x \leftarrow \bigoplus_{j \in \mathsf{Set}(\mathsf{msk}, \langle \{x\}, *, \cdots, * \rangle)} \mathsf{DB}[j]$; send v_x to the client.
- The client receives the information and stores a hint table of size $4n^{1-\epsilon}$ where each hint entry is of the form $\left(\mathsf{msk}, \{v_x\}_{x \in [\![n^{\epsilon/S}]\!]}\right)$.

Query for index $\mathsf{idx} \in [\![n]\!]$. If idx was queried before, the client simply returns the same answer as before; moreover, it runs the query protocol for an arbitrary idx' that has not been queried before. Otherwise, run the following query protocol.

1. The client randomly selects an entry $(\mathsf{msk}, \{v_x\}_x)$ from the hint table such that $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$. If no such hint is found, sample a fresh msk subject to $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, and let $v_x = 0$ for $x \in [\![n^{\epsilon/S}]\!]$. *// made efficient through the efficient membership test* Test
2. The client computes $\mathsf{sk}_0, \cdots, \mathsf{sk}_{S-1} \leftarrow \mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})$, and sends sk_s to server s for each $s \in \{1, \ldots, S-1\}$.
3. For each $s \in \{1, \ldots, S-1\}$, server s receives sk_s, computes $\mathsf{List}_s \leftarrow \mathsf{Subsets}(s, \mathsf{sk}_s)$, and for each $y \in [\![|\mathsf{List}_s|]\!]$, it returns $\mathsf{Parity}(\mathsf{List}_s[y]) = \bigoplus_{j \in \mathsf{List}_s[y]} \mathsf{DB}[j]$ to the client.
4. The client computes $y_0, \ldots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$, let p_s denote the y_s-th parity received from server s, and outputs the answer $v_{y_0} \oplus \bigoplus_{s=1}^{S-1} p_s$.
5. If a matching hint was found in Step 1, then replace the consumed entry in the hint table with a freshly generated $\mathsf{msk} \leftarrow \mathsf{Gen}(n, S, \epsilon)$ subject to $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, and $\{v_x = 0\}_{x \in [\![n^{\epsilon/S}]\!]}$.

// This hint entry is considered broken, and will lead to a correctness error if consumed later.

Figure 6: S-server preprocessing PIR

As mentioned in Section 2.1, given a generic (n, S, ϵ)-mPPRS scheme, we can construct an $(S+1)$-server PIR scheme for a bounded number of queries with constant correctness probability. However, as mentioned in Section 2.2, we can leverage our specific mPPRS in a non-blackbox manner to reduce the number of servers to only S. Here in Figure 6, we directly present the optimized scheme with S servers.

Upgrade. Using standard techniques [ZPSZ24, ISW24], we can upgrade the basic scheme in Figure 6 to support an unbounded number of queries, and achieving all but negligible correctness probability. The idea is as follows:

- We can rely on parallel repetition to boost the correctness probability. Specifically, to achieve $1 - \mathsf{negl}(n)$ correctness probability, we need to run $k = \omega(\log n)$ independent copies of the scheme. Note that the client knows whether it is consuming a broken hint, and whether there is a match in Step 1. So the client is aware whether each copy gives a correct answer. Therefore, as long as one out of the k copies returns a correct answer, the client can output a correct answer for the query.

- To support an unbounded number of queries, we can rerun the preprocessing phase every batch of $n^{1-\epsilon}/10$ queries. Further, the preprocessing for the next batch can be piggybacked over the current batch of $n^{1-\epsilon}/10$ queries.

Efficiency. We analyze the efficiency of the basic scheme in Figure 6. The upgrade only incurs a super-logarithmic blowup in all efficiency measures.

- *Preprocessing cost.* In the preprocessing phase, server computes and sends $4n^{1-\epsilon}$ hints, where each hint requires $O(n^\epsilon)$ computation and $O(Sn^{\epsilon/S})$ communication. Thus the server computation is $O(n)$, and communication is $O(Sn^{1-\epsilon+\epsilon/S})$. Amortized to a window of $Q = n^{1-\epsilon}/10$ queries, the offline server computation is $O(n^\epsilon)$ and the offline bandwidth is $O(Sn^{\epsilon/S})$ per query.
- *Query costs.* For each query, the client first takes $O(Sn^{1-\epsilon})$ computation to find a matching hint. Then, the client sends a punctured key to each server, and each server takes at most $O(n^{\epsilon-\epsilon/S})$ computation to compute the parities. The online bandwidth is $O(S \cdot n^{\epsilon/S})$. The online per-server computation is bounded by $O(n^{\epsilon-\epsilon/S})$. The online client computation is $O(S \cdot n^{1-\epsilon} + S^2 \cdot n^{\epsilon/S})$.
- *Client space.* The client space is bounded by $O(S \cdot n^{1-\epsilon+\epsilon/S})$, for storing $4n^{1-\epsilon}$ hints.

Summarizing the above, we have the following theorem:

Theorem 5.1. *Suppose that n is a power of 2 and $\frac{\epsilon}{S} \cdot \log_2 n$ is a positive integer. Let $\alpha(n)$ be any function of n such that $\alpha(n) = \omega(\log n)$. Then, there exists an information theoretic S-server PIR scheme with the following performance bounds. The one-time preprocessing incurs $O(n\alpha(n))$ server computation and $O(S \cdot n^{1-\epsilon+\epsilon/S}\alpha(n))$ communication. Afterwards, every query incurs $O(S \cdot n^{\epsilon/S}\alpha(n))$ online and offline bandwidth, $O(n^{\epsilon-\epsilon/S}\alpha(n))$ online server computation, $O(n^\epsilon\alpha(n))$ offline server computation, $O((S \cdot n^{1-\epsilon} + S^2 \cdot n^{\epsilon/S})\alpha(n))$ online client computation, and $O(S \cdot n^{\epsilon/S}\alpha(n))$ offline client computation. Further, the client space is bounded by $O(S \cdot n^{1-\epsilon+\epsilon/S}\alpha(n))$.*

In Section 9, we show how to generalize our PIR construction for general n (when n is not necessarily a power of 2, and $\frac{\epsilon}{S} \cdot \log_2 n$ is not necessarily a positive integer).

We prove the above Theorem 5.1 in Section 5.2 and Section 5.3.

5.2 Privacy Proof

We analyze privacy for a single copy of the scheme without the parallel repetition. Earlier works [ZPSZ24, ISW24] essentially proved the following technical lemma through a simple inductive argument.

Lemma 5.2 (Privacy technical lemma [ZPSZ24, ISW24])**.** *Given an msk sampled from an arbitrary distribution $\mathcal{D}$, suppose that $\mathsf{Set}(\mathsf{msk})$ is a deterministic function that defines a subset of $[\![n]\!]$. Consider the following randomized experiment where L and T are arbitrary natural numbers:*

- *Sample a table that contains L keys msk sampled independently from the distribution $\mathcal{D}$.*
- *For $t \in [\![T]\!]$:*
  - *The adversary $\mathcal{A}$ outputs $\mathsf{idx}_t \in [\![n]\!]$;*
  - *Let msk_t^* be a random entry from the table such that $\mathsf{idx}_t \in \mathsf{Set}(\mathsf{msk}_t^*)$. If such a key is found, replace the corresponding entry in the table with a freshly sampled msk subject to $\mathsf{idx}_t \in \mathsf{Set}(\mathsf{msk})$. Otherwise, define msk_t^* be a fresh key sampled from $\mathcal{D}$ subject to $\mathsf{idx}_t \in \mathsf{Set}(\mathsf{msk}_t^*)$.*

– *Send* msk_t^* *to* $\mathcal{A}$.

Then, the following holds:

1. *At the end of every query t, even when conditioned on $\mathcal{A}$'s view so far, the distribution of the table is identical to sampling L keys independently at random from $\mathcal{D}$.*
2. *The keys $\{\mathsf{msk}_t^*\}_{t\in[\![T]\!]}$ received by $\mathcal{A}$ are identically distributed as the following: for each $t \in [\![T]\!]$, sample msk_t^* independently from $\mathcal{D}$ subject to $\mathsf{idx}_t \in \mathsf{Set}(\mathsf{msk}_t^*)$.*

Due to Lemma 5.2 and the privacy of the underlying mPPRS, for any sequence of queries $\mathsf{idx}_0, \ldots, \mathsf{idx}_{T-1}$, for each server $s \in \{1, \ldots, S-1\}$, its view is identically distributed as the output of calling the simulator $\mathsf{mPPRS.Sim}(n, S, \epsilon, s)$ a total of T times. Server 0 is involved only in the preprocessing phase, and clearly its view does not depend on any of the queries.

5.3 Correctness Proof

We prove adaptive correctness as long as the preprocessing server is not controlled by the adversary. Note that this directly implies non-adaptive correctness. Later in Section 10, we devise a new upgrade for achieving full adaptive correctness in the presence of $S-1$ colluding servers.

We first analyze correctness for a single-copy scheme without parallel repetition. It suffices to consider preprocessing followed by up to Q queries within one window. We show that if the preprocessing server remains honest, then for any fixed time t, when the adversary selects a fresh query $\mathsf{idx}_t \notin \mathsf{idx}_0, \ldots, \mathsf{idx}_{t-1}$, the scheme in Figure 6 returns the correct answer with probability at least $1/2$, even under adaptive queries. Repeating the protocol $\omega(\log n)$ times amplifies correctness to $1 - \mathsf{negl}(n)$.

In the online phase, the client's message to the servers depends only on the matched msk. By Lemma 5.2, the adversarially chosen queries $\mathsf{idx}_0, \ldots, \mathsf{idx}_t$ are independent of the msk's in the hint table at the start of the t-th query, which are distributed as freshly and independently sampled keys. Hence, we can equivalently fix an arbitrary query sequence $\mathsf{idx}_0, \ldots, \mathsf{idx}_t$ and then sample the msk's in the hint table at the beginning of step t.

The query idx_t may fail only if one of two bad events occurs: 1) no matching hint is found in Step 1, or 2) a corrupted hint is matched. We next bound the probabilities of these events.

No matched hint. There are $4n^{1-\epsilon}$ random keys in the hint table. By Lemma 4.2, for a randomly sampled msk, the probability that it contains a fixed idx is exactly $\frac{1}{n^{1-\epsilon}}$. Therefore, the probability that no matched hint is found is upper bounded by $\left(1 - 1/(n^{1-\epsilon})\right)^{4n^{1-\epsilon}} \leq 1/6$.

Matching a broken hint. A broken hint can be matched only if the current table has keys that contain idx, some of which are broken (i.e., generated as broken hints by past queries), and a broken one is selected. Furthermore, a hint could be broken only if it contains one of the past queries $\mathsf{idx}_0, \ldots, \mathsf{idx}_{t-1}$, since such a query must have refreshed it. Thus, the probability that a hint containing idx is broken is at most the probability that it also matches one of the past queries $\mathsf{idx}_0, \ldots, \mathsf{idx}_{t-1}$.

By Lemma 4.2, for a random msk subject to containing idx, the probability of matching any fixed $\mathsf{idx}' \neq \mathsf{idx}$ is $p \leq n^{\epsilon-1}$. With up to $Q = n^{1-\epsilon}/10$ past queries, the probability that a random such msk matches none of them is at least $1 - pn^{1-\epsilon}/10 \geq 0.9$. Hence, the probability of matching a broken hint is at most 0.1.

Summarizing the above, a query returns an incorrect answer with probability at most $1/6 + 1/10 < 1/2$.

6 Optimizations: Achieving a Logarithmic Factor Improvement

6.1 Improved Construction

In this section, we describe some new optimizations that enable a logarithmic factor improvement in bandwidth and computation for our multi-server PIR scheme. Specifically, the multi-server PIR scheme of Section 5 relies on super-logarithmically many repetitions of the base scheme to boost the correctness probability from constant to all but negligible. In our improved approach, we will build a base scheme with $\widetilde{O}(n^{-\epsilon/S})$ error probability with asymptotically the same bandwidth and computation as the base scheme of Section 5, we then repeat the base scheme $\omega(S/\epsilon)$ times in parallel to boost the correctness probability to all but negligible. When S and and ϵ are both constants, the number of repetitions is only super-constant rather than super-logarithmic. **Partially broken hint technique.** To get a better base scheme as mentioned above, we introduce a new "partially broken hint" technique which relies on non-blackbox properties of our specific mPPRS construction. Our technique improves the standard broken hint technique [GZS24, ISW24] used in Section 5.

More specifically, in Section 5, whenever a hint in the hint table is consumed, we mark the entire entry as broken; and if this entry is ever consumed later, it introduces an error. Our new "partially broken hint" technique reduces error probability due to consuming a broken hint. During the preprocessing, besides the hint table, the client additionally stores a set of backup keys and their associated parities. Once a hint is consumed during a query for idx, we replace it with a backup msk and the corresponding parities for $\{\mathsf{Set}(\mathsf{msk}\langle\{x\},*,\ldots,*\rangle)\}_{x\in[\![n^{\epsilon/S}]\!]}$, but we modify an entry in $\mathsf{msk}[0]$ such that $\mathsf{idx}\in\mathsf{Set}(\mathsf{msk})$. Now, the parities for $\{\mathsf{Set}(\mathsf{msk}\langle\{x\},*,\ldots,*\rangle)\}_{x\in[\![n^{\epsilon/S}]\!]}$ are all correct except for the set $\mathsf{Set}(\mathsf{msk}\langle\{x_0^*\},*,\ldots,*\rangle)$ where $\mathsf{msk}[0][x_0^*]$ is the modified entry. A hint refreshed in the above manner is called a partially broken hint. If a partially broken hint is consumed by a subsequent query idx', it will only introduce an error if the parity for $\{\mathsf{Set}(\mathsf{msk}\langle\{x_0^*\},*,\ldots,*\rangle)\}_{x\in[\![n^{\epsilon/S}]\!]}$ is needed for idx', where x_0^* is the modified entry mentioned above. Informally speaking, using this new technique, even when a broken hint is consumed, with only $1/n^{\frac{\epsilon}{S}}$ probability do we encounter an error.

Construction. Figure 7 presents our improved scheme where the notable changes are marked in blue. In the base scheme, we now have the client store $O(\epsilon\cdot n^{1-\epsilon}\log n/S)$ entries rather than $O(n^{1-\epsilon})$ — this way, the probability of not finding a matched hint is also $O(n^{-\epsilon/S})$, roughly equal to the error probability due to consuming a partially broken hint.

Efficiency. We first analyze the efficiency of the basic scheme in Figure 7, *without* the parallel repetitions. The repetitions only incur $\omega(S/\epsilon)$ blowup in all efficiency measures.

- *Preprocessing cost.* In the preprocessing phase, server computes and sends $2Q = 2\epsilon\cdot n^{1-\epsilon}\log n/S$ hints, where each hint requires $O(n^\epsilon)$ computation and $O(Sn^{\epsilon/S})$ communication. Thus the server computation is $O(\epsilon\cdot n\log n/S)$, and communication is $O(\epsilon\cdot n^{1-\epsilon+\epsilon/S}\log n)$. Amortized to a window of Q queries, the offline server computation is $O(n^\epsilon)$ and the offline bandwidth is $O(Sn^{\epsilon/S})$ per query.
- *Query costs.* For each query, the client first takes $O(\epsilon\cdot n^{1-\epsilon}\log n)$ computation to find a matching hint. Then, the client sends a punctured key to each server, and each server takes at most $O(n^{\epsilon-\epsilon/S})$ computation to compute the parities. The online bandwidth is $O(S\cdot n^{\epsilon/S})$. The online per-server computation is bounded by $O(n^{\epsilon-\epsilon/S})$. The online client computation is $O(\epsilon\cdot n^{1-\epsilon}\log n + S^2\cdot n^{\epsilon/S})$.

Improved S-server PIR: unbounded queries, negligible error probabability

// Let $\mathsf{mPPRS} = (\mathsf{Gen}, \mathsf{Set}, \mathsf{Test}, \mathsf{GetIndex}, \mathsf{Puncture}, \mathsf{Subsets})$ *be the scheme in Figure 5.*

The following scheme is repeated in parallel $\omega(S/\epsilon)$ number of times. Correctness is guaranteed for each query as long as one instance does not return $\perp$.

Preprocessing. *// Rerun every* $Q = \frac{\epsilon}{S} \cdot n^{1-\epsilon} \log n$ *queries, can be pipelined*

- The preprocessing server (i.e., server 0) repeats the following procedure for $2Q$ times:
  - Sample $\mathsf{msk} \xleftarrow{\$} \mathsf{Gen}(n, S, \epsilon)$, send msk to the client.
  - For each $x \in [\![n^{\epsilon/S}]\!]$, let $v_x \leftarrow \bigoplus_{j \in \mathsf{Set}(\mathsf{msk}, \langle \{x\}, *, \cdots, * \rangle)} \mathsf{DB}[j]$; send v_x to the client.
- The client receives $2Q$ hints each of the form $\left(\mathsf{msk}, \{v_x\}_{x \in [\![n^{\epsilon/S}]\!]}\right)$. The first Q hints form the *hint table*, the remaining Q hints are *backup hints*.

Query for index $\mathsf{idx} \in [\![n]\!]$**.** If idx was queried before, the client simply returns the same answer as before; moreover, it runs the query protocol for an arbitrary idx' that has not been queried before. Otherwise, run the following query protocol.

1. The client randomly selects an entry $(\mathsf{msk}, \{v_x\}_x)$ from the hint table such that $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$. If no such hint is found, sample a fresh msk subject to $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, and let $\{v_x\}_x = \{\perp\}$. *// made efficient through the efficient membership test* Test
2. The client computes $\mathsf{sk}_0, \cdots, \mathsf{sk}_{S-1} \leftarrow \mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})$, and sends sk_s to server s for each $s \in \{1, \ldots, S-1\}$.
3. For each $s \in \{1, \ldots, S-1\}$, server s receives sk_s, computes $\mathsf{List}_s \leftarrow \mathsf{Subsets}(s, \mathsf{sk}_s)$, and for each $y \in [\![|\mathsf{List}_s|]\!]$, it returns $\mathsf{Parity}(\mathsf{List}_s[y]) = \bigoplus_{j \in \mathsf{List}_s[y]} \mathsf{DB}[j]$ to the client.
4. The client computes $y_0, \ldots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$, let p_s denote the y_s-th parity received from server s, and outputs the answer $v_{y_0} \oplus \bigoplus_{s=1}^{S-1} p_s$ — if any of the bits in the xor is the symbol $\perp$, the answer is defined to be $\perp$.
5. If a matching hint was found in Step 1: the client replaces the consumed entry with the following: fetch a new backup hint $\left(\mathsf{msk}, \{v_x\}_{x \in [\![n^{\epsilon/S}]\!]}\right)$, and compute $y_0, \cdots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$. Let msk' be equal to msk except that $\mathsf{msk}'[0][y_0]$ is replaced with $\mathsf{idx} \oplus \bigoplus_{i=1}^{S-1} \mathsf{msk}[i][y_i]$, and let v' be equal to v except that v'_{y_0} is replaced with $\perp$. Replace the consumed hint with $\left(\mathsf{msk}', \{v'_x\}_{x \in [\![n^{\epsilon/S}]\!]}\right)$.

Figure 7: Improved S-server preprocessing PIR

- *Client space.* The client space is bounded by $O(\epsilon \cdot n^{1-\epsilon+\epsilon/S} \log n)$, for storing $2\epsilon \cdot n^{1-\epsilon} \log n/S$ hints.

Summarizing the above, we have the following corollary:

Theorem 6.1 (Our improved S-server PIR scheme)**.** *Suppose that n is a power of 2, $\frac{\epsilon}{S} \cdot \log_2 n$ is a positive integer, and n is reasonably large such that $1 + \frac{\epsilon}{S} \cdot \log_2 n < n^{\epsilon/2S}$. Let $\alpha(n)$ be any function of n such that $\alpha(n) = \omega(S/\epsilon)$. Then, there exists an information-theoretic S-server PIR scheme with the following performance bounds. The one-time preprocessing incurs $O(\epsilon \cdot n(\log n/S) \cdot \alpha(n))$ server computation and $O(\epsilon \cdot n^{1-\epsilon+\epsilon/S} \log n \cdot \alpha(n))$ communication. Afterwards, every query incurs $O(S \cdot n^{\epsilon/S} \cdot \alpha(n))$ online and offline bandwidth, $O(n^{\epsilon-\epsilon/S} \cdot \alpha(n))$ online server computation, $O(n^{\epsilon} \cdot \alpha(n))$ offline server computation, $O((\epsilon \cdot n^{1-\epsilon} \log n + S^2 \cdot n^{\epsilon/S}) \cdot \alpha(n))$ online client computation, and $O(S \cdot n^{\epsilon/S} \cdot \alpha(n))$ offline client computation. Further, the client space is bounded by $O(\epsilon \cdot n^{1-\epsilon+\epsilon/S} \log n \cdot \alpha(n))$.*

The proof of the above Theorem 6.1 is presented in Section 6.2.

6.2 Proof of Theorem 6.1

Privacy proof. Due to the same argument as Section 5.2, it suffices to prove that the modified backup key used to replace a consumed key is distributed as sampling a random key subject to containing the present query idx. In other words, it suffices to prove the following lemma.

Lemma 6.2. *For any $\mathsf{idx} \in [\![n]\!]$, the following two distributions are identical:*

1. *Sample msk from $\mathsf{Gen}(n, S, \epsilon)$, let $y_0, \cdots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$, and let msk' be equal to msk except that $\mathsf{msk}'[0][y_0]$ is replaced with $\mathsf{idx} \oplus \bigoplus_{i=1}^{S-1} \mathsf{msk}[i][y_i]$. Output msk'.*
2. *Sample msk from $\mathsf{Gen}(n, S, \epsilon)$ subject to $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$. Output msk.*

Proof. The proof of Lemma 4.2 implies that $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$ iff

$$\mathsf{LSB}^{(1-\epsilon)\log_2 n}\left(\bigoplus_{i \in [\![S]\!]} \mathsf{msk}[i][y_i]\right) = \mathsf{LSB}^{(1-\epsilon)\log_2 n}(\mathsf{idx}) \tag{2}$$

where $y_0, \ldots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$, and $\mathsf{LSB}^{(1-\epsilon)\log_2 n}(\cdot)$ means taking the least significant $(1-\epsilon)\log_2 n$ bits of the input value. Moreover, the $\mathsf{GetIndex}$ function does not look at the least significant $(1-\epsilon)\log_2 n$ bits of the entries in msk.

Therefore, the second distribution in the lemma is equivalent to sampling all other random bits except the least significant $(1-\epsilon)\log_2 n$ bits of every entry in msk, computing $y_0, \ldots, y_{S-1}$, and then sampling the least significant $(1-\epsilon)\log_2 n$ bits subject to Equation (2). This is equivalent to the first distribution in the lemma. ∎

Correctness proof. We prove adaptive correctness as long as the preprocessing server is not corrupted. This immediately implies non-adaptive correctness. Later in Section 10, we devise a new upgrade for achieving full adaptive correctness in the presence of $S-1$ colluding servers.

We first analyze a single copy of the scheme without the parallel repetition. By the same argument as Section 5.3, we can equivalently fix an arbitrary sequence of queries $\mathsf{idx}_0, \mathsf{idx}_1, \ldots, \mathsf{idx}_t$ and then sample the msk's in the hint table at the beginning of step t. The query idx_t may fail only if one of two bad events occurs:

1. *No matched hint.* The hint table at the beginning of step t does not contain an entry matching idx_t;
2. *A partially broken hint that affects* idx_t *is matched.* The matched hint is partially broken, and the broken position affects idx_t.

We first bound the probability of not finding a matched hint. With $\epsilon \cdot n^{1-\epsilon} \log n/S$ independent random hints in the hint table, the probability of not having a matched hint is at most $(1 - 1/n^{1-\epsilon})^{\epsilon \cdot n^{1-\epsilon} \log n/S} < n^{-\epsilon/S}$.

We next bound the probability that the matched hint is partially broken and the broken position affects idx_t. By Lemma 5.2, the matched msk in step t is distributed as a freshly sampled msk subject to containing idx_t. It can only be partially broken with the broken position directly affecting idx_t if there exists some previously queried index idx' such that

1. $\mathsf{idx}' \in \mathsf{Set}(\mathsf{msk})$;
2. Let $y_0, \ldots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$, let $y'_0, \ldots, y'_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx}')$, then $y_0 = y'_0$.

For some previous query $\mathsf{idx}' \neq \mathsf{idx}_t$, we say that msk is plausibly destroyed by idx' if the above bad events happen. The following fact gives an upper bound for a fixed idx', the probability that a random msk subject to containing $\mathsf{idx} = \mathsf{idx}_t$ is destroyed by idx'.

Fact 6.3. *For any* $\mathsf{idx}, \mathsf{idx}' \in [\![n]\!]$ *where* $\mathsf{idx} \neq \mathsf{idx}'$*, sample* $\mathsf{msk} \xleftarrow{\$} \mathsf{mPPRS.Gen}(n, S, \epsilon)$ *subject to* $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$*, let* $y_0, \cdots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$*, and let* $y'_0, \cdots, y'_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx}')$*. The probability that* $y_0 = y'_0$ *and* $\mathsf{idx}' \in \mathsf{Set}(\mathsf{msk})$ *is at most* $1/n^{1-\epsilon+\epsilon/S}$*.*

Proof. The constraint that $y_0 = y'_0$ and $\mathsf{idx}' \in \mathsf{Set}(\mathsf{msk})$ is equivalent to

$$\mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx}') = (y_0, y'_1, \ldots, y'_{S-1}) \text{ and } \mathsf{idx}' = \mathsf{msk}[0][y_0] \oplus \bigoplus_{i=1}^{S-1} \mathsf{msk}[i][y'_i]$$

Given $\mathsf{idx} \neq \mathsf{idx}'$, there must exists some $i \in \{1, \ldots, S-1\}$ such that $y'_i \neq y_i$, or $\mathsf{idx}' = \mathsf{msk}[0][y_0] \oplus \bigoplus_{i=1}^{S-1} \mathsf{msk}[i][y'_i] = \mathsf{idx}$ cannot hold.

Recall that the result of $\left(_, y'_1, \ldots, y'_{S-1}\right) \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx}')$ only depends on the most significant $(\epsilon - \epsilon/S) \log_2 n$ bits of entries in msk. Thus, conditioning on $\mathsf{idx} = \bigoplus_{i=0}^{S-1} \mathsf{msk}[i][y_i]$ and $\left(_, y'_1, \ldots, y'_{S-1}\right) = \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx}')$, the least significant $(1-\epsilon+\epsilon/S) \log_2 n$ bits of $\bigoplus_{i=1}^{S-1} \mathsf{msk}[i][y'_i]$ are still random. Hence, the probability of $\mathsf{idx}' = \mathsf{msk}[0][y_0] \oplus \bigoplus_{i=1}^{S-1} \mathsf{msk}[i][y'_i]$ is at most $1/n^{1-\epsilon+\epsilon/S}$. ∎

Taking union bound over the up to $Q = \frac{\epsilon}{S} \cdot n^{1-\epsilon} \log n$ prior queries that are not equal to idx_t, the probability that a random msk_t subject to containing idx is plausibly destroyed by one of the Q queries is upper bounded by $Q/n^{1-\epsilon+\epsilon/S} \leq \frac{\epsilon}{S} n^{-\epsilon/S} \log n$. Therefore, the probability of matching a partially broken hint that affects idx is bounded by $\frac{\epsilon}{S} n^{-\epsilon/S} \log n$.

Accounting for both bad events, we have that for a single copy, the correctness probability for a new query is at least $1 - n^{-\epsilon/S} - \frac{\epsilon}{S} \cdot n^{-\epsilon/S} \log n = 1 - (1 + \epsilon \log n/S)\, n^{-\epsilon/S}$.

Therefore, if we repeat the scheme $\omega(S/\epsilon)$ times in parallel, the correctness probability for each new query idx is boosted to

$$1 - \left((1 + \epsilon \log n/S)\, n^{-\epsilon/S}\right)^{\omega(S/\epsilon)} \qquad (\star)$$

Metric	Zelda	QuarterPIR
Prep. Comp.	$O(Wn\log(1/\delta))$	$O(n\log(1/\delta))$ PRF eval. + $O(Wn\log(1/\delta))$
Prep. BW / Storage	$O(\sqrt{W\log n}\, n^{2/3}\log(1/\delta))$	$O(Wn^{1/2}\log(1/\delta))$
BW		
Online	$O(\sqrt{W/\log n}\, n^{1/3}\log(1/\delta))\uparrow$	$O((W+\lambda)n^{1/4})\uparrow\downarrow$
	$O(W\log_n(1/\delta))\downarrow$	
Offline	$O(W^{3/2}/\sqrt{\log n}\, n^{1/3})\downarrow$	
Server Comp.		
Online	$O(W^{3/2}/\sqrt{\log n}\, n^{1/3}\log_n(1/\delta))$	$O(n^{1/2})$ PRF eval. + $O(Wn^{1/2})$
Offline	$O(Wn^{2/3}\log_n(1/\delta))$	
Client Comp.	$O(n^{1/3}\log(1/\delta))$ + BW	$O(n^{1/2})$ PRF eval. + BW

$\uparrow,\downarrow$ denote upload and download respectively.

Table 2: **Asymptotic comparison of Zelda and QuarterPIR under general entry size W.** The asymptotic costs for Zelda rely on the long-entry optimizations described in Section 8.3.

For sufficiently large n such that $1+\frac{\epsilon}{S}\cdot\log n < n^{\epsilon/2S}$,

$$(\star) \geq 1-\left(n^{-\Theta(\epsilon/S)}\right)^{\omega(S/\epsilon)} \geq 1-\mathsf{negl}(n)$$

for some negligibly small function $\mathsf{negl}(\cdot)$.

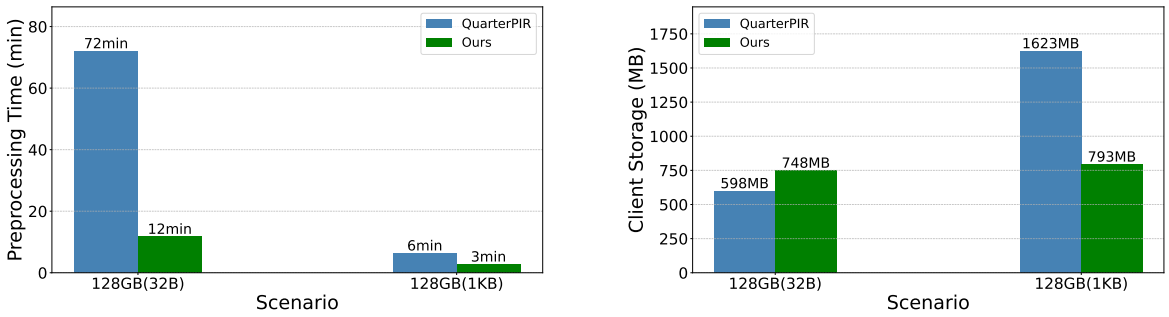
Remark 6.4. *When $S=O(1)$ and $\epsilon=\Omega(1)$ are constants, if we want the correctness failure probability to be bounded by δ, it suffices to set the number of repetitions to $O(\log\frac{1}{\delta}/\log n)$ — we use this generalized form in Section 7 for setting concrete parameters.*

7 Empirical Evaluation

In this section, we evaluate the concrete efficiency of our PIR construction. Our evaluation results shows that Zelda outperforms a state-of-the-art PIR scheme called QuarterPIR in terms of online response time and client space (assuming deployment over a typical fiber optic link), at the price of increased costs for offline maintanence operations.

Practical optimizations, implementation, and open source. In our implementation, we instantiate Zelda with 2 servers and $\epsilon=2/3$, with the additional concrete optimizations documented in Section 8. Our code base is available at https://github.com/p-b-p-b/Zelda.

Baseline of comparison. We evaluate the performance of Zelda and compare it with a state-of-the-art two-server PIR scheme [GZS24] (called QuarterPIR) based on pseudorandom functions (PRFs). QuarterPIR has $O(\sqrt{n})$ online computation, $\tilde{O}_\lambda(n^{1/4})$ communication per query with $\tilde{O}_\lambda(\sqrt{n})$ client storage, which achieves the optimal time-storage tradeoff [CK20, CHK22] and the smallest online communication complexity among the known *practical* two-server PIR schemes [ZPSZ24, NGH24, RMS24, WLZ+23]. To give the reader more context to interpret the comparison results, we spell out the performance of the two schemes in more detail for long entries of width W, and for a correctness failure probability of δ. The comparison can be found in Table 2.



(a) Preprocessing Time

(b) Client Storage

Figure 8: Preprocessing Costs of Zelda and QuarterPIR.

For the purpose of a fair comparison in the two-server setting, we make certain modification and optimizations to the original QuarterPIR implementation, as the original codebase is intended for the one-server setting. We document the details of our modifications in Section 8.4.

Parameters and evaluation setup. We implement our scheme in Golang atop the QuarterPIR codebase [GZS24], using a computational privacy parameter of 128 for [GZS24] and correctness failure probability $\delta = 2^{-40}$ for both schemes, following [GZS24]. The preprocessing server uses 8 threads, while the other server and client each use a single thread.

All experiments are run on a single commercial server (Intel Xeon(R) Platinum 8352S @ 2.20 GHz, 512 GB RAM) simulating both servers and the client via separate processes. This machine is slightly less performant than the AWS m5.8xlarge instance used in [GZS24].

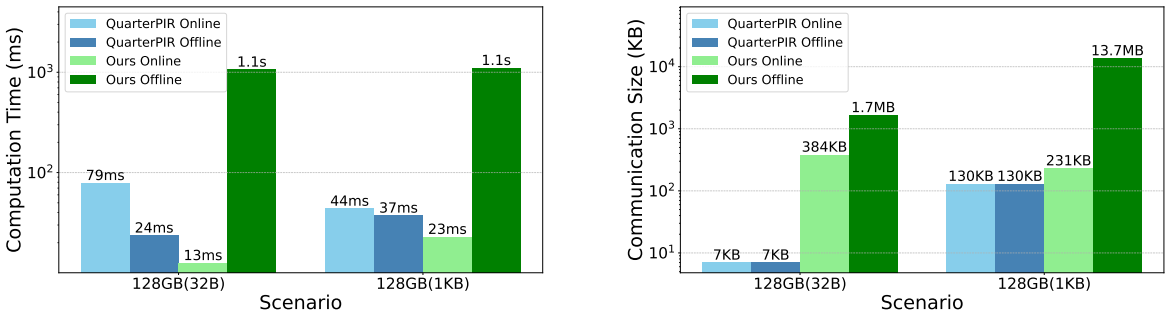
We benchmark both schemes on 128 GB databases with entry sizes of 32 B and 1 KB, representing small and large entry scenarios, respectively.

7.1 One-time Preprocessing Costs

Computation Costs. Asymptotically, both Zelda and QuarterPIR have $O(Wn\log(1/\delta))$ non-cryptographic computation, while QuarterPIR has additional $O(n\log(1/\delta))$ PRF evaluations. Our evaluation shows that the concrete preprocessing server computation time for Zelda is 12min and 3min for the small and large entry scenarios, respectively, while QuarterPIR takes 72min and 6min. The overhead of PRF evaluations in QuarterPIR, which is more significant in the small entry scenario, is the main reason for the difference.

Storage and Communication Costs. In the 2-server setting, the client storage requirement exactly corresponds to the preprocessing communication cost per client, because a server will directly send the preprocessed hints to the client. Our scheme has $O(\sqrt{W\log n}\, n^{2/3}\log(1/\delta))$ client storage requirement, so the dependence on n is worse than QuarterPIR's $O(Wn^{1/2}\log(1/\delta))$. However, our scheme has comparable or even better client storage costs in practice, requiring 748MB and 793MB for the small and large entry scenarios respectively, while QuarterPIR requires 598MB and 1623MB. We have a better dependency on W because our client storage for each hint has two parts:

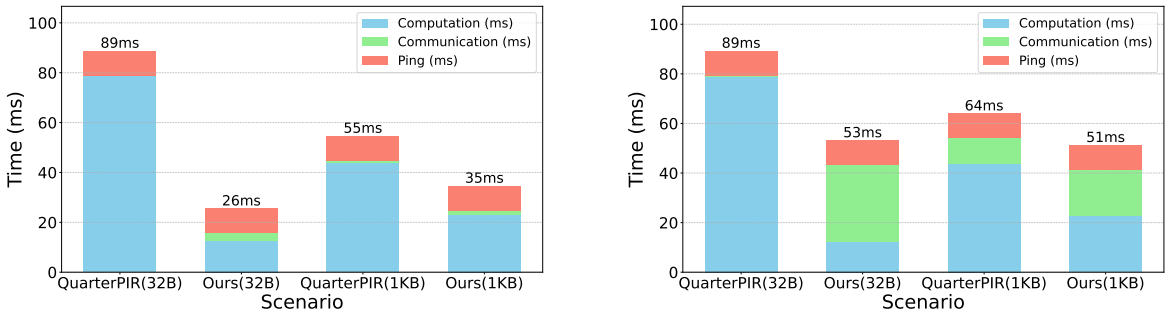
1. The key of the mPPRS (containing random coins), whose length is $(\mathsf{rlen}_0 + \mathsf{rlen}_1)\log n$; and
2. A parity block, whose length is $\mathsf{rlen}_0 \cdot W$.



(a) Computation Cost

(b) Communication Cost

Figure 9: Query Costs of ZELDA and QuarterPIR.



(a) 1 Gbps bandwidth

(b) 100 Mbps bandwidth

Figure 10: Concrete online response time of ZELDA and QuarterPIR.

Therefore we can adjust $\mathsf{rlen}_0, \mathsf{rlen}_1$ to balance the two parts, while keeping their product unchanged. See Section 8.3 for details.

7.2 Query Performance

Computation costs. ZELDA exhibits a distinct computation profile from QuarterPIR. Its online phase scales better with n, whereas the offline phase scales worse. The empirical results are shown in Figure 9a. Concretely, ZELDA achieves online computation times of 13.0 ms and 23.0 ms for the small and large entry settings, respectively, representing $1.9 \times \sum 6.0\times$ improvement over QuarterPIR. This gain stems from ZELDA's better asymptotic complexity and the avoidance of expensive cryptographic primitives, which dominate QuarterPIR's cost in the small entry case. In contrast, ZELDA incurs higher offline cost — about 1.1 s per query in both scenarios — despite faster preprocessing. This is because its preprocessing is amortized over only $\tilde{O}(n^{1/3})$ queries (see Figure 7), compared to $\tilde{O}(n^{1/2})$ in QuarterPIR. Nonetheless, the higher offline cost is typically non-critical, since it runs in the background and does not affect the query's critical path.

Communication costs. Our evaluation (Figure 9b) shows that ZELDA takes 231 to 384 KB to process a query in the online phase, depending on the scenario. For the offline phase, ZELDA takes 1.7 to 13.7 MB communication per query, which is much higher due to the multiplicative factor on W. We argue that the online communication cost is barely noticeable for typical fiber optics links, and the offline communication cost is not a bottleneck in practice either.

Online response time perceived by the client. We calculate the concrete online response time of Zelda and QuarterPIR, assuming either 1 Gbps bandwidth (e.g., on a typical fiber optics link) or 100 Mbps bandwidth (e.g., on a typical home WiFi), and a ping latency 10 ms which is characteristic of having a content discovery network like Akamai hosting the PIR service, such that the client can contact geographically nearby servers. The cost breakdown is shown in Figure 10. Under 1 Gbps bandwidth, Zelda achieves a $1.5\times \sim 3.4\times$ improvement in online response time compared to QuarterPIR, depending on the scenario. The ping latency and the online computation time are the main bottlenecks, while the online communication cost is barely noticeable. Under 100 Mbps bandwidth, we are still able to achieve a $1.2\times \sim 1.6\times$ improvement. The main improvement of Zelda over QuarterPIR comes from the reduction in computation cost.

7.3 Further Comparison with Ren et al.

Ren et al. [RMS24] is another state-of-the-art two-server PIR scheme based on pseudorandom functions (PRFs). It has $O(\sqrt{n})$ computation and communication per query (both online and offline), and $O_\lambda(\sqrt{n})$ client storage. Although their online communication has worse dependence on n than QuarterPIR, they provide an efficient implementation with optimized constant factors. We thus compare with Ren et al. [RMS24] in terms of concrete efficiency. Because their implementation is in C++ whereas ours is in Go, wall-clock runtimes are not directly comparable. Our comparison below focuses on language-agnostic metrics including communication and client storage.

One advantage of our scheme is that the performance improves when we add more servers, whereas Ren et al. is fixed to two servers. We start to see a significant improvement in online communication over Ren et al. when instantiating our scheme with 3 servers. Letting $\epsilon = 2/3$, our communication complexity is $O(n^{2/9} \log_n(1/\delta))$ at 3 servers, which is asymptotically better than their $O(n^{1/2})$; and our client space of $O_\lambda(n^{5/9})$ is roughly on par with their $O_\lambda(n^{1/2})$). In the 128GB/32B scenario, our online communication is 20% of their scheme when matching their client storage and keeping the offline computation under 1 second. For a larger number of servers the gap will only be bigger.

7.4 Case Study: Private DNS Lookups

Private DNS lookup is a compelling application in which the online response time matters more to user experience, which justifies paying a price in offline work to optimize the online response time. Specifically, a user action (e.g., loading a web page or opening an app) typically triggers a short burst of DNS resolutions, followed by a long quiet period. Empirical studies report that DNS clients exhibit highly bursty patterns with long inter-lookup gaps [SRA16, SBK+20]. This behavior lets a system push expensive work off the critical path while keeping the interactive phase fast.

In our implementation, the end-to-end online latency can be as low as 26ms on a 128GB database with 32-byte records, whereas QuarterPIR requires 89ms. On the other hand, our scheme has 1.1 seconds of server offline computation (with 8 cores), whereas QuarterPIR's offline computation is under 100ms (with 1 core). We stress, however, that our offline work is 1) independent of the user's query; and 2) easily parallelizable. In other words, one could prepare the offline work for an unbounded number of queries upfront; and with unbounded parallelism, the offline stage can be completed in time that is independent of the number of queries.

Acknowledgments

This work is in part supported by a grant from ONR, a grant from the DARPA SIEVE program under a subcontract from SRI, a gift from Cisco, Samsung MSL, NSF awards under grant numbers 1705007, 2128519 and 2044679.

Part of Ashrujit Ghoshal's work is supported by a New Faculty Initiation Grant at IIT Madras and by the CyStar Centre of Excellence.

References

[BFG03] Richard Beigel, Lance Fortnow, and William I. Gasarch. A nearly tight bound for private information retrieval protocols. *Electronic Colloquium on Computational Complexity (ECCC)*, 2003. 2

[BGKM22] Vishwas Bhargava, Sumanta Ghosh, Mrinal Kumar, and Chandra Kanta Mohapatra. Fast, algebraic multivariate multipoint evaluation in small characteristic and applications. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, page 403–415, New York, NY, USA, 2022. Association for Computing Machinery. 6

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In *CRYPTO*, pages 55–73, 2000. 2, 6

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In *TCC*, 2017. 2

[CG97] Benny Chor and Niv Gilboa. Computationally private information retrieval. In *STOC*, 1997. 2

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, 1995. 1, 2

[Cha04] Yan-Cheng Chang. Single database private information retrieval with logarithmic communication. In *ACISP*, 2004. 2

[CHK22] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In *Eurocrypt*, 2022. 2, 4, 6, 28

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In *TCC*, 2017. 2

[CK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *EUROCRYPT*, 2020. 2, 4, 6, 28

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In *EUROCRYPT*, pages 402–414, 1999. 2

[DG16] Zeev Dvir and Sivakanth Gopi. 2-server pir with subpolynomial communication. *Journal of the ACM (JACM)*, 63(4):1–15, 2016. 2, 5

[DRRT18] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: scaling private contact discovery. *Proc. Priv. Enhancing Technol.*, 2018(4):159–178, 2018. 2

[Fea] Nick Feamster. Oblivious DNS deployed by Cloudflare and Apple. https://medium.com/noise-lab/oblivious-dns-deployed-by-cloudflare-and-apple-1522ccf53cab. 2, 38

[Gas04] William I. Gasarch. A survey on private information retrieval. *Bulletin of the EATCS*, 82:72–107, 2004. 2

[GLM+24] Ashrujit Ghoshal, Baitian Li, Yaohua Ma, Chenxin Dai, and Elaine Shi. Information-theoretic multi-server PIR with global preprocessing. Cryptology ePrint Archive, Paper 2024/765, 2024. 2, 6

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *ICALP*, 2005. 2

[GZS24] Ashrujit Ghoshal, Mingxun Zhou, and Elaine Shi. Efficient pre-processing pir without public-key cryptography. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 210–240. Springer, 2024. 2, 4, 5, 6, 7, 24, 28, 29, 39, 40, 43

[HDCG+23] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, , and Nickolai Zeldovich. Private web search with Tiptoe. In *29th ACM Symposium on Operating Systems Principles (SOSP)*, Koblenz, Germany, October 2023. 2

[HHCG+22] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. Cryptology ePrint Archive, Paper 2022/949, 2022. https://eprint.iacr.org/2022/949. 2

[HPPY24] Alexander Hoover, Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Plinko: Single-server PIR with efficient updates via invertible prfs. *IACR Cryptol. ePrint Arch.*, page 318, 2024. 6, 7

[ISW24] Yuval Ishai, Elaine Shi, and Daniel Wichs. Pir with client-side preprocessing: information-theoretic constructions and lower bounds. In *Annual International Cryptology Conference*, pages 148–182. Springer, 2024. 2, 3, 4, 5, 6, 7, 8, 9, 12, 13, 14, 21, 22, 24, 40, 43

[KO97] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *FOCS*, 1997. 2

[KU11] Kiran S. Kedlaya and Christopher Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, December 2011. 6

[Lip09] Helger Lipmaa. First CPIR protocol with data-dependent computation. In *ICISC*, 2009. 2

[LLFP24] Arthur Lazzaretti, Zeyu Liu, Ben Fisch, and Charalampos Papamanthou. Multi-server doubly efficient PIR. *IACR Cryptol. ePrint Arch.*, page 829, 2024. 2, 6

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic ram computation from ring lwe. In *STOC*, 2023. 2, 6

[LP23a] Arthur Lazzaretti and Charalampos Papamanthou. Near-optimal private information retrieval with preprocessing. In *Theory of Cryptography*, pages 406–435, Cham, 2023. Springer Nature Switzerland. 2, 6

[LP23b] Arthur Lazzaretti and Charalampos Papamanthou. Treepir: Sublinear-time and polylog-bandwidth private information retrieval from ddh. In *CRYPTO*, 2023. 2, 5, 6, 7, 40, 43

[MCG+08] Carlos Aguilar Melchor, Benoit Crespin, Philippe Gaborit, Vincent Jolivet, and Pierre Rousseau. High-speed private information retrieval computation on GPU. In *Proceedings of the 2008 Second International Conference on Emerging Security Information, Systems and Technologies*, SECURWARE '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society. 2

[MG07] Carlos Aguilar Melchor and Philippe Gaborit. A lattice-based computationally-efficient private information retrieval protocol. *IACR Cryptology ePrint Archive*, 2007:446, 2007. 2

[MW22] Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *IEEE S&P*, 2022. 2

[NGH24] Hoang-Dung Nguyen, Jorge Guajardo, and Thang Hoang. Client-efficient online-offline private information retrieval. Cryptology ePrint Archive, Paper 2024/719, 2024. 28

[obl] Oblivious dns over https. https://tools.ietf.org/html/draft-pauly-dprive-oblivious-doh-04. 38

[OG11] Femi G. Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Financial Cryptography*, pages 158–172, 2011. 2

[OPPW23] Hiroki Okada, Rachel Player, Simon Pohmann, and Christian Weinert. Towards practical doubly-efficient private information retrieval. Cryptology ePrint Archive, Paper 2023/1510, 2023. 6

[OS07] Rafail Ostrovsky and William E. Skeith, III. A survey of single-database private information retrieval: techniques and applications. In *PKC*, pages 393–411, 2007. 2

[PPY18] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In *CCS*, 2018. 2

[RMS24] Ling Ren, Muhammad Haris Mughees, and I Sun. Simple and practical amortized sublinear private information retrieval using dummy subsets. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 1420–1433, 2024. 2, 5, 6, 7, 28, 31, 40, 43

[SACM21] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce Maggs. Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In *CRYPTO*, 2021. 2, 6, 7

[SBK+20] Kyle Schomp, Onkar Bhardwaj, Eymen Kurdoglu, Mashooq Muhaimen, and Ramesh K. Sitaraman. Akamai dns: Providing authoritative answers to the world's queries. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '20, page 465–478, New York, NY, USA, 2020. Association for Computing Machinery. 31

[SC07] Radu Sion and Bogdan Carbunar. On the computational practicality of private information retrieval. In *Network and Distributed Systems Security Symposium (NDSS)*, 2007. 2

[SCV+20] Sudheesh Singanamalla, Suphanat Chunhapanya, Marek Vavruša, Tanya Verma, Peter Wu, Marwan Fayed, Kurtis Heimerl, Nick Sullivan, and Christopher Wood. Oblivious dns over https (odoh): A practical privacy enhancement to dns, 2020. 38

[SRA16] Kyle Schomp, Michael Rabinovich, and Mark Allman. Towards a model of dns client behavior. In Thomas Karagiannis and Xenofontas Dimitropoulos, editors, *Passive and Active Measurement*, pages 263–275, Cham, 2016. Springer International Publishing. 31

[SWZ24] Jaspal Singh, Yu Wei, and Vassilis Zikas. Information-theoretic multi-server private information retrieval with client preprocessing. In *Theory of Cryptography Conference*, pages 423–450. Springer, 2024. 2, 3, 4, 5, 6, 7, 12, 13, 14, 40, 43

[WLZ+23] Yinghao Wang, Xuanming Liu, Jiawen Zhang, Jian Liu, and Xiaohu Yang. Crust: Verifiable and efficient private information retrieval with sublinear online time. Cryptology ePrint Archive, Paper 2023/1607, 2023. 28

[WR25] Zhikun Wang and Ling Ren. Single-server client preprocessing pir with tight space-time trade-off. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025*, pages 94–122, Cham, 2025. Springer Nature Switzerland. 7

[WY05] David Woodruff and Sergey Yekhanin. A geometric approach to information-theoretic private information retrieval. In *20th Annual IEEE Conference on Computational Complexity (CCC'05)*, pages 275–284, 2005. 2, 6

[ZLTS23] Mingxun Zhou, Wei-Kai Lin, Yiannis Tselekounis, and Elaine Shi. Optimal single-server private information retrieval. In *EUROCRYPT*, 2023. 2, 6

[ZPSZ24] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server pir with sublinear server computation. In *IEEE S& P*, 2024. 2, 5, 6, 7, 8, 9, 12, 21, 22, 28, 40, 43

8 Practical Optimizations

In this section, we present additional practical optimizations for our PIR scheme. These optimizations are adopted in our Zelda implementation.

8.1 Online Download Bandwidth Optimization

We describe a further optimization that makes non-blackbox use of the structure of our mPPRS to reduce the online download bandwidth from $n^{\epsilon/S} \cdot \alpha(n)$ entries per server to $\alpha(n)$ entries per server, where $\alpha(\cdot)$ is an arbitrarily small super-constant function.

Intuition. Recall that in the mPPRS construction of Figure 5,

$$\begin{aligned}\mathsf{List}_s[y_s] &= \mathsf{Set}\big(\mathsf{sk}_s\langle *, \cdots, *, * \setminus \{y_s\}, *, \cdots, *\rangle\big) \\ &= \mathsf{Set}\big(\mathsf{msk}\langle \{y_0\}, \cdots, \{y_{s-1}\}, * \setminus \{y_s\}, *, \cdots, *\rangle\big)\end{aligned}$$

where $\mathsf{List}_s \leftarrow \mathsf{Subsets}(s, \mathsf{sk}_s)$, and $y_0, \cdots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$. We need to have $* \setminus \{y_s\}$, i.e., removing y_s from the s-th row, because $\mathsf{sk}_s[s][y_s]$ was rerandomized during $\mathsf{Puncture}$. When the mPPRS is used in the PIR scheme, the server must return the parity for $\mathsf{Set}(\mathsf{sk}_s\langle *, \ldots, * \setminus \{y_s\}, \ldots, *\rangle)$ for all possible choices of y_s. Observe that

$$\mathsf{Set}\big(\mathsf{sk}_s\langle \underbrace{*, \cdots, *}_{s}, * \setminus \{y_s\}, \underbrace{*, \cdots, *}_{S-s-1}\rangle\big) = \mathsf{Set}(\mathsf{sk}_s) \triangle \mathsf{Set}\big(\mathsf{sk}_s\langle \underbrace{*, \ldots, *}_{s}, \{y_s\}, \underbrace{*, \ldots, *}_{S-s-1}\rangle\big) \tag{3}$$

Our idea is for each $s \in \{1, 2, \ldots, S-1\}$, to choose the entry $\mathsf{sk}_s[s][y_s]$ cleverly such that the client has pre-discovered the parity of $\mathsf{Set}(\mathsf{sk}_s\langle \underbrace{*, \ldots, *}_{s-1}, \{y_s\}, \underbrace{* \ldots, *}_{S-s-1}\rangle)$ in the preprocessing phase. This way, the server s only needs to return a single parity for $\mathsf{Set}(\mathsf{sk}_s)$.

To this end, for each msk in the client's hint table, for each $s \in \{1, \ldots, S-1\}$, and for each $x \in [\![n^{\epsilon/S}]\!]$, the client will prepare a complementary key denoted $\mathsf{sk}'_{s,x}$ composed as follows, and download the parity of $\mathsf{Set}(\mathsf{sk}'_{s,x})$ from the preproessing server:

- the first $s-1$ rows contain a single number 0,
- the s-th row contains a single random value chosen from the appropriate distribution (same distribution as the entry $\mathsf{msk}[s][x]$), and
- the remaining rows are copied from the rows of msk.

During the query for idx, when the client computes the punctured key sk_s, it simply sets

$$\mathsf{sk}_s[s][y_s] := \left(\bigoplus_{i \in [\![s]\!]} \mathsf{sk}_s[i][0]\right) \oplus \mathsf{sk}'_{s,y_s}[s][0] = \left(\bigoplus_{i \in [\![s]\!]} \mathsf{msk}_s[i][y_i]\right) \oplus \mathsf{sk}'_{s,y_s}[s][0]$$

This ensures that $\mathsf{Set}\big(\mathsf{sk}_s\langle \underbrace{*, \cdots, *}_{s}, \{y_s\}, \underbrace{*, \cdots, *}_{S-s-1}\rangle\big) = \mathsf{Set}(\mathsf{sk}'_{s,y_s})$, and the client has pre-discovered the parity of this set during preprocessing.

To store each $\mathsf{sk}'_{s,x}$, the client only needs to store the single value in the s-th row since all other rows are either 0 or the same as msk. Therefore, under this new modification, with each hint, the client needs to store an extra of $S \cdot n^{\epsilon/S}$ random indices from $[\![n]\!]$, as well as $S \cdot n^{\epsilon/S}$ parity bits. This does not increase the client's space asymptotically.

Construction. We present the resulting PIR scheme in Figure 11. Changes compared with Figure 7 are highlighted in blue. In this figure, we introduce the additional notation $\overline{\mathsf{msk}}$ for convenience, where $\overline{\mathsf{msk}}[s][x]$ stores the single value in the s-th row of $\mathsf{sk}'_{s,x}$.

Optimization: Minimizing Online Download Bandwidth

// Let $\mathsf{mPPRS} = (\mathsf{Gen}, \mathsf{Set}, \mathsf{Test}, \mathsf{GetIndex}, \mathsf{Puncture}, \mathsf{Subsets})$ *be the scheme in Figure 5.*

The following scheme is repeated in parallel $\omega(S/\epsilon)$ number of times. Correctness is guaranteed for each query as long as one instance does not return $\perp$.

Preprocessing. *// Rerun every* $Q = \frac{\epsilon}{S} \cdot n^{1-\epsilon} \log n$ *queries, can be pipelined*

- The preprocessing server (i.e., server 0) repeats the following procedure for $2Q$ times:
  - Sample $\mathsf{msk}, \overline{\mathsf{msk}} \xleftarrow{\$} \mathsf{Gen}(n, S, \epsilon)$, and send msk and $\overline{\mathsf{msk}}$ to the client.
  - For each $x \in [\![n^{\epsilon/S}]\!]$, let $v_x \leftarrow \bigoplus_{j \in \mathsf{Set}(\mathsf{msk}\langle\{x\},*,\cdots,*\rangle)} \mathsf{DB}[j]$; send v_x to the client.
  - For each $i \in \{1, 2, \cdots, S-1\}$, for each $x \in [\![n^{\epsilon/S}]\!]$:
    - let $\mathsf{sk}'_{i,x}$ be a matrix where the first $i-1$ rows contain a single number 0, the i-th row contains a single number $\overline{\mathsf{msk}}[i][x]$, and the remaining rows are copied from the corresponding rows of msk.
    - let $w_{i,x} \leftarrow \bigoplus_{j \in \mathsf{Set}(\mathsf{sk}'_{i,x})} \mathsf{DB}[j]$; send $w_{i,x}$ to the client.
- The client receives $2Q$ hints each of the form $\left(\mathsf{msk}, \overline{\mathsf{msk}}, \{v_x\}_{x \in [\![n^{\epsilon/S}]\!]}, \{w_{i,x}\}_{i \in \{1,\ldots,S-1\}, x \in [\![n^{\epsilon/S}]\!]}\right)$. The first Q hints form the *hint table*, the remaining Q hints are *backup hints*.

Query for index $\mathsf{idx} \in [\![n]\!]$**.** If idx was queried before, the client simply returns the same answer as before; moreover, it runs the query protocol for an arbitrary idx' that has not been queried before. Otherwise, run the following query protocol.

1. The client randomly selects an entry $(\mathsf{msk}, \overline{\mathsf{msk}}, \{v_x\}_x, \{w_{i,x}\}_{i,x})$ from the hint table such that $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$. If no such hint is found, sample a fresh msk subject to $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$, sample a fresh $\overline{\mathsf{msk}}$, and let $\{v_x\}_x = \{\perp\}, \{w_{i,x}\}_{i,x} = \{\perp\}$.
2. The client computes $\mathsf{sk}_1, \cdots, \mathsf{sk}_{S-1}$, and sends sk_s to server s for each $s \in \{1, \ldots, S-1\}$, where
$$\mathsf{sk}_s = \begin{pmatrix} \mathsf{msk}[0][y_0] \\ \cdots \\ \mathsf{msk}[s-1][y_{s-1}] \\ \mathsf{msk}[s][:]/(y_s \leftarrow \overline{\mathsf{msk}}[s][y_s] \oplus \bigoplus_{i=0}^{s-1} \mathsf{msk}[i][y_i]) \\ \mathsf{msk}[s+1][:] \\ \cdots \\ \mathsf{msk}[S-1][:] \end{pmatrix}$$
3. For each $s \in \{1, \ldots, S-1\}$, server s receives sk_s, and returns $\mathsf{Parity}(\mathsf{Set}(\mathsf{sk}_s)) = \bigoplus_{j \in \mathsf{Set}(\mathsf{sk}_s)} \mathsf{DB}[j]$ to the client.
4. The client receives $p_1, \cdots, p_{S-1}$ from each server $s \in \{1, \ldots, S-1\}$ respectively, and computes $y_0, \ldots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$. Output the answer $v_{y_0} \oplus \bigoplus_{s=1}^{S-1} (p_s \oplus w_{s,y_s})$ — if any of the bits in the xor is the symbol $\perp$, the answer is defined to be $\perp$.
5. If a matching hint was found in Step 1: the client replaces the consumed entry with the following: fetch a new backup hint $\left(\mathsf{msk}, \overline{\mathsf{msk}}, \{v_x\}_x, \{w_{i,x}\}_{i,x}\right)$, and compute $y_0, \cdots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$. Let msk' be equal to msk except that $\mathsf{msk}'[0][y_0]$ is replaced with $\mathsf{idx} \oplus \bigoplus_{i=1}^{S-1} \mathsf{msk}[i][y_i]$, and let v' be equal to v except that v'_{y_0} is replaced with $\perp$. Replace the consumed hint with $\left(\mathsf{msk}', \overline{\mathsf{msk}}, \{v'_x\}_x, \{w_{i,x}\}_{i,x}\right)$.

Figure 11: Improving the online download bandwidth from $n^{\epsilon/S} \cdot \alpha(n)$ to $\alpha(n)$ per server.

Privacy holds in almost the same way as before, since $\overline{\mathsf{msk}}$ is also random from the view of server $1, 2, \cdots, S-1$. Correctness is still guaranteed by observing that $\mathsf{Set}(\mathsf{sk}_s) \triangle \mathsf{Set}(\mathsf{sk}'_{s,y_s}) = \mathsf{Set}(\mathsf{msk}\langle *, \cdots, *, * \setminus \{y_s\}, *, \cdots, * \rangle)$, as explained above.

Special case of $S = 2$. We note that the scheme can be further simplified when $S = 2$. Concretely, note that when $S = 2$, the $\mathsf{sk}'_{i,x}$ has no dependence on msk. Therefore, we can decouple the $\overline{\mathsf{msk}}$ from the hint table. Since each query only consumes a single $\overline{\mathsf{msk}}$ and w, we can fetch them dynamically from the preprocessing server, eliminating the need to store them locally. Furthermore, each time only a single entry in $\overline{\mathsf{msk}}$ and w is used, so it's possible to save the unused entries for the next query. [4]

8.2 Constant-Factor Optimizations

In our implementation, we adopt the following optimizations which allow us to improve the parameters by constant factors.

Merged hint tables. Instead of using independent repetitions as in Figure 11, we merge the hint tables together, and consume $\mathsf{NumCopy}$ hints for each query. This modification preserves perfect privacy, and intuitively gives better correctness guarantee.

In-place background refresh. Naively implementing the pipelined periodic preprocessing would incur a $2\times$ factor in client space. We use an in-place trick to avoid this $2\times$ blowup. Specifically, for each query, $\mathsf{NumCopy}$ hints are consumed where $\mathsf{NumCopy}$ denotes the number of parallel repetitions. Recall that all repetitions now share the same hint table. Therefore, after serving each query, we perform two refreshes:

1. Refresh the consumed hints. We fetch $\mathsf{NumCopy}$ new hints from the preprocessing server, modifies them to contain the current query index, and put them back to the hint table. Basically, we avoid the big backup table by dynamically fetching backup hints from the preprocessing server.

2. In-place background refresh. We maintain a rotating pointer ptr which points to the last background-refreshed hint in the hint table. After serving each query, we fetch another $\mathsf{NumRefresh}$ hints from the preprocessing server and use them to replace the next $\mathsf{NumRefresh}$ hints beginning from ptr. We then move ptr forward by $\mathsf{NumRefresh}$ steps. This way, we can still guarantee that all hints in the hint table are generated in recent rounds, while avoiding the $2\times$ blowup in client space.

8.3 Optimizations for Long Entries when $S = 2$

In real-life applications of PIR (e.g., oblivious DNS [Fea, obl, SCV+20]) typically each entry is not a single bit, but is a wide entry containing W bits. We can choose the row lengths as follows to balance the W-depenent part of the hint (i.e., the parities of entries) and the W-independent part of the hint (i.e., the mPPRS key), and thus minimizing the hint-size.

Specifically, in our scheme when $S = 2$, if we use mPPRS with row lengths $\mathsf{rlen}_0, \mathsf{rlen}_1$ for rows $0, 1$ respectively, then each hint takes $\mathsf{rlen}_0 \cdot W + (\mathsf{rlen}_0 + \mathsf{rlen}_1) \cdot \log n$ bits. The first part $\mathsf{rlen}_0 \cdot W$ stores parities, and the second part $(\mathsf{rlen}_0 + \mathsf{rlen}_1) \cdot \log n$ corresponds to metadata. To minimize the

[4] In our implementation, we still fetched $\overline{\mathsf{msk}}$ and w in their entirety, so correctness for adaptively chosen queries is preserved.

hint size, we choose rlen_0 and rlen_1 to balance the parity part and the metadata part. Recall that we have the constraint that $\mathsf{rlen}_0 \cdot \mathsf{rlen}_1 = n^\epsilon$. To balance the two costs, we may choose $C = \sqrt{W/\log n}$ and $\mathsf{rlen}_0 = n^{\epsilon/2}/C$ and $\mathsf{rlen}_1 = n^{\epsilon/2} \cdot C$. The asymptotic expressions in Table 2 are obtained using this parameter choice. Note that this optimization is not needed for $S > 2$ because after applying optimizations in Section 8.1, all row lengths contribute the same.

8.4 Modifications to the Baseline QuarterPIR

As mentioned in Section 7, we make certain modifications and optimizations to QuarterPIR [GZS24] since the original implementation is intended for the one-server setting. First, we remove the original linear download cost of the preprocessing in QuarterPIR and let the client download the preprocessed hint from the second server, saving the offline communication cost from $\Omega(n)$ to $\tilde{O}_\lambda(\sqrt{n})$. Second, we apply our in-place pipelined preprocessing technique to QuarterPIR, which avoids a $2\times$ blowup in client space.

9 Generalizing Parameters

In previous sections, we stated the results for the case when n is a power of 2 and $(\epsilon/S)\log_2 n$ is integral. When $S = \omega(1)$, rounding n up such that $(\epsilon/S)\log_2 n$ to an integer may incur significant blowup. In this section, we generalize our results to support arbitrary $\epsilon \in (0,1)$ and n sufficiently large w.r.t. ϵ and S. The generalized theorem is formally stated below:

Theorem 9.1 (Our generalized S-server PIR scheme)**.** *Fix an arbitrary $\epsilon \in (0,1)$. Suppose that n is reasonably large such that $1 + \frac{\epsilon}{S} \cdot \log_2 n < n^{\epsilon/2S}$. Let $\alpha(n)$ be any function of n such that $\alpha(n) = \omega(S/\epsilon)$. Then, there exists an information-theoretic S-server PIR scheme with the following performance bounds. The one-time preprocessing incurs $O(\epsilon \cdot n(\log n/S) \cdot \alpha(n))$ server computation and $O(\epsilon \cdot n^{1-\epsilon+\epsilon/S} \log n \cdot \alpha(n))$ communication. Afterwards, every query incurs $O(S \cdot n^{\epsilon/S} \cdot \alpha(n))$ online and offline bandwidth, $O(n^{\epsilon-\epsilon/S} \cdot \alpha(n))$ online server computation, $O(n^\epsilon \cdot \alpha(n))$ offline server computation, $O((\epsilon \cdot n^{1-\epsilon} \log n + S^2 \cdot n^{\epsilon/S}) \cdot \alpha(n))$ online client computation, and $O(S \cdot n^{\epsilon/S} \cdot \alpha(n))$ offline client computation. Further, the client space is bounded by $O(\epsilon \cdot n^{1-\epsilon+\epsilon/S} \log n \cdot \alpha(n))$.*

9.1 Proof of Theorem 9.1

The main idea is to use *variable row lengths*. Henceforth, we use the notations $\mathsf{rlen}_0, \cdots, \mathsf{rlen}_{S-1}$ to denote the row lengths. We prove a rounding lemma, that is, Lemma 9.2, which essentially shows that by allowing variable row lengths, we can always round up n by a constant factor, such that $\prod_{i=0}^{S-1} \mathsf{rlen}_i$ is a divisor of n.

We now generalize our previous (n, S, ϵ)-mPPRS to support variable row lengths, assuming that there exist positive integers $\mathsf{rlen}_0, \cdots, \mathsf{rlen}_{S-1}$, such that $\prod_{i=0}^{S-1} \mathsf{rlen}_i$ is a divisor of n.

Notations. $\mathsf{rlen}_0, \cdots, \mathsf{rlen}_{S-1}$ are defined as above. We define $\mathsf{rlen}_{-1} = n/\prod_{i=0}^{S-1} \mathsf{rlen}_i$, and $B_i = \prod_{j=0}^{i} \mathsf{rlen}_j$ for $i \in [\![S]\!]$. For convenience, let $B_{-1} = 1$. We encode each number $\mathsf{idx} \in [\![n]\!]$ as $(w_{-1}, w_0, \cdots, w_{S-1})$, where $\mathsf{idx} = \sum_{i=-1}^{S-1} w_i B_i$ and $w_i \in [\![\mathsf{rlen}_i]\!]$. Instead of using XOR as before,

we now define the operations $\oplus$ and $\ominus$ as

$$\mathsf{idx} \oplus \mathsf{idx}' := \sum_{i=-1}^{S-1} ((w_i + w'_i) \bmod \mathsf{rlen}_i) \cdot B_i$$

$$\mathsf{idx} \ominus \mathsf{idx}' := \sum_{i=-1}^{S-1} ((w_i - w'_i) \bmod \mathsf{rlen}_i) \cdot B_i$$

where $(w_{-1}, w_0, \cdots, w_{S-1})$ is the encoding of idx and $(w'_{-1}, w'_0, \cdots, w'_{S-1})$ is the encoding of idx'.

Construction. The generalized construction of mPPRS is given in Figure 12, where we mark the important changes in blue. As before, the notation $\mathsf{sk}[i][:]/(x_i^* \leftarrow \$)$ means i-th row of sk but with the x_i^*-th element rerandomized, by resampling the element from the appropriate distribution.

The correctness and privacy of the generalized construction can be proved in a similar way as before. We now analyze the efficiency of the generalized mPPRS:

- Gen: runs in time $O(\sum_{i=0}^{S-1} \mathsf{rlen}_i)$.
- Set: runs in time $O(\prod_{i=0}^{S-1} \mathsf{rlen}_i)$.
- Test: runs in time $O(S)$.
- Puncture: runs in time $O(S \sum_{i=0}^{S-1} \mathsf{rlen}_i)$.
- Subsets: using a naïve approach, the algorithm $\mathsf{Subsets}(s, \mathsf{sk}_s)$ runs in time $O(\mathsf{rlen}_s \prod_{i=s}^{S-1} \mathsf{rlen}_i)$ for outputting all rlen_s sets, each of size $O(\prod_{i=s}^{S-1} \mathsf{rlen}_i)$. However, we point out that any two of those sets differ by only $\prod_{i=s+1}^{S-1} \mathsf{rlen}_i$ elements.

We can use the generalized mPPRS scheme to construct an S-server PIR scheme by plugging this mPPRS into Figure 7. The only modification needed is that in the last step of the query algorithm, when we replace the consumed hint with a backup hint, we set $\mathsf{msk}'[0][y_0]$ to be $\mathsf{idx} \ominus \left(\bigoplus_{i=1}^{S-1} \mathsf{msk}[i][y_i]\right)$. This gives us Theorem 9.1.

9.2 Rounding Lemma

We now use a rounding lemma to get rid of all constraints on n.

Lemma 9.2 (Rounding lemma)**.** *Let n be a positive integer, $0 < \epsilon < 1$, and $S > 1$ be an integer. Then there exists positive integers $\mathsf{rlen}_0, \cdots, \mathsf{rlen}_{S-1}$ such that $\lfloor n^{\epsilon/S} \rfloor \le \mathsf{rlen}_i \le \lfloor n^{\epsilon/S} \rfloor + 1$, and $n^\epsilon \le \prod_{i=0}^{S-1} \mathsf{rlen}_i < 2n^\epsilon$.*

Proof. We first set $\mathsf{rlen}_i = \lfloor n^{\epsilon/S} \rfloor$ for $i \in [\![S-1]\!]$. Apparently, $\prod_{i=0}^{S-1} \mathsf{rlen}_i \le n^\epsilon$. Then, for j from 0 to $S-1$, we increase rlen_j by 1, until $\prod_{i=0}^{S-1} \mathsf{rlen}_i \ge n^\epsilon$. Since the product is multiplied by $1 + 1/\mathsf{rlen}_j$ each time, it is at most $2n^\epsilon$ when it first exceeds n^ϵ. ∎

10 Discussion on Adaptive Correctness

In this section, we discuss how to modify Zelda to achieve *adaptive correctness*. In Section 10.1, we give a generic upgrade to achieve adaptive correctness against $S-1$ colluding servers. In Section 10.2, we explain why without this upgrade, we cannot achieve adaptive correctness when the queries are chosen adaptively based on the preprocessing server's views. The attack given in Section 10.2 can be extended to break the adaptive correctness of prior multi-server client-side preprocessing PIR schemes [RMS24, ISW24, ZPSZ24, GZS24, SWZ24, LP23b].

Generalized construction of mPPRS

$\underline{\mathsf{Gen}(n, S, \epsilon)}$:

- Sample msk as a matrix with S rows and the i-th row has rlen_i elements; for $i \in [\![S]\!]$, $x \in [\![\mathsf{rlen}_i]\!]$, $\mathsf{msk}[i][x]$ is sampled uniformly at random from $\{x \cdot B_i, x \cdot B_i + 1, \cdots, (x+1) \cdot B_i - 1\}$.
- Return msk.

$\underline{\mathsf{Set}(\mathsf{msk})}$: *// This algorithm can also take in a punctured key* sk *as input.*

- Return $\left\{ \bigoplus_{i=0}^{S-1} \mathsf{msk}[i][x_i] : x_i \in [\![|\mathsf{msk}[i][:]|]\!] \right\}$.

$\underline{\mathsf{Test}(\mathsf{msk}, \mathsf{idx})}$:

- For i from $S-1$ down to 0:
  - Let $x_i^* \leftarrow \lfloor \mathsf{idx} / B_i \rfloor$.
  - Let $\mathsf{idx} \leftarrow \mathsf{idx} \ominus \mathsf{msk}[i][x_i^*]$.
- If $\mathsf{idx} = 0$, return 1; otherwise, return 0.

$\underline{\mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})}$:

- For i from $S-1$ down to 0:
  - Let $x_i^* \leftarrow \lfloor \mathsf{idx} / B_i \rfloor$.
  - Let $\mathsf{idx} \leftarrow \mathsf{idx} \ominus \mathsf{msk}[i][x_i^*]$.
- Return $x_0^*, \cdots, x_{S-1}^*$.

$\underline{\mathsf{Puncture}(\mathsf{msk}, \mathsf{idx})}$:

- Let $y_0, \cdots, y_{S-1} \leftarrow \mathsf{GetIndex}(\mathsf{msk}, \mathsf{idx})$.
- Return $\mathsf{sk}_0, \cdots, \mathsf{sk}_{S-1}$ where for $s \in [\![S]\!]$,

$$\mathsf{sk}_s = \begin{pmatrix} \mathsf{msk}[0][y_0] \\ \vdots \\ \mathsf{msk}[s-1][y_{s-1}] \\ \mathsf{msk}[s][:]/(y_s \leftarrow \$) \\ \mathsf{msk}[s+1][:] \\ \vdots \\ \mathsf{msk}[S-1][:] \end{pmatrix}$$

$\underline{\mathsf{Subsets}(s, \mathsf{sk}_s)}$:

- Return $\left\{ \mathsf{Set}\big(\mathsf{sk}_s \langle \underbrace{*, \cdots, *}_{s}, * \setminus \{x_s\}, \underbrace{*, \cdots, *}_{S-s-1} \rangle\big) : x_s \in [\![n^{\epsilon/S}]\!] \right\}$.

Figure 12: Generalized construction of mPPRS

10.1 Upgrade for Adaptive Correctness

Our PIR schemes so far guarantee correctness only if the queries are independent of the view of the preprocessing server, but may be chosen adaptively based on all other server's views. In this section, we describe an upgrade for achieving full adaptive correctness against an arbitrary set of $S-1$ colluding servers.

Theorem 10.1. *Given an S-server client-preproc PIR scheme with the following property:*

- *The PIR scheme satisfies adaptive correctness in the presence of $S-1$ colluding servers as the 0-th server is not corrupted.*
- *For any query sequence, with probability 1, the PIR scheme outputs a correct answer or $\perp$ for every query.*

Then, we can upgrade it to an S-server PIR that satisfies adaptive correctness in the presence of any set of $S-1$ colluding servers, with only a S-factor blowup in all performance metrics.

Proof. The main idea is to run S parallel different instances of the PIR scheme. In each instance we renumber the servers, such that in each instance a different server acts as the 0-th server. In more detail, the upgrade is as follows.

- The client runs S parallel different instances of the PIR scheme. In the i-th instance, the i-th server is renumbered as the 0-th server, and the other servers are renumbered arbitrarily.
- For each query, the client collects the answers from the S instances, and outputs any non-$\perp$ answer.

Privacy is trivially preserved by the above transformation. For adaptive correctness, note that when the adversary only controls $S-1$ servers, there must exist an instance where the adversary does not control the 0-th server, and that instance outputs the correct answer everytime except with negligible probability. All other instances either output the correct answer or $\perp$, so the aggregated answer is correct except with negligible probability. ■

In particular, Theorem 10.1 can be applied to the constructions in Figure 6, Figure 7 and Figure 11.

10.2 Attack Against Adaptive Correctness

In this section, we explain why without the upgrade, we can break the correctness by choosing the queries adaptively based on the preprocessing server's views. Note that the preprocessing server knows the client's hint table after the preprocessing phase. Imagine the query sequence is chosen as follows:

- For each hint $h = (\mathsf{msk}, v)$ that satisfies $\mathsf{idx} \in \mathsf{Set}(\mathsf{msk})$
  - For $\Theta(\mathsf{poly}\log n)$ distinct $\mathsf{idx}' \neq \mathsf{idx}$ such that $\mathsf{idx}' \in \mathsf{Set}(\mathsf{msk})$, query idx'
- Query idx

We argue that the query to idx will produce a correctness error with non-negligible probability. Notice that, each idx' is contained in $O(\mathsf{poly}\log n)$ hints in expectation. So for each idx', the probability that the hint h is consumed is $O(1/\mathsf{poly}\log n)$. Since, the adversary queries $\Theta(\mathsf{poly}\log n)$ different idx's, it is not hard to show that the probability that h is not consumed is negligible in n. Now, using a union bound we can argue that all hints in that were in the table at the beginning that contained idx will be consumed except with negligible probability. Therefore, except with small probability, the query on idx will return $\perp$.

We note that a similar adversary can be constructed against many prior multi-server client-side preprocessing PIR constructions e.g. [ISW24, LP23b, ZPSZ24, RMS24, GZS24, SWZ24] since all of them share the same structure where the preprocessing server knows the hint table of the client after the preprocessing phase, and it can adaptively choose a query sequence to exhaust all hints that contain a particular idx with non-negligible probability, and then query idx.