Plinko: Single-Server PIR with Efficient Updates via Invertible PRFs

Alexander Hoover [2,1], Sarvar Patel[1], Giuseppe Persiano[3,1], and Kevin Yeo[1,4]

[1]Google
[2]University of Chicago
[3]Università di Salerno
[4]Columbia University

Abstract

We study single-server private information retrieval (PIR) where a client wishes to privately retrieve the x-th entry from a database held by a server without revealing the index x. In our work, we focus on PIR with client pre-processing where the client may compute hints during an offline phase. The hints are then leveraged during queries to obtain sub-linear online time. We present *Plinko* that is the first single-server PIR with client pre-processing that obtains optimal trade-offs between client storage and total (client and server) query time for all parameters. Our scheme uses $t = \tilde{O}(n/r)$ query time for any client storage size r. This matches known lower bounds of $r \cdot t = \Omega(n)$ up to logarithmic factors for all parameterizations whereas prior works could only match the lower bound when $r = \tilde{O}(\sqrt{n})$. Moreover, Plinko is also the first *updateable* PIR scheme where an entry can be updated in worst-case $\tilde{O}(1)$ time.

As our main technical tool, we define the notion of an *invertible pseudorandom function* (*iPRF*) that generalizes standard PRFs to be equipped with an efficient inversion algorithm. We present a construction of an iPRF from one-way functions where forward evaluation runs in $\tilde{O}(1)$ time and inversion runs in time linear in the inverse set (output) size. Furthermore, our iPRF construction is the first that remains efficient and secure for arbitrary domain and range sizes (including small domains and ranges). In the context of single-server PIR, we show that iPRFs may be used to construct the first hint set representation where finding a hint containing an entry x may be done in $\tilde{O}(1)$ time.

1 Introduction

Private Information Retrieval (PIR) enables a client to query for elements in a database held by a potentially adversarial server without revealing what the client is trying to access. The notion of a PIR was introduced by Chor et al. [CGKS95] that considered the multi-server setting and the *single-server* setting was first shown to be feasible by Kushilevitz and Ostrovsky [KO97]. This powerful primitive has had a long line of work which focuses on understanding the limitations and variations of PIR for theory and practice. This interest is spurred primarily by the potential application for PIR in modern systems including: advertising [GLM16], certificate transparency [HHCG$^+$23], communication [MOT$^+$11, AS16], device enrollment [Dev], media consumption [GCM$^+$16], password leak check [ALP$^+$21], and publish-subscribe systems [CSM$^+$20].

The original works studied PIR with information-theoretic security and multiple non-colluding servers (see [CGKS95, Amb97, BI01, BIKR02, Yek08, DG15] and references therein). However, more recent work has pushed toward computational PIR and settings with a single server including [KO97, CMS99, BIKM99, KO00, GR05, OS07, ABFK16, ACLS18, PPY18, GH19, ALP$^+$21, MCR21, MW22]. Throughout our work, we will focus on single-server PIR to avoid the stronger trust assumptions between different organizations required in multi-server PIR. Unfortunately, there are known barriers to the computational requirements and

cryptographic assumptions for traditional single-server PIR. Specifically, prior work has shown that single-server PIR requires linear server computation [BIM00], and reducing to sublinear communication implies oblivious transfer [DMO00] (implying that single-server PIR requires public-key assumptions).

PIR with Pre-processing. One approach to circumvent these barriers was server pre-processing that was first considered in [BIM00]. In this setting, the server is able to store pre-process the database with the goal of obtaining sub-linear query time. Several works study this problem [BIM00, BIPW17, PY22] with the recent breakthrough work of Lin et al. [LMW23] showing that poly-logarithmic query time is achievable. However, all these schemes require super-linear server storage to date.

Another line of work, starting from [PPY18], has considered client pre-processing. In an offline phase, clients will compute hints consisting of parities of random subsets of database entries that may be used to help speed-up query times. The breakthrough work of Corrigan-Gibbs and Kogan [CK20] presented a PIR with client pre-processing achieving sub-linear query time. This spawned a long line of work in the area including [SACM21, KC21, CHK22, ZLTS23, LP23b, LP23a, ZPZS24, RMS24, GZS24]. These schemes do not suffer from super-linear storage blowup. To date, the best single-server constructions requiring only a single offline phase obtain query time $t = \tilde{O}(r + n/r)$ if the client has $\tilde{O}(r)$ storage. Curiously, there remains a gap with the best known lower bounds of $t = \Omega(n/r)$ [Yeo23] when client storage r is large. For example, when $r = O(n^{2/3})$, the best constructions require $\tilde{O}(n^{2/3})$ time whereas the lower bound states that only $\Omega(n^{1/3})$ time is necessary. This leads to the following:

Does there exist a single-server PIR construction that obtains optimal trade-offs between client storage and query time for all parameters?

Another difficulty of PIR with pre-processing is the ability to handle updates to database entries. Several works [ZPZS24, LMW23] considered the static-to-dynamic transform of Bentley and Saxe [BS80]. While these incur amortized logarithmic update time, the worst-case update time will be linear in the database size. For PIR with client pre-processing, there is a straightforward approach where the server may simply send updated entries to clients. Then, clients need to update their hints (parities of database entries) accordingly. However, this requires $\tilde{O}(r)$ client time to search the hints. In contrast, PIR without pre-processing can handle updates in worst-case constant time by the server simply updating the according entry locally. A natural problem one can consider is:

Does there exist a single-server PIR construction that handles a database entry update in worst-case $\tilde{O}(1)$ time?

Often in PIR applications, the client is resource constrained in space and time; however, these prior works do not enable a trade-off between these two resources. Instead, a client's time burden to query with, search through, and update their hints *increases* as they store more hints. In practice though, a client may want very short query times and be willing to pay more in space for this. Prior works allow the client to increase their storage to improve server query time, but increase the resource-constrained client query time, cutting against the client's desire (or only being favorable up to a certain point). Ours is the first scheme to *decrease* the client's time burden as their storage increases, for *any amount of storage*, which is essential to achieve short query times.

1.1 Our Contributions

In this paper, we answer in the affirmative for both of these questions. We improve the state-of-the-art in single-server PIR schemes. We construct the first single-server PIR scheme, called Plinko, that achieves an optimal trade-off between client storage and total computation time across the entire curve. For example, Plinko obtains optimal time for any parameterization of client storage size. In the process of building Plinko, we define and construct a novel primitive called an *invertible PRF* (iPRF). As a natural consequence of our method, Plinko provides a method to implement updates more efficiently and more simply than methods from prior work.

Scheme Name	Crypto Assumption	Client Storage	Query Time	Query Comm.
CHK [CHK22]	LWE	r	$r + n/r$	n/r
ZLTS [ZLTS23]	LWE	r	$r + n/r$	$\text{polylog}(n)$
LP [LP23a]	LWE	r	$r + n/r$	$\text{polylog}(n)$
Piano [ZPZS24]	OWF	r	$r + n/r$	n/r
RMS [RMS24]	OWF	r	$r + n/r$	n/r
GZS [GZS24]	OWF	r	$r + n/r$	$\sqrt{n/r}$
Lower Bound [Yeo23]	-	r	n/r	-
Plinko	OWF	r	n/r	n/r

Figure 1: Comparison of the amortized query time and query communication for existing single-server offline/online PIR schemes, ignoring polylog factors. We give a client r bits of hint storage for a database of size n. See Figure 6 for further comparison.

Scheme Name	Crypto Assumption	Update Time		Update Comm.	
		Worst-case	Amortized	Worst-case	Amortized
CHK [CHK22]	LWE	$\sqrt{n}$	$\sqrt{n}$	$\log n$	$\log n$
ZLTS [ZLTS23]	LWE	$\sqrt{n}$	$\sqrt{n}$	$\log n$	$\log n$
LP [LP23a]	LWE	$\sqrt{n}$	$\sqrt{n}$	$\log n$	$\log n$
Piano [ZPZS24]	OWF	n	$\text{polylog}(n)$	n	$\text{polylog}(n)$
RMS [RMS24]	OWF	$\sqrt{n}$	$\sqrt{n}$	$\log n$	$\log n$
GZS [GZS24]	OWF	$\sqrt{n}$	$\sqrt{n}$	$\log n$	$\log n$
Plinko	OWF	$\text{polylog}(n)$	$\text{polylog}(n)$	$\log n$	$\log n$

Figure 2: Comparison of existing single-server offline/online PIR schemes, for a client querying a size n database. For simplicity, we set client storage to be $r = \sqrt{n}$ for all schemes. Query times and communication are amortized. For a more complete comparison, see Figure 6.

Invertible PRFs. Our first contribution is a new cryptographic primitive that generalizes pseudorandom functions that we denote as an invertible PRF (iPRF). To our knowledge, we are the first to consider PRFs with arbitrary domains and ranges that are efficiently invertible. This allows someone with the key to efficiently enumerate the inputs which map to any specific output. For security, an iPRF should appear indistinguishable from a random function to any computational adversary without the key.

The most similar cryptographic primitive previously studied are pseudorandom permutations (PRP) that also enable inversion. However, it is quite clear that (pseudorandom) random permutations are distinguishable from (pseudorandom) random functions. For example, random permutations are always bijections whereas random functions will not be a bijection with overwhelming probability. Another approach may be to try and build a PRF using a truncated PRP where one aims to build a PRF using a PRP where each output is truncated. Tight bounds for distinguishing a truncated PRP from a random function are well known [Sta78, HWKS98, GG15, GGM18, Men19, GG21]. For most domain and range sizes (including settings for our PIR applications), an adversary will successfully distinguish between a truncated PRP and PRF. Therefore, we are unaware of any straightforward way to build iPRFs from (truncated) PRPs. Finally, we note invertible PRFs were defined in another context in [BKW17] where only *injective* random functions were considered (that is also not appropriate for our PIR application).

Our contributions for iPRFs are twofold. We formally define iPRFs for arbitrary domains and ranges.

Then, we construct the first iPRF which is secure even for small domain and range sizes (where it is computationally feasible for an adversary to enumerate the entire input and output space). The small domain and range property will be critical for our PIR application. Our construction composes a (small-domain) PRP together with a new sampling technique to preserve a uniform distribution for every input while allowing efficient inversion. Furthermore, our techniques allows us to construct efficient iPRFs for domains and ranges of any sizes even if they are drastically different. In terms of efficiency, our iPRF construction enables evaluation in $\tilde{O}(1)$ time and inversion in time linear (ignoring logarithmic factors) in the output size, which is the size of the inverse set. Furthermore, our construction is built only assuming the existence of one-way functions. This is in contrast to some other PRF-related primitives used to build PIR in other contexts, which rely on stronger assumptions such as learning with errors. We point readers to Section 4 for our iPRF construction.

New Single-Server PIR with Optimal Trade-off Curve. Recent lower bounds for traditional PIR have shown that any PIR with pre-processing scheme with client storage r and query time t must obey $r \cdot t = \Omega(n)$ [CK20, CHK22, Yeo23]. Recent work have matched this bound (up to poly-logarithmic factors), but only for certain parameterizations of client storage. In these prior constructions, the server requires $\tilde{O}(n/r)$ time. However, the query algorithms of prior work requires the client to perform a linear pass through $O(r)$ hints stored in client storage, so that the total runtime for a query is $t = \tilde{O}(r + (n/r))$. In other words, these prior works obtain trade-offs of $r \cdot t = \tilde{O}(r^2 + n)$. These constructions match the lower bound for small client storage sizes of $r = O(\sqrt{n})$, but are not optimal for larger client storage sizes. For example, when $r = O(n^{2/3})$, prior works require query times of $t = \tilde{O}(n^{2/3})$ whereas the lower bound specifies query time only need be as large as $\Omega(n^{1/3})$.

We address this deficiency with our new single-server PIR scheme, Plinko, that is actually a modification of two recently proposed PIR schemes [ZPZS24, RMS24]. Both schemes rely on using PRFs to generate uniformly random offsets, that serve as compressed representations of sets. By substituting the PRFs with iPRFs in the constructions and maintaining the client's memory carefully, we effectively preserve all of the functionality (correctness and privacy) while dramatically improving client query time for many parameter choices.

Using iPRFs, our Plinko construction avoids the linear pass over the $\tilde{O}(r)$ hints in client storage. Instead, the client can search for the relevant hint in $\tilde{O}(1)$ time during queries. Furthermore, this improvement does not come at any cost to the server time either. As a result, we achieve total query time of $t = \tilde{O}(n/r)$ and trade-off of $r \cdot t = \tilde{O}(n)$ matching the lower bound (up to poly-logarithmic factors). In other words, Plinko obtains optimal query time $t = \tilde{O}(n/r)$ for any choice of client storage r. Even more interesting is that we achieve this trade-off without the use of public-key cryptography, because Plinko only requires the existence of one-way functions. In contrast, all prior work (even when assuming public-key cryptography) are unable to obtain optimal query times for all client storage sizes. See Figure 1 for a detailed comparison with prior works.

In Section 5, we present our Plinko scheme, which is based on the scheme from [RMS24]. We show that our technique of utilizing iPRFs is flexible by also presenting another version of Plinko from [ZPZS24]. Note

Efficient Updates. Yet another benefit of using invertible PRFs is that our scheme lends itself to a simple and efficient method to support dynamic databases. Prior work for updateable single-server PIR has taken two main approaches.

First, there are works [BS80, ZPZS24, LMW23] relying on the Bentley-Saxe transform [BS80] to support database modifications. This involves a server maintaining geometrically growing PIR instances and rebuilding them on a schedule as updates happen. This method has logarithmic amortized communication and runtime, but the worst-case update will require linear communication and time. Additionally, this requires the client and server to maintain more complicated data structures.

The more simple approach just has the server store a changelog and push out updates to clients. However, this method either requires the client to store the changes locally or to update their hints. Storing changes is impractical in heavy-write workloads, because a client with only $r \ll n$ bits of storage will need to fold

the changes into their hints or rerun the offline phase after r updates, which amortizes to either $\tilde{O}(n/r)$ or $\tilde{O}(r)$ computation per update.

Until our work, whenever a client needed to update their hints after a database element changed, they would have to perform a linear $\tilde{O}(r)$ pass over their hints, checking if each hint was effected by the change and updating the hint if it were the case. Fortunately, invertible PRFs get around this linear pass, when the client maintains their hints in the appropriate data structure. Specifically, a client can essentially just invert the PRFs at the updated index and receive an enumeration of every hint that needs to be updated. This means, with just one-way, server-to-client, and worst-case logarithmic time and communication, the client can perform their update! We point readers to Figure 2 for more comparisons.

We note that independent, concurrent work [LP24, FLLP24] additionally achieves efficient updates, but uses different techniques from us. Their work either uses 2 servers [LP24] or public-key operations [FLLP24].

1.2 Related Works

PIR without Pre-processing. Beimel et al. [BIM00] show any PIR scheme (even multi-server) requires linear query time without pre-processing. Therefore, the majority of PIR without pre-processing focuses on reducing communication as well as concrete computational costs for practical applications. Earlier single-server PIR works considered constructions using various number-theoretic assumptions including [KO97, CMS99, GR05]. More recent works build single-server PIR using lattice-based assumptions such as [ABFK16, ACLS18, GH19, PT20, ALP+21, MCR21, MW22, HHCG+23]. Another line of single-server PIR work also consider supporting keyword queries [MK22, PSY23]. A long line of work has also studied multi-server PIR with information-theoretic privacy including [CGKS95, Amb97, BI01, BIKR02, Yek08] culminating in the work by Dvir and Gopi [DG15] showing that sub-polynomial communication is sufficient for two-server PIR. Two-server PIR has also been studied in computational setting using function secret sharing [BGI16].

PIR with Pre-processing. To obtain sub-linear query times, prior works have studied PIR with pre-processing. The first construction by Beimel et al. [BIM00] presented multi-server PIR schemes with query time $t = O(n^{1/2+\epsilon})$ but required servers to store pre-processing on the size of $r = n^{1+O(1/\epsilon)}$. This primitive has also been studied under the notion of public-key doubly-efficient PIR [BIPW17] from obfuscation. A very recent breakthrough work by Lin et al. [LMW23] presented a single-server PIR with $t = \text{polylog}(n)$ query time and server pre-processing of size $r = n^{1+O(1)}$. The highest lower bounds show that $r \cdot t = \Omega(n \log n)$ [PY22].

Another line of work considers the case where the client may also store the pre-processing hidden from the view of the adversarial server. This has been studied under the notion of private-key doubly-efficient PIR [BIPW17, CHR17] using new assumptions based on permuted puzzles [BHW19]. Patel et al. [PPY18] presented a single-server PIR with client pre-processing with sub-linear public-key operations from standard assumptions (also studied in [MCR21]). The breakthrough work of Corrigan-Gibbs and Kogan [CK20] presented PIR schemes with total sub-linear time. This spawned a large number of recent works obtaining sub-linear query time constructions including [SACM21, KC21, CHK22, ZLTS23, Yeo23, LP23b, LP23a, ZPZS24, RMS24, GZS24]. Several works have studied lower bounds in this setting [CK20, CHK22, Yeo23] where the highest lower bound is $r \cdot t = \Omega(n)$ where t is the online query time and r is the client storage.

PRFs, PRPs, and Truncated PRPs. Pseudorandom functions (PRFs) were originally introduced by Goldreich et al. [GGM84]. They are an essential building block of cryptography and can be built only assuming the existence of one-way functions. There have also been many variations of PRFs that are used in many different context and rely on different assumptions.

However, the primary variant of PRFs which allows a user with the secret key to invert are pseudorandom permutations (PRPs) [LR88]. The distinguishing advantage between a random permutation and random function is well understood through various switching lemmas [IR89, HWKS98, BR06, Din20] and matching distinguishing attacks. In particular, it is easy for an adversary to distinguish between a random function and random permutation when the output space is small. To the authors' knowledge, the only other variant of PRFs that is efficiently invertible was introduced by Boneh et al. [BKW17], but this concept is for a

random injective function which is insufficient our application in PIR since evaluations at different points never collide, similar to a PRP.[1]

Prior work has constructed something between a random permutation and a random function called a truncated permutation, which can be instantiated with a PRP. There is a long line of work understanding the ability for one to distinguish between a random function and a truncated permutation [Sta78, HWKS98, GG15, GGM18, Men19]. This work culminates with the recent work by Gilboa and Gueron [GG21], which summarizes attacks and proves them optimal. These attacks, like those of a PRP, present a distinguisher with large advantage when the output space is small.

Preimage Sampleable Functions. Although the name sounds similar, preimage sampleable (trapdoor) functions (PSFs) [GPV08] are a distinct primitive from our new invertible pseudorandom function. In particular, PSFs do not care about pseudorandomness and have some trapdoor that allows the inversion, and therefore requires public-key operations. In contrast, our new primitive and construction is pseudorandom, symmetric key, and only relies on the existence of one-way functions. Additionally, we can recover the entire pre-image with our inversion operations, not just sample from it.

2 Technical Overview

In this section, we present a simplified Offline-Online PIR (OO-PIR) scheme (similar to [ZPZS24, RMS24]) to highlight an inefficiency persistent in all Offline-Online PIR schemes to date. We then (informally) introduce a new primitive called an *invertible PRF (iPRF)* and sketch how we can construct them efficiently. Finally, we show how we can modify the OO-PIR scheme to use our new primitive to improve over the original's efficiency.

A Simple Offline-Online PIR. We recall, at a high level, the framework for building OO-PIR used in multiple recent works [ZPZS24, RMS24]. These schemes follows similar ideas from other prior work such as [CK20, CHK22], but simplifies some of the ideas to achieve more efficient queries. Both recent works [ZPZS24, RMS24] stream the database in the offline phase. In Section 5.2, we discuss that any OO-PIR assuming only one-way functions (and query communication is sub-linear) must have linear offline phase communication meaning database streaming is optimal.

Suppose the client has storage of r bits. The offline phase has the client retrieve the parity of the database entries for $w = r$ random sets of size n/r. Throughout our work, we will commonly refer to these parities as hints. To sample these random sets, we think of the database as being separated into n/r "blocks" each of size $w = r$. A random set is represented by n/r ordered offsets $o_1, \ldots, o_{n/r}$ that are each uniformly chosen from $\{0, \ldots, w-1\}$. Then, the set consists of the indices $\{j \cdot w + o_j\}_{j \in [n/r]}$. To find the parity of these sets' database entries in the offline phase, the client will stream the database, one block at a time, and locally compute the parities. Notice that, as described, the client has to store more than n bits of information for these random sets. However, the client may use a pseudorandom function to succinctly store these sets.

In the online phase to query at $x \in [n]$, the client will find a hint set containing x. Suppose the found hint is $o_1, \ldots, o_{n/r}$ where $i \cdot w + o_i = x$ (that is, x appears in i-th block). Next, the client removes the i-th offset corresponding to x from the list of offsets, and sends the remaining $n/r - 1$ offsets to the server. The server, not knowing that the i-th block's offset was removed, will try all possible n/r options for the missing offsets and compute the parity of the corresponding $n/r - 1$ entries. For concreteness, the server's j-th parity will assume the j-th block was removed and compute the parity assuming the $n/r - 1$ offsets are for the other $n/r - 1$ blocks. We note prior work [ZPZS24] showed that these n/r parities may be computed $\tilde{O}(n/r)$ time by the server. Finally, the server sends the n/r parities back to the client. Upon receiving these parities, the client will use the parity where the i-th block was removed. Then, it can take the difference in the received

[1] This primitive is often called an "invertible pseudorandom function," overlapping with the our new primitive. However, we believe our definition is better suited to the name, and that this previous variant is better named "pseudorandom invertible/injective function."

parity from the server and client stored parity (hint) from the offline phase to compute the value of x in the database. As a disclaimer, we presented the query algorithm from Piano [ZPZS24] whereas Mughees et al. [RMS24] take a slightly different approach. Nevertheless, our improvements can apply to either scheme.

Hint Searching. Unfortunately one drawback of the above scheme is elided in the brief description that the client will "find a hint set containing x," which itself will depend on the set representation. When using PRFs, the client can just run the PRF with the relevant offset for x to test for membership in constant time. But, this will still require the client to perform a linear pass through their hints! And in expectation, this requires the client to look through w hints before they find one containing their query.

Other OO-PIR schemes that use this kind of paradigm represent their sets in different ways, using for example privately puncturable PRFs [SACM21, ZLTS23]. These different representations provide an array of theoretical or practical benefits and trade-offs in communication/time/etc. However, most proposals rely on building representations with (near) constant-time *set membership tests*, and the schemes require that the client scan through their hints running this set membership on the order of $\tilde{O}(r)$ times. Once it finds the hint, either the client or server must enumerate the n/r set elements which must take $\tilde{O}(n/r)$ time. Therefore, the total query time is $\tilde{O}(r + n/r)$ across both the client and the server. Note, even if the client enumerates the set, the server's time is always $\tilde{O}(n/r)$, meaning the client's time is the bottleneck. Our work is the first to overcome this bottleneck by proposing a set representation which has a (near) constant-time *hint searching* algorithm that will reduce the total query time to $\tilde{O}(n/r)$ by improving the client's hint searching time to $\tilde{O}(1)$. This allows our new OO-PIR scheme to obtain optimal space-time trade-offs for all parameters as well as efficient updates as we will show later.

Invertible PRFs. The primary tool we use to build efficient hint searching is a new primitive we call an *invertible PRF (iPRF)*. All iPRFs are PRFs, so, to an adversary who does not know k, evaluating $\mathsf{iF.F}(k, \cdot)$ appears as a random function. However, iPRFs also have an *efficient* inversion algorithm $\mathsf{iF.F}^{-1}(k, \cdot)$, which returns the pre-image of the given input and whose runtime depends only on the size of the pre-image. This new primitive syntactically makes sense as a PRF and (for our application) only requires traditional PRF security, but we define a stronger security game which requires the inversions to appear as random as well.

In Section 4, we show how to build an iPRF from one-way functions (the minimal assumption for iPRF existence). We model a random function from $[n] \to [m]$ as throwing n labeled balls into m labelled bins. At a high level, our iPRF follows two main steps. We first apply a PRP to the input, which is analogous to randomly permuting n balls. Then, we give a new method to throw n ordered balls into m ordered bins (with a PRG). Our method also gives near constant-time algorithms to determine which bin a particular ball was thrown into and to enumerate all the balls in a particular bin.

One key insight for our sampling technique is that we can find out how many balls land in any bin though a series of binary choices. We imagine a binary tree sitting above the bins, and starting at the root, we decide for each ball whether it will go left or right. By repeating these binary left/right decisions, each ball can find its appropriate bin after only $\log m$ coin flips. In order to track the *number of balls* that land in each bin then, we can sample binomials random variables at each node of the tree. So, at the root, we decide how many of the n balls go left or right (i.e. sample from $\mathsf{Binomial}(n, 1/2)$). Then, based on that outcome, we can follow down any path to a specific bin keeping track of the number of balls that have fallen along that path.[2] In addition to finding the number of balls in any bin, we can follow down *any ball i* to find which bin it landed in. We just canonically say, if k balls go left, we can just imagine the balls labeled from 1 to k were the ones that went that way.

This ability to find all the balls in a given bin and which bin a given ball landed in is essential to giving our iPRF efficient computation in both the forward and reverse directions. In fact, with access to a PRP, the evaluation and inversion of the iPRF simply amounts to composing our sampler with the PRP in the appropriate orders.

[2]This insight, which is critical to our efficient algorithm, is the origin of our scheme's name "Plinko," a game where a ball falls on subsequent pegs and will either go left or right at each level.

Fast Hint Searching in OO-PIR. Next, we show how invertible PRFs can be used to give an efficient *hint searching* algorithm without sacrificing the cost of efficient set membership and set enumeration algorithms (which are also important to other parts of OO-PIR schemes). Unfortunately, iPRFs do not generically improve all OO-PIR, because many rely on specific puncturable properties of their PRFs, which are not provided by iPRFs. However, we can apply iPRFs to two recently proposed schemes, Piano and RMS [ZPZS24, RMS24].

Continuing with our sketch of the framework used in [ZPZS24, RMS24], we can replace the calls to the PRF with calls to an iPRFs. For each of the n/r blocks, we pick a different iPRF key, and for hint h, we can compute the offset for block i with a forward evaluation $\mathsf{iF.F}(k_i, h)$, which appears uniformly random. Note, the n/r keys for each of the iPRFs can also be pseudorandomly generated using a PRF. Therefore, this only requires storing a single PRF key.

Now, to find a hint that contains an element x, which is in block i with offset o, all we have to do is run $\mathsf{iF.F}^{-1}(k_i, o)$! This will give us a set of hint indices that corresponds exactly with the hints containing x. We show that, if the hints are stored in an appropriate data structure, this gives a (near) constant-time way to find a hint for querying. As a result, the client's query time becomes $\tilde{O}(n/r)$ with the main cost coming from enumerating the n/r offsets in the found hint set. In contrast, prior constructions also needed to scan through all $w = r$ hints requiring $\tilde{O}(r + n/r)$ client time.

Improved Space-Time Trade-offs. Our work is the first to achieve an *optimal trade-off* for all parameter choices matching recent lower bounds against PIR with client pre-processing (ignoring logarithmic factors). In particular, [Yeo23] showed that there is an inverse relationship between the size of the client's hint and the server run-time of query processing. For a database of n items, it is known that $r \cdot t = \Omega(n)$, where r is the client's hint storage size and t is the time of the server. Notice that this immediately implies a lower bound for the *total* query time, including both the client and server. Prior constructions have matched this bound for *server time*, but have sacrificed the run-time of the client when the client has large storage sizes, r. To date, all constructions require a client to perform a linear pass over their own hint before issuing a query. This means, for example, if a client has $r = w = n^{2/3}$ bits of storage, the runtime of the client will be $\tilde{O}(n^{2/3})$. So, even though the server runtime may be $\tilde{O}(n/r) = \tilde{O}(n^{1/3})$, the total time to process a query still grows with the client's storage size and remains $\tilde{O}(n^{2/3})$. Notice, that lower bounds show that the total time only need be $\Omega(n/r) = \Omega(n^{1/3})$.

Fortunately, an efficient hint searching algorithm circumvents this barrier in OO-PIR constructions. By finding hints directly without a linear pass over the entire storage, our clients can run in $\tilde{O}(n/r)$ time, which is significantly more efficient for a large number of hints. This gives us the first OO-PIR scheme with total query time $\tilde{O}(n/r)$ that obtains the optimal $r \cdot t = \tilde{O}(n)$ trade-off for *any* choice of client storage size r.

Efficient Database Updates. The other primary benefit of our hint search algorithm allows clients to update their hints more efficiently. One simple way to design PIR with updates is to require the server to send the changes to the database each time a client connects (via something like a changelog). However, if an index changes and the client does not want to recompute all of their hints, they need to find all of their hint sets that contain the index, so that they can update the stored parity for that set. Unfortunately, schemes as written currently would require the client to pass through each hint set and check the membership for the updated element.

Our efficient hint searching algorithm has the benefit of not only giving one candidate hint but *all* hints that contain the specified index. So, when a client is told to perform an update, they can just do a single call to the iPRF in use, find the hints containing the index, and then update those parities immediately. This amounts to only an update functionality that requires only logarithmic communication, logarithmic time for the client, and is conceptually simple.

This is a large improvement over alternative methods to reduce update time. Specifically, prior works have suggested using a type of Bentley-Saxe transform to build a geometrically growing group of PIR [BS80, ZPZS24, LMW23]. This gives an asymptotically logarithmic amortized update time, which is comparable to our schemes in this paper. However, it also has amortized logarithmic communication for updates, and in the

worst case, this method requires $O(n)$ time and communication. And, as far as the authors are aware, there are no generic de-amortization results that can work in the PIR setting. For example, solutions for ORAM may be able to de-amortize the server's rebuilding of different levels. But, they give no generic way for the client to de-amortize the preprocessing phase that is performed after the data structure is built. Moreover, such a transform would not improve over our approach. Finally, the growing PIR schemes require the client to maintain independent sets of hints for each and query in a more complicated way, which leads to more complicated data management. In contrast, our method provides a very simple and efficient way to perform and manage hints for the client and the server even in the presence of updates.

Additions and Deletions. Our work focuses only on updating specific entries of the database. We note standard PIR typically considers a fixed-size array with no sense of adding/deleting but only updating array entries. One can also consider the more general keyword-PIR (key-value) where it makes sense to add/delete new entries. A standard cuckoo-hashing technique (see [ZPZS24, ALP+21]) can reduce keyword-PIR to standard PIR. In this case, add/delete/update translates to updating a couple array entries. This standard approach can also be applied to extend Plinko to a keyword-PIR enabling additions/deletions.

Another approach is appending array entries following [MZRA22]. Plinko can use the same general approach: initialize with a larger size of zeroed-out entries and "insert" into the zero locations. To delete, we can overwrite elements with random information or a canonical "deleted" value.

3 Preliminaries

Through the paper we use $[n]$ to denote the set $\{0, 1, \ldots, n-1\}$. For a function $f : \mathcal{D} \to \mathcal{R}$, we define $f^{-1} : \mathcal{R} \to 2^{\mathcal{D}}$ as $f^{-1}(y) = \{x \in \mathcal{D} : f(x) = y\}$. When f is a permutation on $\mathcal{D}$, then $f^{-1} : \mathcal{D} \to \mathcal{D}$ since the pre-image of every $y \in \mathcal{D}$ is unique. We also use asymptotic notation to describe the behavior of variables.

Asymptotic Notation. We use notation such as $O(\cdot)$ and $\Omega(\cdot)$ in the standard way. However, we slightly abuse the notation for $\tilde{O}(\cdot)$. In particular, we use this throughout to he paper to denote that we ignore multiplicative factors which are poly-logarithmic in the main variables we are considering, which is clear from context. So, for example, we may say algorithm running for PIR on a database of size n runs in $\tilde{O}(1)$ time, which is equivalent polylog(n) time. This differs from some conventional use, which would ignore poly-logarithmic factors in the input to $\tilde{O}(\cdot)$. We also treat the security parameter λ as a constant for most of our asymptotic notation.

Distributions. In some of our proofs and constructions we use the Bernoulli ($\mathsf{Bernoulli}$), binomial ($\mathsf{Binomial}$), and multinomial (MN) distributions. We also occasionally treat these as randomized functions which take parameters as input and have an output distributed according to the distribution queried. We also sometimes see these functions with some randomness r to turn it into a deterministic function. For example, $\mathsf{Binomial}(n, p; r)$ will always return the same value, but over the random choice of r with sufficient entropy, the output will be distributed according to a binomial. The MN distribution is parameterized by n and m, so that $\mathsf{MN}(n, m)$ is a tuple of m random variables which are each distributed according to $\mathsf{Binomial}(n, 1/m)$ conditioned on them all summing to exactly n.

Pseudorandom Functions and Permutations. Next we recall a standard definition of pseudorandom functions (PRFs) in Definition 3.1. This definition captures a class of functions that are indistinguishable from random functions over the choice of the random key. Notice that for small domains that are polynomial in the security parameter, we could equivalently give the adversary the entire table of function values.

Definition 3.1. *A* pseudorandom function *(PRF) [GGM84] from $\mathcal{D}$ to $\mathcal{R}$ with keyspace $\mathcal{K}$ is pair of efficiently computable functions: a randomized key generation function* $\mathsf{Gen} : \{0,1\}^* \to \mathcal{K}$ *and a deterministic*

function $\mathsf{F} : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$. *For an adversary* $\mathcal{A}$ *and PRF* F, *we define*

$$\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\lambda, \mathcal{A}) = \left| \Pr_{k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)} [\mathcal{A}^{\mathsf{F}_k(\cdot)}(1^\lambda) = 1] - \Pr_{\mathsf{R} \xleftarrow{\$} \mathsf{Func}[\mathcal{D},\mathcal{R}]} [\mathcal{A}^{\mathsf{R}(\cdot)}(1^\lambda) = 1] \right|.$$

We call a PRF F *secure if for all efficient* $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{prf}}_{\mathsf{F}}(\lambda, \mathcal{A}) \leq \mathsf{negl}(\lambda)$.

We additionally recall the definition of a pseudorandom permutation (PRP) in Definition 3.2. This captures a class of permutations which are indistinguishable from a random permutation. Notice that for small domains that are polynomial in the security parameter, we could equivalently give the adversary the entire table of permutation values.

Definition 3.2. *A* pseudorandom permutation *(PRP) [LR88] over* $\mathcal{D}$ *with keyspace* $\mathcal{K}$ *is triple of efficiently computable functions: a randomized key generation function* $\mathsf{Gen} : \{0,1\}^* \to \mathcal{K}$, *a deterministic function* $\mathsf{P} : \mathcal{K} \times \mathcal{D} \to \mathcal{D}$, *and a deterministic function* $\mathsf{P}^{-1} : \mathcal{K} \times \mathcal{D} \to \mathcal{D}$. *For an adversary* $\mathcal{A}$ *and PRF* P, *we define*

$$\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{P}}(\lambda, \mathcal{A}) = \left| \Pr_{k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)} [\mathcal{A}^{\mathsf{P}_k(\cdot), \mathsf{P}_k^{-1}(\cdot)}(1^\lambda) = 1] - \Pr_{\pi \xleftarrow{\$} \mathsf{Perm}[\mathcal{D}]} [\mathcal{A}^{\pi(\cdot), \pi^{-1}(\cdot)}(1^\lambda) = 1] \right|.$$

We call a PRP P *secure if for all efficient* $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{prp}}_{\mathsf{P}}(\lambda, \mathcal{A}) \leq \mathsf{negl}(\lambda)$.

We do not provide a formal definition, but we sometimes refer to a *truncated permutation*. This is computed by using a PRP on the domain $\{0,1\}^n$ and then truncating the output to the first $m < n$ bits. Over large domains, this can be used to approximate a random function. This type of function has the benefit that it can be inverted efficiently. For any bit-string in $\{0,1\}^m$, one can append every possible string in $\{0,1\}^{n-m}$ and run the PRP inversion. Unfortunately, for our purposes in this paper, these truncated permutations are insufficient, because there exists attacks due to [GG15, GGM18] which have large advantage when the PRP domain is small.

4 Invertible PRFs

In this section, we generalize the notion of pseudorandom functions (PRFs) [GGM84] and define what we call invertible pseudorandom functions (iPRFs). Prior work [BKW17] has defined a primitive by the same name, but our definitions and constructions differ from theirs. In particular, the prior work only considers invertibility for injective random functions where each output element has at most one inverse In contrast, we consider arbitrary random functions without this restriction (that will be necessary in our later PIR applications).

First, we give a formal definition for an invertible PRF. We require an iPRF to have a generation method as well as algorithms for forwar and backward evaluation. We also introduce a security definition for iPRFs which gives an adversary access to both the forward and inverse oracle. If we just take the generation and forward functions for a secure iPRF, this will simply be a PRF.

Definition 4.1. *An* invertible pseudorandom function *(iPRF) from domain* $\mathcal{D}$ *to range* $\mathcal{R}$ *with keyspace* $\mathcal{K}$ *is a triple of efficiently computable functions: a randomized key generation function* $\mathsf{Gen} : \{0,1\}^* \to \mathcal{K}$, *a deterministic function* $\mathsf{F} : \mathcal{K} \times \mathcal{D} \to \mathcal{R}$, *and a deterministic function* $\mathsf{F}^{-1} : \mathcal{K} \times \mathcal{R} \to 2^{\mathcal{D}}$. *We say an iPRF* iF *is* correct *if, for all* $y \in \mathcal{R}$,

$$\Pr_{k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)} [\mathsf{F}_k^{-1}(y) \neq \{x \in \mathcal{D} \mid \mathsf{F}_k(x) = y\}] \leq \mathsf{negl}(\lambda).$$

For an adversary $\mathcal{A}$ *and iPRF* iF, *we define the adversarial advantage as*

$$\mathbf{Adv}^{\mathrm{iprf}}_{\mathsf{iF}}(\lambda, \mathcal{A}) = \left| \Pr_{k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)} [\mathcal{A}^{\mathsf{F}_k(\cdot), \mathsf{F}_k^{-1}(\cdot)}(1^\lambda) = 1] - \Pr_{\mathsf{R} \xleftarrow{\$} \mathsf{Func}[\mathcal{D},\mathcal{R}]} [\mathcal{A}^{\mathsf{R}(\cdot), \mathsf{R}^{-1}(\cdot)}(1^\lambda) = 1] \right|.$$

We call a iPRF iF *secure if for all efficient* $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{iprf}}_{\mathsf{iF}}(\lambda, \mathcal{A}) \leq \mathsf{negl}(\lambda)$.

We also observe that this definition is strictly stronger than the PRF security definition. For example, we can construct an iPRF which has a secure forward function but which is not a fully secure iPRF as specified by Definition 4.1, at least in the case of large domains, when n is much larger than the adversary's runtime. We can build a pathological iPRF iF' built from an underlying iPRF iF which is defined as $\mathsf{iF}'.\mathsf{F}_k(x) = \mathsf{iF}.\mathsf{F}_k(x)$ for every x except that $\mathsf{iF}'.\mathsf{F}_k(k) = 0$. This is indistinguishable from the original iF when n is large (because it is unlikely that any adversary will query k) and therefore it is indistinguishable from a random function in the forward direction. However, iF' is clearly not a fully secure iPRF because an adversary could query the inverse oracle on 0 to get a small list of candidates (assuming the range is about as large as or larger than the range) for the key, which it could then check against other evaluations.

In order to construct our own iF, we only require two tools, which can both be built out of one-way functions. The first is a pseudorandom permutation Definition 3.2, which has been well studied. Our other tool is new to this work. It is a primitive we call a pseudorandom multinomial sampler (PMNS) in Definition 4.2. Intuitively, we use a PMNS in our construction of an iPRF to efficiently sample a pre-image distribution that matches that of a random function.

4.1 Pseudorandom Multinomial Samplers

For integers n and m, the (n, m)-*Multinomial Distribution* $\mathsf{MN}(n, m)$ is the distribution over the sequences $(l_0, \ldots, l_{m-1})$ of the loads of m distinct bins after each of n identical balls has been assigned to a uniformly random and indipendently chosen bin. Our interest in the multinomial distribution lies in the fact that $\mathsf{MN}(n, m)$ is the distribution of the sizes of the pre-images of a random function from a domain of size n to a range of size m. Each sequence in the support of $\mathsf{MN}(n, m)$ can be encoded using $O(m \log n)$ bits but for our applications this can be too large and thus we are interested in *succinct* encodings of size $\mathsf{polylog}(n, m)$. Clearly, this comes at the cost of having to settle for a distribution that that is only computationally indistinguishable from the multinomial distribution. Neverthless, this will suffice for our applications.

We define a *Pseudorandom Multinomial Sampler* (a PMNS) as a triplet of algorithms $(\mathsf{Gen}, \mathsf{S}, \mathsf{S}^{-1})$. Roughly speaking, Gen, on input security parameter 1^λ, samples a sequence $(l_0, \ldots, l_{m-1})$ and outputs an encoding k of it. Algorithms S and S^{-1} are used to access the encoding. Specifically, algorithm S takes the encoding k and a ball index $0 \le x \le n-1$ and outputs x that is the assigned bin for k. Finally, algorithm S^{-1} takes as input a description k and a bin index y and outputs the balls assigned to the y-th bin. There is a subtle point here that needs to be clarified. The multinomial distribution considers n identical balls being thrown into m distinct bins. So, we number the balls so that each bin has consecutively numbered balls and, for $i < j$, the balls in bin i have indices no larger than the balls in index j. In this way we obtain an *ordered* assignment of balls to bins. For pseudorandomness, we want the descriptions output by Gen to be computationally indistinguishable from a real MN sampler. In terms of correctness, we require S and S^{-1} to be functional inverses of each other.

Definition 4.2. *A* multinomial sampler *(MNS) for $\mathsf{MN}(n, m)$ with encoding space $\mathcal{K}$ is a triple of efficiently computable functions: a randomized encoding generation function $\mathsf{Gen} : \{0,1\}^* \to \mathcal{K}$, a deterministic function $\mathsf{S} : \mathcal{K} \times [n] \to [m]$, and a deterministic function $\mathsf{S}^{-1} : \mathcal{K} \times [m] \to 2^{[n]}$. such that $y \in [m]$,*

$$\Pr_{k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)} [\mathsf{S}^{-1}(k, y) \neq \{x \in [n] \mid \mathsf{S}(k, x) = y\}] \le \mathsf{negl}(\lambda).$$

To define pseudorandomness, we consider the *advantage* of adversary $\mathcal{A}$ as

$$\mathbf{Adv}^{\mathrm{pmns}}_{\mathsf{S}}(\lambda, \mathcal{A}) = \left| \Pr_{k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)} [\mathcal{A}^{|\mathsf{S}^{-1}(k,\cdot)|}(1^\lambda) = 1] - \Pr_{D \xleftarrow{\$} \mathsf{MN}(n,m)} [\mathcal{A}^{D(\cdot)}(1^\lambda) = 1] \right|,$$

where we use the notation $|\mathcal{O}(\cdot)|$ to indicate that the adversary can query an input in $[m]$ and gets back the size of the set returned and $D(y)$ is the load of bin y in the sampled sequence D.

Definition 4.3. *A multinomial sampler $\mathsf{MN}(n, m) = (\mathsf{Gen}, \mathsf{S}, \mathsf{S}^{-1})$ is a* pseudorandom multinomial sampler *(PMNS) if, for all efficient $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{pmns}}_{\mathsf{S}}(\lambda, \mathcal{A}) \le \mathsf{negl}(\lambda)$. Further, if $\mathbf{Adv}^{\mathrm{pmns}}_{\mathsf{S}}(\lambda, \mathcal{A}) = 0$ for all λ and adversaries $\mathcal{A}$, then we say that S is a* perfectly secure *multinomial sampler.*

4.2 Building an iPRF

Before explaining how we build a pseudorandom multinomial sampler, we will show how to compose a PRP with a PMNS to achieve an invertible pseudorandom function. First, it is useful to observe that a PRP is already close to an iPRF if the domain is large enough, which is a well known fact that is established by the PRF/PRP switching lemma [IR89, HWKS98, BR06, Din20]. However, for small domains, an adversary can easily observe the distributional differences between a PRP and a random function, since one will have no collisions. A potential fix, that has been well studied but still fails, is to use a truncated PRP, which would still allow for efficient inversion. This will allow for a few collisions and be harder to distinguish from a random function than an original PRP, but unfortunately there are still substantial biases from a random function for small domains [GG21]. The core issue with truncated PRPs is that each output *has the same pre-image size*. This means that means that after each sampled output, the conditional distribution of the next observed output is changed. This is fundamentally different than a random function from $[n]$ to $[m]$, which will have pre-image sizes that are distributed as though one threw n balls into m bins.

This is exactly where a pseudorandom multinomial sampler comes to the rescue. Our PMNS primitive provides an ordered assignment of balls to bins that is clearly not random looking at all. For this reason, we then use a PRP to randomly permute the balls. By first throwing n balls into m bins efficiently and then applying a PRP, we can make use of the invertibility of a PRP while preserving the necessary distribution of a random function! Theorem 4.4 gives the exact construction and proof that this of composition achieves our goals.

Theorem 4.4. *Let* P *be a secure PRP over* $[n]$ *and* $(\mathsf{Gen}, \mathsf{S}, \mathsf{S}^{-1})$ *be a secure PMNS from* $[n]$ *to* $[m]$*. Then, there exists a secure iPRF* iF*, defined as follows:*

- $\mathsf{iF.Gen}(1^\lambda) = (\mathsf{P.Gen}(1^\lambda), \mathsf{S.Gen}(1^\lambda))$
- $\mathsf{iF.F}((k_1, k_2), x) = \mathsf{S}(k_2, \mathsf{P}(k_1, x))$
- $\mathsf{iF.F}^{-1}((k_1, k_2), y) = \{\mathsf{P}^{-1}(k_1, x) : x \in \mathsf{S}^{-1}(k_2, y)\}$

Additionally, iF *is efficient and only requires a single call to the underlying* S *for both the forward an backward directions and one call to* P *for every domain or range element output.*

Proof. First, we comment that this construction is efficient when instantiated with an efficient PMNS and PRP. In Theorem 4.7, we will give a construction of such a PMNS and prove that it is efficient. We also can instantiate the above scheme with a specific small-domain PRP, like the Sometimes-Recurse Shuffle [MR14], that provides full security over domains of size n and makes only $O(\log n)$ calls to an underlying efficient PRF. This establishes that our iPRF construction will be efficient for any domain and range.

Next, we go on to prove the security of iF. To do so we consider a sequence of three pairs of oracles $(\mathcal{O}_0, \mathcal{O}_0^{-1}), (\mathcal{O}_1, \mathcal{O}_1^{-1}), (\mathcal{O}_2, \mathcal{O}_2^{-1})$ parameterized by n, m and the security parameter λ. We will show that the three pairs of oracles are indistinguishable and we finish the proof by showing that the first pair computes a random function and its inverse and the last pair computes an iF and its inverse.

The three pairs of oracles all work with a partition of $[n]$ into m sets $R[0], \ldots, R[m-1]$. Given the m sets, all oracle calls are served in the same way: oracle $\mathcal{O}$, on input x, returns y such that $x \in R[y]$; oracle $\mathcal{O}^{-1}$, on input y, returns $R[y]$. The three pairs of oracles though differ in how the m sets are computed.

Let us start by defining $\mathcal{O}_0$ and $\mathcal{O}_0^{-1}$. We sample $(l_0, \ldots, l_{m-1})$ according to distribution $\mathsf{MN}(n, m)$ and then we define $R[0], \ldots, R[m-1]$, by picking a random permutation Π of $[n]$ and assigning to $R[0]$ the image under Π of $\{0, \ldots, l_0 - 1\}$; to $R[1]$ the image under Π of $\{l_0, \ldots, l_0 + l_1 - 1\}$; and so on, until $R[m-1]$ that has the image under Π of $\{n - l_{m-1}, \ldots, n-1\}$. The partition of $[n]$ into the sets $R[0], \ldots, R[m-1]$ then defines the input output behaviors of $\mathcal{O}_0$ and $\mathcal{O}_0^{-1}$ according to the rules above. It is easy to see that $\mathcal{O}_0$ and $\mathcal{O}_0^{-1}$ compute a random function and its inverse, respectively.

Now we define $\mathcal{O}_1$ and $\mathcal{O}_1^{-1}$. The only difference from the previous pair is in the construction of $(l_0, \ldots, l_{m-1})$. Specifically, we start by executing $k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$ and by setting $l_y = |\mathsf{S}^{-1}(k, y)|$, for $y = 0, \ldots, m-1$. The sets $R[0], \ldots, R[m-1]$ are defined as in $\mathcal{O}_0$ and $\mathcal{O}_0^{-1}$, based on the sequence $(l_0, \ldots, l_{m-1})$.

If $(\mathsf{Gen}, \mathsf{S}, \mathsf{S}^{-1})$ is a PMNS, then it is easy to see that $(\mathcal{O}_1, \mathcal{O}_1^{-1})$ are indistinguishable from $(\mathcal{O}_0, \mathcal{O}_0^{-1})$. For the sake of contradiction, suppose that there exists an efficient adversary $\mathcal{A}$ and a polynomial poly for which

$$|\Pr[\mathcal{A}^{\mathcal{O}_0(\cdot),\mathcal{O}_0^{-1}(\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1(\cdot),\mathcal{O}_1^{-1}(\cdot)}(1^\lambda) = 1]| \geq 1/\text{poly}(\lambda)$$

and consider the following adversary $\mathcal{B}$ that has access to an oracle $\mathcal{M}$ that returns sequences $(l_0, \ldots, l_{m-1})$. $\mathcal{B}(1^\lambda)$ obtains a sequence $(l_0, \ldots, l_{m-1})$ by invoking $\mathcal{M}(1^\lambda)$ and then sets up two oracles $\mathcal{G}$ and $\mathcal{G}^{-1}$ based on the sequence obtained as above. $\mathcal{B}$ receives $(l_0, \ldots, l_{m-1})$, picks a random permutation Π of $[n]$ and constructs the partition of $[n]$ into sets $R[0], \ldots, R[m-1]$ used by $\mathcal{G}$ and $\mathcal{G}^{-1}$. Then $\mathcal{B}$ runs $\mathcal{A}$ with access to the two oracles. When $\mathcal{A}$ stops and returns a bit, $\mathcal{B}$ stops and returns the same bit.

Now observe that if $\mathcal{M}$ outputs a sequence sampled according to $\mathsf{MN}(n, m)$, then $\mathcal{B}$ is providing $\mathcal{A}$ with access to $(\mathcal{O}_0, \mathcal{O}_0^{-1})$. Consider the case when the sequence output by $\mathcal{M}$ is obtained by first running $k \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$ and then by setting $l_y = |\mathsf{S}^{-1}(k, y)|$, for $y = 0, \ldots, m-1$. Then, in this case, $\mathcal{B}$ is providing $\mathcal{A}$ with access to $(\mathcal{O}_1, \mathcal{O}_1^{-1})$. By our assumption that $\mathcal{A}$ can distinguish the two pairs of oracles, we obtain that $\mathcal{B}$ can break the pseudorandomness of the PMNS $(\mathsf{Gen}, \mathsf{S}, \mathsf{S}^{-1})$ providing a contradiction.

Finally, we define oracles $\mathcal{O}_2$ and $\mathcal{O}_2^{-1}$. Here, we first randomly sample a pseudorandom permutation $k_1 \xleftarrow{\$} \mathsf{P.Gen}(1^\lambda)$ and an encoding $k_2 \xleftarrow{\$} \mathsf{S.Gen}(1^\lambda)$ of a multinomial sampler. Then we set $R[y] = \{\mathsf{P}(k_1, x) : x \in \mathsf{S}^{-1}(k_2, y)\}$. Towards a contradiction, suppose there exists an efficient adversary $\mathcal{A}$ satisfying

$$\left|\Pr[\mathcal{A}^{\mathcal{O}_2(\cdot),\mathcal{O}_2^{-1}(\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{\mathcal{O}_1(\cdot),\mathcal{O}_1^{-1}(\cdot)}(1^\lambda) = 1]\right| \geq 1/\text{poly}(\lambda)$$

and consider now the following efficient adversary $\mathcal{B}$ that has access to a pair of oracles $\mathcal{M}$ and $\mathcal{M}^{-1}$. $\mathcal{B}$ prepares the sets $R[0], \ldots, R[m-1]$ by randomly sampling an encoding $k \xleftarrow{\$} \mathsf{S.Gen}(1^\lambda)$ of a multinomial sampler and then by setting, for $y = 0, \ldots, m-1$, $R[y] = \{\mathcal{M}(x) : x \in \mathsf{S}^{-1}(k, y)\}$. Then, $\mathcal{B}$ executes $\mathcal{A}$ by providing access to oracles $\mathcal{G}$ and $\mathcal{G}^{-1}$ based on the partition of $[n]$ constructed. When $\mathcal{A}$ stops and outputs a bit, $\mathcal{B}$ stops and outputs the same bit. Now we make the following observations. If $\mathcal{M}$ is a random permutation of $[n]$ then $\mathcal{A}$ is run with access to oracles $\mathcal{O}_1, \mathcal{O}_1^{-1}$. On the other hand if $\mathcal{M}$ implements a pseudorandom permutation P then $\mathcal{A}$ is run with access to oracles $\mathcal{O}_2, \mathcal{O}_2^{-1}$. By our assumption that $\mathcal{A}$ can distinguish the two pairs of oracles we obtain that $\mathcal{B}$ can break the pseudorandomness of the PRP.

The proof is completed by observing that oracles $\mathcal{O}_2$ and $\mathcal{O}_2^{-1}$ evaluate, respectively, an iF and its inverse. □

Corollary 4.5 follows immediately as any secure PRF with efficient inversion satifies the properties of being a PMNS.

Corollary 4.5. *Let P be a secure PRP and F be a secure PRF with an efficiently computable deterministic function $\mathsf{F}^{-1} : \mathcal{K} \times \mathcal{R} \to 2^{\mathcal{D}}$ which correctly computes the inverse of F (i.e. $x \in \mathsf{F}^{-1}(k, y)$ iff $\mathsf{F}(k, x) = y$ for every $x \in \mathcal{D}$, $y \in \mathcal{R}$, and $k \in \mathcal{K}$). Then, there exists an efficient and secure iPRF iF, defined as follows:*

- $\mathsf{iF.Gen}(1^\lambda) = (\mathsf{P.Gen}(1^\lambda), \mathsf{F.Gen}(1^\lambda))$
- $\mathsf{iF.F}((k_1, k_2), x) = \mathsf{F}(k_2, \mathsf{P}(k_1, x))$
- $\mathsf{iF.F}^{-1}((k_1, k_2), y) = \{\mathsf{P}^{-1}(k_1, x) : x \in \mathsf{F}^{-1}(k_2, y)\}$

Additionally, iF is efficient and only requires a single call to the underlying F for both the forward an backward directions and one call to P for every domain or range element output.

Proof. Note that a PRF F with an efficiently computable inverse is a pseudorandom multinomial sampler. This is an immediate corollary of Theorem 4.4. □

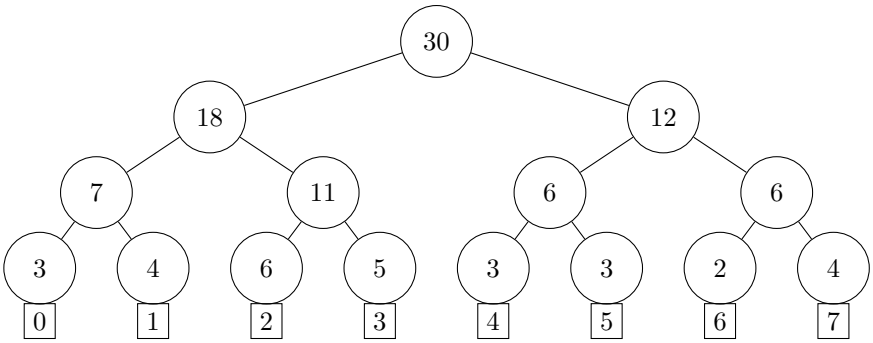

Figure 3: Visualization of how our $\mathsf{S} : [30] \to [8]$ construction from Figure 4 might throw 30 balls into 8 bins. By seeding the samples, one can compute the value of a leaf by sampling along its path.

4.3 Building a Pseudorandom Multinomial Sampler.

Now that we understand how a pseudorandom multinomial sampler can be used, we can explore how we can build one.

The key insight is in the following lemma and it relies on the Binomial distribution. The Binomial distribution is very closely related to the multinomial as it corresponds to the case of $m = 2$ to bins. It will be useful in our treatment to consider $\mathsf{Binomial}(n, p)$, for $0 \leq p \leq 1$, as the distribution of the number of balls in the first bin when each of the n balls is assigned independently to the first bin with probability p (and to the second bin with probability $1 - p$).

Intuitively, the following method of sampling the multinomial distribution is used in our construction of a pseudorandom multinomial sampler.

Multinomial Algorithm

For every n and m and for every $1 \leq j \leq m - 1$, the following process outputs a sequence distributed according to $\mathsf{MN}(n, m)$:

1. If $n = 0$ then Output: the sequence $(0, \ldots, 0)$ consisting of m 0's.
2. If $m = 1$ then Output: the sequence (n).
3. Sample $L \xleftarrow{\$} \mathsf{Binomial}(n, j/m)$ and set $R = n - L$.
4. Let $\texttt{left} \xleftarrow{\$} \mathsf{MN}(L, j)$.
5. Let $\texttt{right} \xleftarrow{\$} \mathsf{MN}(R, m - j)$.
6. Output: $\texttt{left}||\texttt{right}$.

In other words, the algorithm above says that to sample $\mathsf{MN}(n, m)$ we can partition the m bins in a left part, consisting of j bins, and in a right part, consisting of $m - j$ bins. Then we sample the number of balls L that go to the left part according to $\mathsf{Binomial}(n, j/m)$. Finally we recurse on the left part to assign the L balls to the j bins and $R = n - L$ balls to the remaining $m - j$ bins. We implicitly prove the above algorithm correctly samples the $\mathsf{MN}(n, m)$ in the proof of Theorem 4.7.

For simplicity, let us visualize the process above for $m = 2^\kappa$, for some integer κ. In this case we can pick $j = m/2 = 2^{\kappa-1}$ and the Binomial invocation tells us whether the ball falls in left half or in the right half. Specifically, balls $0, \ldots, L - 1$ fall in the first half or not. This determines the first bit of the bin index. We can then continue specifying the bits of the bin index in a similar for all κ bits.

To build a pseudorandom multinomial sampler, we could try to repeat this n times with some pseudorandomness and record where each ball lands. This would give us an efficient way to determine which bin any particular ball lands in, just by recomputing the $\log m$ random bits that correspond to that ball. However, the inversion function would require finding the balls which landed in a particular bins, which requires recomputing and testing each of the n balls' bins to see if it landed in the bin in question.

A key insight, however is that we only need the *number of balls* in each bin to be distributed as though we threw n balls into m bins. And by throwing all n balls into the bins "at the same time," our construction can simulate this distribution and invert it efficiently. We visualize our construction in Figure 3. Following the same idea of coin flips for each ball, we sample (pseudorandomly) from the binomial distribution at each node in the tree. Then, based on the number output, we send that many balls to the left child and the remaining to the right child. By recursively evaluating a binomial distribution with the number of passed from the parent, our leaf nodes will contain some number of balls that is distributed identically to that of throwing n balls into m bins.

In more detail, our PMNS construction works by tracing through a binary tree to determine the bin that a specific ball lands in (the forward evaluation S) or list all of the balls in a specific bin (the reverse evaluation S^{-1}). In either direction, our scheme follows the same basic principle of walking the path of a binary tree, tracking all of the balls which appear along the path. The difference between the forward and backward evaluation is how the algorithm determines which path it will follow, and what it ultimately returns. Note that throughout our construction, the only pseudorandomness comes from our use of a pseudorandom function to sample the from the binomial distribution. If one were to instead use true randomness, then our construction would be perfectly secure as defined in Definition 4.2.

When evaluated in either direction, our algorithm S begins by sampling a number $s \sim \mathsf{Binomial}(n, 1/2)$ from a binomial distribution, which is seeded by an evaluation of some function $F(k, \cdot)$ (usually a PRF), so that it is the same on each evaluation. This number represents the number of balls which are in the left-most bins (bins $0, \ldots, m/2 - 1$). Notice that in a random multinomial distribution, this is distributed according to $\mathsf{Binomial}(n, 1/2)$.

Notice that our S construction only needs to have the *number of balls* to match the multinomial distribution, and in particular, it does not need the indices of the balls in the bins to appear random. So, the algorithm matches the multinomial distribution at the first step by sending the balls $0, \ldots, s-1$ to the left and the remaining balls $s, \ldots, n$ to the right. This way we can encode the set of balls along each path (left or right) using the just the number of balls (s or $n - s$) and the start index (0 or s) which take at most $2 \log n$ bits. Additionally, the algorithm tracks where it is in the tree by storing the leftmost and rightmost leaves it can reach at each step, using only $2 \log m$ bits.

After this first step, the algorithm will determine whether to recurse to the left or right depending on its input. If evaluated in the forward direction S will check where the input ball $x \in [n]$ was sent. So, for example if $x < s$, then it will recurse left and otherwise will recurse right. If instead, the algorithm runs the backward direction S^{-1}, the algorithm determines if the input bin $y \in [m]$ is to the left or right of the root node (i.e. if $y < m/2$) and recurse toward the bin y.

Now at this next node, S changes the total number of balls and start index from the root's implicit n and 0 respectively to the inherited values for the children, s and 0 (if it went left) or $n - s$ and s (if it went right). From here, if we call the new total of balls $\hat{n}$ and the start index $\hat{\sigma}$, our algorithm proceed with the same procedure as before! In particular, the algorithm will sample a new $\hat{s} \sim \mathsf{Binomial}(\hat{n}, 1/2)$, again seeded with function $F(k, \cdot)$ to remain consistent across queries. Then, it will send $\hat{s}$ balls to the left starting with ball $\hat{\sigma}$ and $\hat{n} - \hat{s}$ balls to the right starting with ball $\hat{\sigma} + \hat{s}$.

From here, the algorithm continues to recurse left or right along its desired path. When running forward, S will go left when the input ball $\hat{\sigma} \le x \le \hat{\sigma} + \hat{s} - 1$ and otherwise will go right. When running backward, S^{-1} will go left when the input bin y is to the left of the current node, which it can determine using the leftmost and rightmost descendant leaves.

Once the algorithm has reached a leaf node (in $\log m$ rounds), it outputs the desired information based on the direction that it was queried. When running forward, S will have traced the ball $x \in [n]$ all the way down to the bin that it landed it, so the algorithm only needs to output the bin index $y \in [m]$ that x was

$\underline{\mathsf{S}(k, x \in [n])}$
$\mathsf{start} \leftarrow 0$; $\mathsf{count} \leftarrow n$
$\mathsf{low} \leftarrow 0$; $\mathsf{high} \leftarrow m - 1$
$\mathsf{node} \leftarrow (\mathsf{start}, \mathsf{count}, \mathsf{low}, \mathsf{high})$
While $\mathsf{low} < \mathsf{high}$:
  $(\mathsf{left}, \mathsf{right}, s) \leftarrow \mathsf{children}(k, \mathsf{node})$
  If $x < \mathsf{start} + s$ then $\mathsf{node} \leftarrow \mathsf{left}$
  Else $\mathsf{node} \leftarrow \mathsf{right}$
  $(\mathsf{start}, \mathsf{count}, \mathsf{low}, \mathsf{high}) \leftarrow \mathsf{node}$
Return low

$\underline{\mathsf{S}^{-1}(k, y \in [m])}$
$\mathsf{start} \leftarrow 0$; $\mathsf{count} \leftarrow n$
$\mathsf{low} \leftarrow 0$; $\mathsf{high} \leftarrow m - 1$
$\mathsf{node} \leftarrow (\mathsf{start}, \mathsf{count}, \mathsf{low}, \mathsf{high})$
While $\mathsf{low} < \mathsf{high}$:
  $(\mathsf{left}, \mathsf{right}, s) \leftarrow \mathsf{children}(k, \mathsf{node})$
  $\mathsf{mid} \leftarrow \lfloor (\mathsf{high} + \mathsf{low})/2 \rfloor$
  If $y \leq \mathsf{mid}$ then $\mathsf{node} \leftarrow \mathsf{left}$
  Else $\mathsf{node} \leftarrow \mathsf{right}$
  $(\mathsf{start}, \mathsf{count}, \mathsf{low}, \mathsf{high}) \leftarrow \mathsf{node}$
Return $\{\mathsf{start}, \ldots, \mathsf{start} + \mathsf{count} - 1\}$

$\underline{\mathsf{children}(k, \mathsf{node})}$
$(\mathsf{start}, \mathsf{count}, \mathsf{low}, \mathsf{high}) \leftarrow \mathsf{node}$
$\mathsf{mid} \leftarrow \lfloor (\mathsf{high} + \mathsf{low})/2 \rfloor$
$p \leftarrow (\mathsf{mid} - \mathsf{low} + 1)/(\mathsf{high} - \mathsf{low} + 1)$
$s \leftarrow \mathsf{Binomial}(\mathsf{count}, p; F(k, \mathsf{node}))$
$\mathsf{left} \leftarrow (\mathsf{start}, s, \mathsf{low}, \mathsf{mid})$
$\mathsf{right} \leftarrow (\mathsf{start} + s, \mathsf{count} - s, \mathsf{mid} + 1, \mathsf{high})$
Return $(\mathsf{left}, \mathsf{right}, s)$

Figure 4: Construction of a PMNS. $\mathsf{Binomial}(n, p; r)$ is a derandomized binomial sampling function using r as the randomness. F can be instantiated with a PRF or random function for computational or statistical security, respectively.

eventually thrown into. When running backward, S^{-1} will have tracked the start index σ and number of balls n' that landed in its input bin $y \in [m]$. So, in order to output the set of all balls that end up in bin y, the algorithm just outputs the enumerated set $\{\sigma, \sigma + 1, \ldots, \sigma + n' - 1\}$ (or $\emptyset$ if $n' = 0$). Of course, in practice, we would probably have our algorithm to just output a compressed representation of this set by just giving the start index and number of balls (especially when $n \gg m$). This additional practical benefit of our scheme that falls outside our formalism used throughout the paper. However, it enables additional functionality that may be practically useful. For example, our S scheme can be easily adapted to efficiently sample from the pre-image even when it is inefficient to enumerate the pre-image.

For a more precise formulation, we give pseudocode in Figure 4. This pseudocode follows the same intuition as above but will additionally work for m that is not a power of 2 (by biasing the binomial sampling). In Theorem 4.7, we formally prove that this construction is in fact a PMNS, using following Lemma 4.6.

Lemma 4.6. *Let n be an integer and $p \in [0, 1]$. Let $X \sim \mathsf{Binomial}(n, p)$ and $Y \sim \mathsf{Binomial}(X, q)$. Then, $Y \sim \mathsf{Binomial}(n, pq)$.*

Proof. We can express X as a sum of n independent Bernoulli random variables $X = X_1 + \cdots + X_n$, where $X_i \sim \mathsf{Bernoulli}(p)$. We can similarly express Y as $Y = X_1 Y_1 + \cdots + X_n Y_n$, where $Y_i \sim \mathsf{Bernoulli}(q)$. Finally, using the fact that product of two Bernoulli random variables is itself a Bernoulli random variable, we observe each $X_i Y_i = Z_i$, where $Z_i \sim \mathsf{Bernoulli}(pq)$. So, we can write $Y = Z_1 + \cdots + Z_n$, establishing that $Y \sim \mathsf{Binomial}(n, pq)$. □

Theorem 4.7. *The construction S in Figure 4 from $[n]$ to $[m]$ is a computationally secure PMNS when F is a pseudorandom function with range of size at least n. And, it is statistically secure when F is a truly random function.*

Additionally, S *is efficient and only requires* $O(\log m)$ *time and* $\log m$ *calls to the* F*, when evaluated in the forward or reverse direction.*

Proof. First, we establish that the construction in Figure 4 is efficient. In particular, on any forward or reverse evaluation our algorithm makes at most $\lceil \log m \rceil$ calls to children as it traces along its path to a leaf (when $\mathsf{low} = \mathsf{high}$). The only other non-atomic operations in our code are the calls to F and $\mathsf{Binomial}$. By assumption F is efficient, and $\mathsf{Binomial}$ can be implemented efficiently which is logarithmic in the input length (see [Dev86]). So, when F is implemented with a constant-time PRF, our algorithm can run in time that is polylog(n, m).

Next we go on to prove the security of our construction. By assumption, $F(k, \mathsf{node})$ outputs elements that are indistinguishable from a random sample over a range of size at least n. So, we will prove that when every call to $F(k, \mathsf{node})$ is uniformly random and independently sampled, the tuple

$$(|\mathsf{S}^{-1}(k,0)|, |\mathsf{S}^{-1}(k,1)|, \ldots, |\mathsf{S}^{-1}(k,m-1)|),$$

is identical to

$$(D(0), \ldots, D(m-1)) \sim \mathsf{MN}(n, m).$$

In other words, we assume each uniquely seeded binomial sample in the construction Figure 4 is independent of other binomial samples. This will also establish the further claim that it is statistically secure when F is a random function.

For the remainder of the proof, let

$$X = (X_1, \ldots, X_m) = (|\mathsf{R}^{-1}(1)|, |\mathsf{R}^{-1}(2)|, \ldots, |\mathsf{R}^{-1}(m)|),$$

and

$$Y = (Y_1, \ldots, Y_m) = (D(1), \ldots, D(m)).$$

We will show that $\mathrm{TV}(X; Y) = 0$.

For simplicity we assume that m is a power of 2. Observe that Y_1 is generated as a series of binomial samples. If we write $S_1, \ldots, S_{\log m}$ as the intermediate samples, we see that $S_1 = n$, $S_2 \sim \mathsf{Binomial}(S_1, 1/2)$, and so on until $Y_1 \sim \mathsf{Binomial}(S_{\log m}, 1/2)$. By applying Lemma 4.6 repeatedly, we prove that $Y_1 \sim \mathsf{Binomial}(n, 1/m)$. Since choosing a random function is equivalent to throwing n balls into a m bins, we know the frequency of balls in the first bin by itself is distributed according to $X_1 \sim \mathsf{Binomial}(n, 1/m)$. This establishes that $\mathrm{TV}(X_1; Y_1) = 0$.

Next, we prove that for $k \geq 1$, $\mathrm{TV}(X_{k+1}; Y_{k+1} | (X_1, Y_1, \ldots, X_k, Y_k)) = 0$, which will establish the theorem.

First, observe that

$$X_{k+1} | X_1, \ldots, X_k \sim \mathsf{Binomial}(n - \sum_{i=1}^{k} X_i, \frac{1}{m-k}).$$

This can be seen because we are conditioning on sample on an exactly number of n balls that landed in the first k bins. The remaining balls, we know did not land in those bins, but will appear uniformly at random among the remaining $m - k$ bins.

All the remains is to establish is that

$$Y_{k+1} | Y_1, \ldots, Y_k \sim \mathsf{Binomial}(n - \sum_{i=1}^{k} Y_i, \frac{1}{m-k}).$$

Here we again consider the intermediate random variables on the path to Y_{k+1} and call them $S_1, \ldots, S_{\log m}$, each of which are the number of balls thrown that are thrown to their corresponding tree node. Notice that each S_i is exactly equal to the sum of all of its descendant leaves.

First, we consider the root node which corresponds to S_1 and is along the path to any leaf. By definition S_1 is always n with probability 1 and is therefore unaffected by conditioning on the values of $Y_1, \ldots, Y_k$. The count of S_2, however is usually sampled according according to $\mathsf{Binomial}(n, \frac{1}{2})$. But, when conditioning on

values $Y_1, \ldots, Y_k$, the distribution of S_2 is changed, because we know exactly how many balls travel along the paths to $Y_1, \ldots, Y_k$.

If S_2 is along the path of every $Y_1, \ldots, Y_k$ (when $k < m/2$), then this is equivalent to filling the first k bins with $Y_1, \ldots, Y_k$ balls respectively and then throwing the remaining balls into S_2 according to the number of remaining bins on either side of the midpoint. So, we can just write

$$S_2|Y_1, \ldots, Y_k \sim \sum_{i=1}^{k} Y_i + \mathsf{Binomial}(S_1 - \sum_{i=1}^{k} Y_i, \frac{m/2 - k}{m - k})$$

since S_2 is the left child, since $k + 1 \leq m/2$ and S_2 is on the path to Y_{k+1}.

If instead S_2 is only on some (or none) of the $Y_1, \ldots, Y_k$ paths, then a similar argument shows how S_2 is sampled. For example when $k \geq m/2$, S_2 is actually fixed by the values $Y_1, \ldots, Y_{m/2}$. So, we can write that $S_2|Y_1, \ldots, Y_k \sim \sum_{i=m/2+1}^{k} Y_i + \mathsf{Binomial}(S_1 - \sum_{i=1}^{k} Y_i, 1)$, for the right child which is determined entirely by S_1 (and we write the summation and binomial for convenience later on). Notice that if $k \geq m/2$ then S_2 will always be the right child, because the path to Y_{k+1} is further right in the tree than any of $Y_1, \ldots, Y_k$.

This analysis also applies to an intermediate node S_{i-1} with m_{i-1} total descendant leaves, including some of the fixed values $Y_a, \ldots, Y_k$. When determining the distribution of the child S_i in this conditioned space, we break into to cases. If fewer than $m_{i-1}/2$ leaves are conditioned (i.e. $k - a < m_{i-1}/2$), then

$$S_i|Y_1, \ldots, Y_k \sim \sum_{i=a}^{k} Y_i + \mathsf{Binomial}(S_{i-1} - \sum_{i=a}^{k} Y_i, \frac{m_{i-1}/2 - (k - a)}{m_{i-1} - (k - a)}),$$

since S_i is the left child, because $k+1 \leq a+m_{i-1}/2$ and S_i is on the path to Y_{k+1}. If instead at least $m_{i-1}/2$ leaves are conditioned, then the right child is sampled as $S_i|Y_1, \ldots, Y_k \sim \sum_{i=a+m_{i-1}/2}^{k} Y_i + \mathsf{Binomial}(S_{i-1} - \sum_{i=a}^{k} Y_i, 1)$, and S_i will never be the left child in this case because $k + 1$ is further right than all of the conditioned on leaves.

Additionally, the applies to the leaf nodes and in particular Y_{k+1}.

$$Y_{k+1}|Y_1, \ldots, Y_k \sim \mathsf{Binomial}(S_{\log m}, 1/2),$$

if none of $Y_1, \ldots, Y_k$ are descendants of $S_{\log m}$ and otherwise, if Y_k is a descendant of $S_{\log m}$, then $Y_{k+1} \sim \mathsf{Binomial}(S_{\log m} - Y_k, 1)$.

In either case, Y_{k+1} is sampled through a series of recursive binomial distributions. Each of the conditionally distributed S_i random variables can be written as some fixed value X_i added to a binomial sample, $S_i \sim X_i + \mathsf{Binomial}(S_{i-1} - Y_i, p_i)$. Notice however that in each case, $Y_i = X_{i-1}$ and that the base case S_1 is trivially a binomial distribution.

By applying Lemma 4.6 repeatedly, we find that

$$Y_{k+1} \sim \mathsf{Binomial}(S_1 - \sum_{i=1}^{k} Y_i, \prod_{j=2}^{\log m} p_i),$$

where each p_i is the probability used in the description of S_i's distribution. To finally establish the theorem, we argue that the product of each of these p_i is exactly $\frac{1}{m-k}$. This follows because the probabilities (that are not equal to 1) are telescoping so that the denominator of p_i is exactly the numerator of p_{i-1}. So, we are only left with the denominator of p_2 and numerator of $p_{\log m}$, which is exactly $\frac{1}{m-k}$. □

5 Single-Server PIR

In this section, we give a new construction for single-server offline-online PIR, Plinko, with two advantages. First, it obtains optimal query time $t = \tilde{O}(n/r)$ for any client storage size r. Prior works were unable

to obtain this trade-off for all client storage sizes as they require query time $t = \tilde{O}(r + (n/r))$. Secondly, Plinko enables worst-case polylog(n) time and $O(1)$ communication to update a database entry. Prior work required either $\Theta(n)$ worst-case runtime and communication for an update or $\tilde{\Theta}(\sqrt{n})$ runtime for updates with only $\Theta(\log n)$ bits of communication. We present a comprehensive comparison of Plinko and prior works in Figure 6.

5.1 Definitions

Before presenting our scheme, we recall the definition of offline-online private information retrieval in Definition 5.1. Our definition stresses the offline model of two prior works [ZPZS24, RMS24], which gives the client a single streaming pass over the database to compute their hints initially. We keep with this model in our formalism of $\mathsf{HintInit}$ by giving it streaming oracle access to D because it simplifies the presentation of our improvements over these schemes, allows using only one-way functions for security, and amortize out the cost of computing new hints.

However, more in line with other prior works, we could instead consider a $\mathsf{HintInit}$ protocol which is run under fully-homomorphic encryption by the single-server to return encrypted hints back to the client. In either model, we are the first to achieve the gains of efficient updates and an optimal space-time trade-off.

In our definition below, we use the notation $\mathsf{HintInit}^D$ to denote an algorithm that is run by the client that requires a streaming pass over the database D. In contrast, we write $A(x; y) \to z$ to denote algorithms that take variable(s) x as input, have random read (and/or write) access to some memory y, and output z. This is to allow our formalism to define algorithms that run in time that is sublinear in y (e.g. $\mathsf{UpdateHint}$).

Definition 5.1. *An* offline-online private information retrieval *(PIR) scheme for a database D of size n and supporting Q queries is a tuple $\Pi = (\mathsf{HintInit}, \mathsf{Query}, \mathsf{Answer}, \mathsf{Recon})$ of efficient algorithms:*

- $\mathsf{HintInit}^D(1^\lambda) \to \mathsf{st}$*, a randomized algorithm that takes a security parameter, has a single streaming pass of $D \in \{0,1\}^n$, and outputs a client state* st*.*
- $\mathsf{Query}(i; \mathsf{st}) \to (q, h)$*, a randomized algorithm that takes as an index $i \in [n]$, and read access to* st*, and outputs a query q and reconstruction hint h.*
- $\mathsf{Answer}(q; D) \to r$*, a deterministic algorithm that takes as input a query q and read access to $D \in \{0,1\}^n$ and outputs a response r.*
- $\mathsf{Recon}(h, r; \mathsf{st}) \to a$*, a randomized algorithm that takes as input reconstruction hint h, a server response r, and read/write access to* st*, and outputs an answer $a \in \{0,1\}$ and may modify* st*.*

For adversary $\mathcal{A}$ issuing at most Q queries to its $\mathcal{Q}$ oracle and PIR Π, we define

$$\mathbf{Adv}^{\mathrm{cor}}_{\Pi}(\lambda, \mathcal{A}) = \Pr[\mathrm{G}^{\mathrm{cor}}(\lambda, \mathcal{A}) = 1]$$

and

$$\mathbf{Adv}^{\mathrm{pir}}_{\Pi}(\lambda, \mathcal{A}) = \Pr[\mathrm{G}^{\mathrm{pir}}(\lambda, \mathcal{A}) = 1],$$

where the games are specified in Figure 5.

We all a PIR scheme Π correct if for all efficient $\mathcal{A}$ issuing at most Q queries to $\mathcal{Q}$ (and none to $\mathcal{U}$), $\mathbf{Adv}^{\mathrm{cor}}_{\Pi}(\lambda, \mathcal{A}) \leq \mathsf{negl}(\lambda)$. Similarly, we call a PIR scheme Π secure if for all efficient $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{pir}}_{\Pi}(\lambda, \mathcal{A}) \leq \mathsf{negl}(\lambda)$.

Definition 5.2. *An* updateable *PIR scheme is a tuple of six efficient algorithms. The first four are as defined in Definition 5.1, and the last two are defined as:*

- $\mathsf{UpdateDB}(i, d; D) \to \delta$*, a deterministic algorithm that take an update (i, d) and read/write access to the database D, and outputs a summary of the update δ and changes $D[i] \leftarrow d$.*
- $\mathsf{UpdateHint}(\delta; \mathsf{st})$*, a deterministic algorithm with no output that takes the a summary of updates δ, and read/write access to* st *which it may modify.*

Game $\mathrm{G}^{\mathrm{cor}}(\lambda, \mathcal{A})$	$\mathcal{Q}(i \in [n])$	$\mathcal{U}(i, d)$
$D \leftarrow \mathcal{A}(1^\lambda)$; $\mathsf{bad} \leftarrow 0$	$(q, h) \xleftarrow{\$} \mathsf{Query}(i; \mathsf{st})$	$\delta \leftarrow \mathsf{UpdateDB}(i, d; D)$
$\mathsf{st} \xleftarrow{\$} \mathsf{HintInit}^D(1^\lambda, Q)$	$r \leftarrow \mathsf{Answer}(q; D)$	$\mathsf{UpdateHint}(\delta; \mathsf{st})$
$\mathcal{A}^{\mathcal{Q},\mathcal{U}}(1^\lambda)$	$a \xleftarrow{\$} \mathsf{Recon}(h, r; \mathsf{st})$	Return δ
Return bad	If $a \neq D[i]$: $\mathsf{bad} \leftarrow 1$	
	Return a	

Game $\mathrm{G}^{\mathrm{pir}}(\lambda, \mathcal{A})$	$\mathcal{Q}(i_0 \in [n], i_1 \in [n])$	$\mathcal{U}(i, d)$
$D \leftarrow \mathcal{A}(1^\lambda)$; $b \xleftarrow{\$} \{0,1\}$	$(q, h) \xleftarrow{\$} \mathsf{Query}(i_b; \mathsf{st})$	$\delta \leftarrow \mathsf{UpdateDB}(i, d; D)$
$\mathsf{st} \xleftarrow{\$} \mathsf{HintInit}^D(1^\lambda)$	$r \leftarrow \mathsf{Answer}(q; D)$	$\mathsf{UpdateHint}(\delta; \mathsf{st})$
$b' \leftarrow \mathcal{A}^{\mathcal{Q},\mathcal{U}}(1^\lambda)$	$a \xleftarrow{\$} \mathsf{Recon}(h, r; \mathsf{st})$	Return δ
Return $b = b'$	Return q	

Figure 5: Games for PIR correctness and security. The $\mathcal{U}$ oracles are only used for updateable PIR schemes. Here, n is the size of the database D.

We say an updateable PIR scheme is correct and secure if it satisfies for all efficient $\mathcal{A}$ issuing at most Q queries to $\mathcal{Q}$ (and any number to $\mathcal{U}$), $\mathbf{Adv}^{\mathrm{cor}}_{\Pi}(\lambda, \mathcal{A}) \leq \mathsf{negl}(\lambda)$ and $\mathbf{Adv}^{\mathrm{pir}}_{\Pi}(\lambda, \mathcal{A}) \leq \mathsf{negl}(\lambda)$ respectively.

We give formal games in Figure 5 for the correctness and privacy of a PIR scheme, and explicitly allow the adversaries to call an update oracle. However, these games are effectively equivalent to the traditional definitions given for security and correctness. We define our offline-online PIR and games with respect to a fixed number of queries Q, because we can generically convert a PIR satisfying the syntax to one which runs $\mathsf{HintInit}$ over the Q queries by streaming the database in n/Q blocks to prepare a new set of hints before the first set expires.

5.2 Our Construction: Plinko

Now we present our new single-server PIR scheme, called Plinko. Plinko is built on top off [RMS24], but uses invertible PRFs (Section 4) to achieve significant asymptotic improvements. Although, we note our techniques are flexible as they can also be applied to [ZPZS24] for example (Appendix B). Plinko is parameterized with respect to a security parameter λ, a block size w, and a number of supported queries before refresh q. We view w and q as parameters which influence the amount of storage required by the client. In prior work, these parameters were fixed around $\sqrt{n}$ for simplicity and asymptotic optimums, but we present Plinko without fixing these, since it highlights where we differ from prior work and how we achieve an optimal trade-off between client storage and query time.

Theorem 5.3. *Assuming one-way functions, there exists a single-server, updatable, offline-online PIR scheme such that for every n-bit database and every choice of client storage parameter $r < n$:*

- *Each online query runs in $\tilde{O}(n/r)$ time.*
- *Each online query uses $\tilde{O}(n/r)$ bits of communication.*
- *Each database update runs in time $\tilde{O}(1)$ time.*
- *Each database update uses $O(\log n)$ bits of communication.*
- *The client uses $\tilde{O}(r)$ memory.*
- *The offline phase runs in $\tilde{O}(n)$ time and uses $O(n)$ communication.*

For simplicitly, we consider single-bit entries, but one can easily extend our construction to B-bit entries. The rest of this section will prove this theorem.

Construction with Improved Query Time. We present our Plinko construction. For detailed pseudocode, see Figure 7. Following previous work, starting from [PPY18], we let the client pre-compute hints during an offline phase. Roughly speaking, a hint is the description of a subset $P \subset [n]$ of database entries along with the parity (bitwise XOR) of all entries in P denoted by p. The hints may be constructed with a single streaming pass of the database or, at the cost of additional assumptions, we could use either fully-homomorphic encryption or a second server. Following [ZPZS24, RMS24], we will assume the streaming approach.

In a query for index x, we look for a hint such that $x \in P$, ask the server to send us the parities of the entries in $P \setminus \{x\}$ without revealing the index x. The client obtains x-th entry by XORing the response with original parity p. A hint can only be used once and thus we have the problem of replacing the consumed hint with a new hint that includes x for two reasons. First, if we do not, we could be left with no hint that includes x and will not be able to query for x. More importantly, we want the distribution of the hint be independent of the queries that have been performed. Next, we are going to describe a novel way of computing the hints that is designed to efficiently search for hints that contains the entry for a given index x. This will reduce client computation time at query time and enable efficient hint modifications when the database is updated.

The set $[n]$ of the indices of the database is partitioned into $c := n/w$ *blocks* of w consecutive indices. We write each index $i \in [n]$ as $i = \alpha w + \beta$ with $0 \leq \alpha \leq c-1$ and $0 \leq \beta \leq w-1$. One can view that the i-th entry is in the α-th block at offset β. Algorithm $\mathsf{HintInit}$ constructs a set of $\lambda w + q$ hints: λw *regular* hints and q *backup* hints. We present Plinko in generality, but it may be helpful to think of $w = q = \tilde{O}(r)$ to ensure $\tilde{O}(r)$ client storage. Regular and backup hints are stored in table H and T respectively with each table having $\lambda w + q$ slots. The regular hint table H is initialized with λw regular hints that are stored in slots $0, \ldots, \lambda w - 1$. The backup table T is initialized with q backup hints that are stored in slots $\lambda w, \ldots, \lambda w + q - 1$. The k-th query will consume one regular hint and *promote* the backup hint $T[\lambda w + k]$ to a regular hint by moving it to $H[\lambda w + k]$. Thus, the number of regular hints always stays λw. Moreover, the backup hint keeps the same table index when it moves from T to H and therefore the first λw slots of the backup table T stay empty throughout the q queries whereas the last q slots of the regular table H are filled one at a time with a promoted backup hint. All hints (regular, backup and promoted backup) specify a subset of the blocks and a randomly chosen offset within each chosen block. The hint will also include the parity of the database entries at each of the random offsets in the chosen subset of blocks. Regular and backup hints differ only on the number of chosen blocks of blocks. Backup hints will require storing two parities (instead of one). Also, promoted backup hints have a slightly different way of specifying the offsets within blocks than regular hints.

We start with the regular hints found in $H[0], \ldots, H[\lambda w - 1]$. For $j \in [\lambda w]$, we randomly select a subset of $c/2 + 1 = n/(2w) + 1$ blocks denoted P_j. For each block $\alpha \in P_j$, we select entry $i = \alpha w + \mathsf{iF.F}(k_\alpha, j)$, where k_α is the seed for block α. Note, $\mathsf{iF.F}(k_\alpha, j)$ is the offset. We denote p_j as the parity of the $c/2 + 1$ chosen database entries specified in P_j. The j-th regular hint is $H[j] = (P_j, p_j)$. Next, the q backup hints are found in $T[\lambda w, \ldots, \lambda w + q - 1]$. The j-th backup hint will consist of three components $T[j] = (B_j, \ell_j, r_j)$ for $j \in \{\lambda w, \ldots, \lambda w + q - 1\}$. B_j is a randomly selected subset of $c/2 = n/(2w)$ blocks. We select offsets using $\mathsf{iF.F}(k_\alpha, j)$ as before for all c blocks $\alpha \in [c]$. The parity ℓ_j computes the XOR of the $c/2$ database entries specified in B_j at the offsets chosen by iF. In contrast, the parity r_j computes the XOR of the $c/2$ database entries outside of B_j that are also at the offsets picked by iF.

Note that the same seed k_α is used for each block $\alpha \in [c]$ in both regular and backup hints. This has the following significant advantage when searching for hints for any entry x. Suppose that $x = \alpha w + \beta$ meaning the x-th entry is in block α at offset β. Then, we see that $\mathsf{iF.F}^{-1}(k_\alpha, \beta)$ immediately returns the subset of all hints (both regular and backup) where the offset in the α-th block has chosen β (i.e., the x-th entry).

Let us now describe how a query for entry x is performed. Let $x = \alpha w + \beta$ be an entry from block α with offset β. First, we find a regular hint $H[j]$ such that $\alpha \in P_j$ and $\mathsf{iF.F}(k_\alpha, j) = \beta$. To do this, we simply execute $\mathsf{iF.F}^{-1}(k_\alpha, \beta)$ to compute the subset of hints containing the x-th entry. Note, this is a key difference between Plinko and prior work [ZPZS24, RMS24] as a hint containing the x-th entry can be searched efficiently. We also later show that the number of hints containing the x-th entry is $\tilde{O}(1)$ (except

with negligible probability). For each of these $\tilde{O}(1)$ candidate hints, we enumerate the subset of blocks P_j to see if the α-th block was chosen. Altogether, this requires $\tilde{O}(n/r)$ total time. When using normal PRFs, the process of finding all $\tilde{O}(1)$ candidates would require checking the offsets for each of the $\lambda w = \tilde{O}(r)$ unused regular and promoted backup hints individually meaning an additional $\tilde{O}(r)$ time is required during queries.

Back to querying, we pick a random hint in H (either, regular or promoted backup) that was not previously used. For now, suppose this is a regular hint, $H[j] = (P_j, p_j)$. The query constructs the following two sets of $c/2$ indices denoted by S and $\hat{S}$. S will consist of all database entries with the x-th entry removed. $\hat{S}$ will consist of a random entry from each block outside of P_j, which is $[c] \setminus P_j$ as well as α. We can define S and $\hat{S}$ as follows:

$$S = \{i = aw + b : a \in P_j \setminus \{\alpha\}, b = \mathsf{iF.F}(k_a, j)\}$$
$$\hat{S} = \{i = aw + b : a \in ([c] \setminus P_j) \cup \{\alpha\} \text{ and each } b \text{ is randomly chosen from } [w]\}.$$

One can also set b as the output of iF for all choices of $a \neq \alpha$ for $\hat{S}$. This is done in our pseudocode (Figure 7) and [RMS24]. The client sends the two sets to the server in random order. The server computes two parities of entries $s = \oplus_{i \in S} D[i]$ and $\hat{s} = \oplus_{i \in \hat{S}} D[i]$ and sends them to the client. The client uses the parity of entries in S to recover $D[x] = p_j \oplus s$. For privacy, we note that the server sees a random partitioning of the c blocks into two equal parts of size $c/2$ as well as a random entry within each of the c blocks. All of this is independent of the query x.

Next, we show how Plinko promotes backup hints following [RMS24]. After the k-th query at index x has been completed, we need to provide our hint table with a hint that contains x in its partition and offset vectors (in order to avoid skewing our future queries). Fortunately, we can always promote the next backup hint $T[j] = (B_j, \ell_j, r_j)$, with $j = \lambda w + k$, from our backup table to replace whichever hint was just consumed. Effectively, if x was in block α with offset β and B_j does not contain block α, then we can implicitly add the block α to B_j by adding x to the promoted backup hint. Thus, obtaining a partition of size $c/2 + 1$ like in a regular hint. We then update ℓ_j by setting $\ell_j = \ell_j \oplus D[x]$ and, finally, set $H[j] = (B_j, 0, x, \ell_j)$. If instead $\alpha \in B_j$, we update r_j by setting $r_j = r_j \oplus D[x]$ and set $H[j] = (B_j, 1, x, r_j)$. Note that r_j is the XOR of all entries in $[c] \setminus B_j \cup \{a\}$. The 0-1 entry in the promoted backup hint tells us whether we should consider B_j or its complement $\overline{B_j} = [c] \setminus B_j$ as the block subset equivalent in regular hints. Note, all of this remains compatible with invertible PRFs.

As it is clear from the above, a promoted backup hint $H[j] = (B_j, \eta, x, p_j)$ will not show up as a candidate when we look for hints containing x, as it is very unlikely that $\mathsf{iF.F}(k_a, b) = j$. Therefore, we store the result of each query in a hash table Cache and, in case of repeated queries, we return the value found in Cache and issue a query for another index that has not been queried before.

Finally, we note that the above only enables querying at most q times. There are several ways to address this. The simplest is to simply re-execute the offline phase. More recent works [ZPZS24, RMS24] propose amortizing the offline phase by streaming $n/q = \tilde{O}(n/r)$ database entries following each query. Therefore, the client can maintain a separate regular and backup hint table that is partially constructed after each query. Once the original regular and backup hints, the client can simply use the new set of regular and backup hints as the database has been completely streamed. Note, this does not increase query time or communication as each query already uses $\tilde{O}(n/r)$ communication and time.

Update Algorithm. Our updates offer another significant improvement that Plinko brings over the original construction [RMS24], because we can just use our hint searching algorithm, which just requires an iPRF inversion. To update an x-th entry of the database from u to u', the server just needs to send $(x, u \oplus u')$ to the client. Then, the client can perform an iPRF inversion, like when querying x, to enumerate hints which have x in them and update the corresponding parities. We note that the client must also update the Cache storing repeated query's answers as well as hints being constructed in the amortized offline phase. See Figure 7 for the full details.

Efficiency. We start with query time. As the first step, the client searches for a hint containing the query index x using our iPRF construction that returns the subset of all hints containing the x-th entry. We show later that at most $\tilde{O}(1)$ hints containing any single database entry. Afterwards, the client enumerates the offsets requiring $\tilde{O}(n/r)$ time to upload to the server. The server computes the corresponding parities in $\tilde{O}(n/r)$ (there is no change here compared to prior works [ZPZS24, RMS24]). To answer the query, the client picks the correct parity and computes a final parity. Afterwards, the client receives a streamed partition of the database to update client-stored hints also requiring $\tilde{O}(n/r)$ time. Altogether, the query requires $\tilde{O}(n/r)$ time. For updates, the server sends the index and contents of the updated entry to the client. The client executes a single iPRF inversion and updates all $\tilde{O}(1)$ hints containing the updated entry. Finally, for the offline phase, we use nearly identical algorithms as the ones in [ZPZS24, RMS24] with same efficiency.

It remains to show that any database entry does not appear in too many hints. For any $x \in [n]$, we note that each hint (regular or backup) independently chooses to include the x-th entry with probability $O(1/w)$. As there are $\lambda w + q = \tilde{O}(r)$ hints, we know the expected number of hints containing the x-th entry is $\tilde{O}(r/w) = \tilde{O}(1)$. By Chernoff's bound, we know that the x-th entry will not appear in more $\max\{O(\lambda + \log n), \tilde{O}(r/w)\}$ except with probability $2^{-\lambda - \log n}$. Recall that $w = \tilde{O}(r)$, so the x-th entry appears in at most $\tilde{O}(1)$ entries except with probability $2^{-\lambda - \log n}$. By a final union bound over all n entries, we know that no entry will have more than $\tilde{O}(1)$ entries except with probability $2^{-\lambda}$. So, all entries appear in at most $\tilde{O}(1)$ hints except with negligible probability.

Correctness. For correctness, we note that the main difference in our PIR scheme is that we generate hints in a different way than prior works where we replace PRFs with iPRFs. As iPRFs are also indistinguishable from random functions, we note that our hint distribution is identical to [RMS24]. As a result, we can directly use the correctness arguments from [RMS24]. For updates, we note that the client will correctly update all relevant hints as long as the underlying iPRF inversion algorithm is correct.

Privacy. The privacy argument follows quite similarly to the correctness argument. Our replacement of PRFs with iPRFs for hint generation does not change the hint distribution (in the view of a computational adversary). Therefore, we can also directly apply the privacy arguments from [RMS24]. In other words, the usage of iPRFs simply speeds up the algorithms for searching and updating hints. The usage of iPRFs does not affect anything else.

Invertible PRF Requirements. We also briefly discuss the iPRF requirements in our PIR scheme. In Plinko, we use an iPRF to map hints to offsets within a block. It is clear to see that our usage of iPRF requires security for both small domains and ranges as the number of hints is similar to the size of client storage meaning truncated PRPs cannot be used. Furthermore, in many natural parameter settings, the iPRF has a larger domain than range meaning that the injective invertible PRFs studied in [BKW17] are also unusable. Therefore, our new iPRF construction for arbitrary domain and range sizes are necessary to enable our improved PIR scheme.

We note out that our PIR scheme only requires the underlying invertible PRF to be computationally indistinguishable from a random function for an adversary with access to the forward oracle (as the server only sees forward evaluations). Nevertheless, we present invertible PRFs in generality as we believe the stronger security notion where adversaries also have access to the inverse oracle will allow easier usage in other applications and comes with no additional costs.

Necessity of Database Streaming. In the offline phase of Plinko (as well as [ZPZS24, RMS24]), the database is streamed to the client, requiring linear communication. One could consider other options (such as FHE used in [CHK22]) to improve the communication of the offline phase. If one constructs a single-server PIR with sub-linear offline phase communication and sub-linear query communication, it is easy to see that this can be used to build oblivious transfer (OT) following the reduction in [DMO00]. Given that OT requires public-key operations [IR89], the offline phase must use linear communication if one insists on building single-server PIR schemes with sub-linear query communication from only one-way functions (like

Plinko). Therefore, offline database streaming is optimal if we assume that OT exist (i.e., we only rely on the security of one-way functions). to avoid one-way functions altogether, or to achieve better communication

6 Conclusion

In this work, we show that it is possible to construct single-server PIR that obtains optimal trade-offs between client storage and query time for all parameters. Our new construction Plinko achieves this trade-off and additionally is the first single-server PIR with near-constant worst-case time and communication updates. In the process of achieving these efficiency games, we define the novel concept of invertible pseudorandom functions (iPRFs) which allow the client to quickly perform hint searching. By introducing this notion, we leave open the possibility of finding other applications for iPRFs in cryptography for either efficiency or security improvements.

Our works leaves open many interesting open questions for future work in PIR. We build our iPRF using minimal assumptions (i.e. one-way functions); however, there are properties of PRFs which one could try and build into iPRFs from other assumptions. For example, many PIR schemes improve their query communication using puncturable PRFs from a learning with errors assumption. A puncturable iPRF would lead to direct improvements for single-server PIR, and we leave this as a potential direction for future work.

Finally, although we achieve an optimal trade-off between client storage and query time, but we point out that there is still room for improvement. Current lower bounds do not bound the client's query time directly. It seems possible that one could construct the PIR scheme with nearly-constant online client query time. This would achieve the same total query trade-off that Plinko does but could be even more efficient for the client. In fact, some schemes [SACM21, LP23b, LP23a, ZLTS23] have nearly-constant online time *after* a client has found a hint. Developing an efficient hint finding scheme for these papers could be an interesting improvement for better asymptotic results.

Acknowledgements. The last author was partially supported by NSF grant CCF-2312242.

References

[ABFK16] Carlos Aguilar-Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. XPIR: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies*, 2016(2):155–174, April 2016.

[ACLS18] Sebastian Angel, Hao Chen, Kim Laine, and Srinath T. V. Setty. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy*, pages 962–979, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[ALP+21] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. Communication-computation trade-offs in PIR. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 1811–1828. USENIX Association, August 11–13, 2021.

[Amb97] Andris Ambainis. Upper bound on communication complexity of private information retrieval. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *ICALP 97: 24th International Colloquium on Automata, Languages and Programming*, volume 1256 of *Lecture Notes in Computer Science*, pages 401–407, Bologna, Italy, July 7–11, 1997. Springer Berlin Heidelberg, Germany.

[AS16] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *OSDI 16*, pages 551–569, 2016.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers,

and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1292–1303, Vienna, Austria, October 24–28, 2016. ACM Press.

[BHW19] Elette Boyle, Justin Holmgren, and Mor Weiss. Permuted puzzles and cryptographic hardness. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 465–493, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland.

[BI01] Amos Beimel and Yuval Ishai. Information-theoretic private information retrieval: A unified construction. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP 2001: 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 912–926, Heraklion, Crete, Greece, July 8–12, 2001. Springer Berlin Heidelberg, Germany.

[BIKM99] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. One-way functions are essential for single-server private information retrieval. In *31st Annual ACM Symposium on Theory of Computing*, pages 89–98, Atlanta, GA, USA, May 1–4, 1999. ACM Press.

[BIKR02] Amos Beimel, Yuval Ishai, Eyal Kushilevitz, and Jean-François Raymond. Breaking the $O(n^{1/(2k-1)})$ barrier for information-theoretic private information retrieval. In *43rd Annual Symposium on Foundations of Computer Science*, pages 261–270, Vancouver, BC, Canada, November 16–19, 2002. IEEE Computer Society Press.

[BIM00] Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 55–73, Santa Barbara, CA, USA, August 20–24, 2000. Springer Berlin Heidelberg, Germany.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 662–693, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[BKW17] Dan Boneh, Sam Kim, and David J. Wu. Constrained keys for invertible pseudorandom functions. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part I*, volume 10677 of *Lecture Notes in Computer Science*, pages 237–263, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[BR06] Mihir Bellare and Phillip Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 409–426, St. Petersburg, Russia, May 28 – June 1, 2006. Springer Berlin Heidelberg, Germany.

[BS80] Jon Louis Bentley and James B Saxe. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science*, pages 41–50, Milwaukee, Wisconsin, October 23–25, 1995. IEEE Computer Society Press.

[CHK22] Henry Corrigan-Gibbs, Alexandra Henzinger, and Dmitry Kogan. Single-server private information retrieval with sublinear amortized time. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part II*, volume 13276 of *Lecture Notes in Computer Science*, pages 3–33, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017: 15th Theory of Cryptography Conference, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 694–726, Baltimore, MD, USA, November 12–15, 2017. Springer, Cham, Switzerland.

[CK20] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 44–75, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

[CMS99] Christian Cachin, Silvio Micali, and Markus Stadler. Computationally private information retrieval with polylogarithmic communication. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 402–414, Prague, Czech Republic, May 2–6, 1999. Springer Berlin Heidelberg, Germany.

[CSM+20] Raymond Cheng, William Scott, Elisaweta Masserova, Irene Zhang, Vipul Goyal, Thomas Anderson, Arvind Krishnamurthy, and Bryan Parno. Talek: Private group messaging with hidden access patterns. In *Annual Computer Security Applications Conference*, pages 84–99, 2020.

[Dev] Protecting your device information with private set membership. `https://security.googleblog.com/2021/10/protecting-your-device-information-with.html`.

[Dev86] Luc Devroye. Non-uniform random variate generation, 1986.

[DG15] Zeev Dvir and Sivakanth Gopi. 2-server PIR with sub-polynomial communication. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th Annual ACM Symposium on Theory of Computing*, pages 577–584, Portland, OR, USA, June 14–17, 2015. ACM Press.

[Din20] Itai Dinur. On the streaming indistinguishability of a random permutation and a random function. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 433–460, Zagreb, Croatia, May 10–14, 2020. Springer, Cham, Switzerland.

[DMO00] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 122–138, Bruges, Belgium, May 14–18, 2000. Springer Berlin Heidelberg, Germany.

[FLLP24] Ben Fisch, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. ThorPIR: Single server PIR via homomorphic thorp shuffles. Cryptology ePrint Archive, Report 2024/482, 2024.

[GCM+16] Trinabh Gupta, Natacha Crooks, Whitney Mulhern, Srinath Setty, Lorenzo Alvisi, and Michael Walfish. Scalable and private media consumption with popcorn. In *NSDI 16*, pages 91–107, 2016.

[GG15] Shoni Gilboa and Shay Gueron. Distinguishing a truncated random permutation from a random function. Cryptology ePrint Archive, Report 2015/773, 2015.

[GG21] Shoni Gilboa and Shay Gueron. The advantage of truncated permutations. *Discrete Applied Mathematics*, 294:214–223, 2021.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *25th Annual Symposium on Foundations of Computer Science*, pages 464–479, Singer Island, Florida, October 24–26, 1984. IEEE Computer Society Press.

[GGM18] Shoni Gilboa, Shay Gueron, and Ben Morris. How many queries are needed to distinguish a truncated random permutation from a random function? *Journal of Cryptology*, 31(1):162–171, January 2018.

[GH19] Craig Gentry and Shai Halevi. Compressible FHE with applications to PIR. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 438–464, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland.

[GLM16] Matthew Green, Watson Ladd, and Ian Miers. A protocol for privately reporting ad impressions at scale. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 1591–1601, Vienna, Austria, October 24–28, 2016. ACM Press.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th Annual ACM Symposium on Theory of Computing*, pages 197–206, Victoria, BC, Canada, May 17–20, 2008. ACM Press.

[GR05] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *ICALP 2005: 32nd International Colloquium on Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 803–815, Lisbon, Portugal, July 11–15, 2005. Springer Berlin Heidelberg, Germany.

[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.

[HHCG+23] 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. In *Usenix Security*, volume 23, 2023.

[HWKS98] Chris Hall, David Wagner, John Kelsey, and Bruce Schneier. Building PRFs from PRPs. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 370–389, Santa Barbara, CA, USA, August 23–27, 1998. Springer Berlin Heidelberg, Germany.

[IR89] Russell Impagliazzo and Steven Rudich. Limits on the provable consequences of one-way permutations. In *21st Annual ACM Symposium on Theory of Computing*, pages 44–61, Seattle, WA, USA, May 15–17, 1989. ACM Press.

[KC21] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 875–892. USENIX Association, August 11–13, 2021.

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th Annual Symposium on Foundations of Computer Science*, pages 364–373, Miami Beach, Florida, October 19–22, 1997. IEEE Computer Society Press.

[KO00] Eyal Kushilevitz and Rafail Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 104–121, Bruges, Belgium, May 14–18, 2000. Springer Berlin Heidelberg, Germany.

[LMW23] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Doubly efficient private information retrieval and fully homomorphic RAM computation from ring LWE. In Barna Saha and Rocco A. Servedio, editors, *55th Annual ACM Symposium on Theory of Computing*, pages 595–608, Orlando, FL, USA, June 20–23, 2023. ACM Press.

[LP23a] Arthur Lazzaretti and Charalampos Papamanthou. Near-optimal private information retrieval with preprocessing. In Guy N. Rothblum and Hoeteck Wee, editors, *TCC 2023: 21st Theory of Cryptography Conference, Part II*, volume 14370 of *Lecture Notes in Computer Science*, pages 406–435, Taipei, Taiwan, November 29 – December 2, 2023. Springer, Cham, Switzerland.

[LP23b] Arthur Lazzaretti and Charalampos Papamanthou. TreePIR: Sublinear-time and polylog-bandwidth private information retrieval from DDH. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 284–314, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

[LP24] Arthur Lazzaretti and Charalampos Papamanthou. Single pass client-preprocessing private information retrieval. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*, Philadelphia, PA, USA, August 14–16, 2024. USENIX Association.

[LR88] Michael Luby and Charles Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal on Computing*, 17(2), 1988.

[MCR21] Muhammad Haris Mughees, Hao Chen, and Ling Ren. OnionPIR: Response efficient single-server PIR. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2292–2306, Virtual Event, Republic of Korea, November 15–19, 2021. ACM Press.

[Men19] Bart Mennink. Linking stam's bounds with generalized truncation. In Mitsuru Matsui, editor, *Topics in Cryptology – CT-RSA 2019*, volume 11405 of *Lecture Notes in Computer Science*, pages 313–329, San Francisco, CA, USA, March 4–8, 2019. Springer, Cham, Switzerland.

[MK22] Rasoul Akhavan Mahdavi and Florian Kerschbaum. Constant-weight PIR: Single-round keyword PIR via constant-weight equality operators. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 1723–1740, Boston, MA, USA, August 10–12, 2022. USENIX Association.

[MOT+11] Prateek Mittal, Femi G. Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. PIR-tor: Scalable anonymous communication using private information retrieval. In *USENIX Security 2011: 20th USENIX Security Symposium*, San Francisco, CA, USA, August 8–12, 2011. USENIX Association.

[MR14] Ben Morris and Phillip Rogaway. Sometimes-recurse shuffle - almost-random permutations in logarithmic expected time. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 311–326, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.

[MW22] Samir Jordan Menon and David J. Wu. SPIRAL: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy*, pages 930–947, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press.

[MZRA22] Yiping Ma, Ke Zhong, Tal Rabin, and Sebastian Angel. Incremental offline/online PIR. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 1741–1758, Boston, MA, USA, August 10–12, 2022. USENIX Association.

[OS07] Rafail Ostrovsky and William E. Skeith, III. A survey of single-database private information retrieval: Techniques and applications (invited talk). In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007: 10th International Conference on Theory and Practice of Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 393–411, Beijing, China, April 16–20, 2007. Springer Berlin Heidelberg, Germany.

[PPY18] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Private stateful information retrieval. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018: 25th Conference on Computer and Communications Security*, pages 1002–1019, Toronto, ON, Canada, October 15–19, 2018. ACM Press.

[PSY23] Sarvar Patel, Joon Young Seo, and Kevin Yeo. Don't be dense: Efficient keyword PIR for sparse databases. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 3853–3870, Anaheim, CA, USA, August 9–11, 2023. USENIX Association.

[PT20] Jeongeun Park and Mehdi Tibouchi. SHECS-PIR: Somewhat homomorphic encryption-based compact and scalable private information retrieval. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020: 25th European Symposium on Research in Computer Security, Part II*, volume 12309 of *Lecture Notes in Computer Science*, pages 86–106, Guildford, UK, September 14–18, 2020. Springer, Cham, Switzerland.

[PY22] Giuseppe Persiano and Kevin Yeo. Limits of preprocessing for single-server PIR. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *33rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2522–2548, Virtual Conference / Alexandria, VA, USA, January 9–12, 2022. ACM-SIAM.

[RMS24] Ling Ren, Muhammad Haris Mughees, and I Sun. Simple and practical amortized sublinear private information retrieval using dummy subsets. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024: 31st Conference on Computer and Communications Security*, pages 1420–1433, Salt Lake City, UT, USA, October 14–18, 2024. ACM Press.

[SACM21] Elaine Shi, Waqar Aqeel, Balakrishnan Chandrasekaran, and Bruce M. Maggs. Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part IV*, volume 12828 of *Lecture Notes in Computer Science*, pages 641–669, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

[Sta78] Adriaan Johannes Stam. Distance between sampling with and without replacement. *Statistica Neerlandica*, 32(2):81–91, 1978.

[Yek08] Sergey Yekhanin. Towards 3-query locally decodable codes of subexponential length. *Journal of the ACM (JACM)*, 55(1):1–16, 2008.

[Yeo23] Kevin Yeo. Lower bounds for (batch) PIR with private preprocessing. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 518–550, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

[ZLTS23] Mingxun Zhou, Wei-Kai Lin, Yiannis Tselekounis, and Elaine Shi. Optimal single-server private information retrieval. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part I*, volume 14004 of *Lecture Notes in Computer Science*, pages 395–425, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

Scheme Name	Crypto Assumption	Client Storage	Servers	Query Comm.		Query Time		Update Time		Update Comm.	
				Upload	Download	Client	Server	Worst-case	Amortized	Worst-case	Amortized
CHK [CHK22]	LWE	$\lambda B(\lambda w+q)$	1	$\lambda n/w$	λB	$\lambda^2 w+\lambda n/w$	$\lambda n/w$	$\lambda(\lambda w+q)$	$\lambda(\lambda w+q)$	$B+\log n$	$B+\log n$
SACM [SACM21]	LWE	$\lambda B(\lambda w+q)$	2	$\lambda\text{polylog}(n)$	λB	$\lambda^2 w$	$\lambda n/w$	$\lambda(\lambda w+q)$	$\lambda(\lambda w+q)$	$B+\log n$	$B+\log n$
ZLTS [ZLTS23]	LWE	$\lambda B(\lambda w+q)$	1	$\lambda\text{polylog}(n)$	λB	$\lambda^2 w$	$\lambda n/w$	$\lambda(\lambda w+q)$	$\lambda(\lambda w+q)$	$B+\log n$	$B+\log n$
LP [LP23a]	LWE	$\lambda B(\lambda w+q)$	1	$\lambda\text{polylog}(n)$	λB	$\lambda^2 w$	$\lambda n/w$	$\lambda(\lambda w+q)$	$\lambda(\lambda w+q)$	$B+\log n$	$B+\log n$
Piano [ZPZS24]	DDH	$\lambda B(w+q)$	1	n/w	B	$\lambda w+n/w$	n/w	λn	$\lambda\log n$	Bn	$B\log n$
TreePIR [LP23b]	DDH	$\lambda B(w+q)$	2	$\log n$	B	λw	n/w	$\lambda(w+q)$	$\lambda(w+q)$	$B+\log n$	$B+\log n$
GZS [GZS24]	DDH	$\lambda B(w+q)$	1	$\lambda\sqrt{n/w}$	λB	$\lambda^2 w+\lambda n/w$	$\lambda n/w$	$\lambda^2(w+q)$	$\lambda^2(w+q)$	$B+\log n$	$B+\log n$
Piano [ZPZS24]	OWF	$\lambda B(w+q)$	1	n/w	Bn/w	$\lambda w+n/w$	n/w	λn	$\lambda\log n$	Bn	$B\log n$
TreePIR [LP23b]	OWF	$\lambda B(w+q)$	2	$\log n$	Bn/w	λw	n/w	$\lambda(w+q)$	$\lambda(w+q)$	$B+\log n$	$B+\log n$
RMS [RMS24]	OWF	$B(\lambda w+q)$	1	n/w	B	$\lambda w+n/w$	n/w	$\lambda w+q$	$\lambda w+q$	$B+\log n$	$B+\log n$
GZS [GZS24]	OWF	$\lambda B(w+q)$	1	$\lambda\sqrt{n/w}$	$\lambda B\sqrt{n/w}$	$\lambda^2 w+\lambda n/w$	$\lambda n/w$	$\lambda^2(w+q)$	$\lambda^2(w+q)$	$B+\log n$	$B+\log n$
Plinko	OWF	$B(\lambda w+q)$	1	n/w	B	$\lambda\log n+n/w$	n/w	$\lambda\log n$	$(\lambda+q/w)\log n$	$B+\log n$	$B+\log n$
Plinko-Piano	OWF	$\lambda B(w+q)$	1	n/w	Bn/w	$\lambda\log n+n/w$	n/w	$\lambda\log n$	$\lambda(1+q/w)\log n$	$B+\log n$	$B+\log n$

Figure 6: Table comparing existing single-server offline/online PIR schemes, for a client querying a size n database with B-bit entries. We parameterize the schemes for a PIR scheme with λ-bits of security that refreshes its hints after q queries, and uses sets of size n/w (i.e. uses blocks of size w). In all entries, we hide constant and polylog factors.

[ZPZS24] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. Piano: Extremely simple, single-server PIR with sublinear server computation. In *2024 IEEE Symposium on Security and Privacy*, pages 4296–4314, San Francisco, CA, USA, May 19–23, 2024. IEEE Computer Society Press.

A Plinko Pseudocode

In this section, we show how we can use invertible PRFs (iPRFs) to improve the client-side computation for queries and updates in the construction from [RMS24]. We present the pseudocode for our modified construction in Figure 7. This construction is meant to complement the main body of Section 5, which has a more digestible description of the scheme.

We also provide figure Figure 6, which compares Plinko to other offline-online PIR schemes across many performance metrics.

B Plinko-Piano

In this section, we show how we can use invertible PRFs (iPRFs) to improve the client-side computation for queries and updates in Piano [ZPZS24]. We present the pseudocode for our modified construction in Figure 8.

The main idea in Piano [ZPZS24] is to use sets of size n/w, where each element is some offset for a "block" of w elements in the database. The client is allowed to stream the database in the offline phase and to refresh their hints. Using a PRF to compute these offsets allows for a compressed representation of the hint sets and allows the client to efficiently compute their hint parities by only streaming a n/w block of the database at a time.

In the online phase, the client finds a hint set with the queried index, removes the corresponding offset, and sends the remaining $n/w-1$ offsets to the server. The server responds with n/w possible parities, one for each possible skipped block. Finally, the client uses their hint parity and the server's response to compute the value of the database at the queried index.

By replacing PRFs with iPRFs, we can improve the scheme. First, in the offline phase, we can efficiently compute hints without streaming blocks at a time and instead streaming the database one element at a time (and possibly out of order). Additionally and more importantly, using iPRFs improves the client's run-time when they issue a query. With standard PRFs, the client performs a linear pass over their hint sets to find

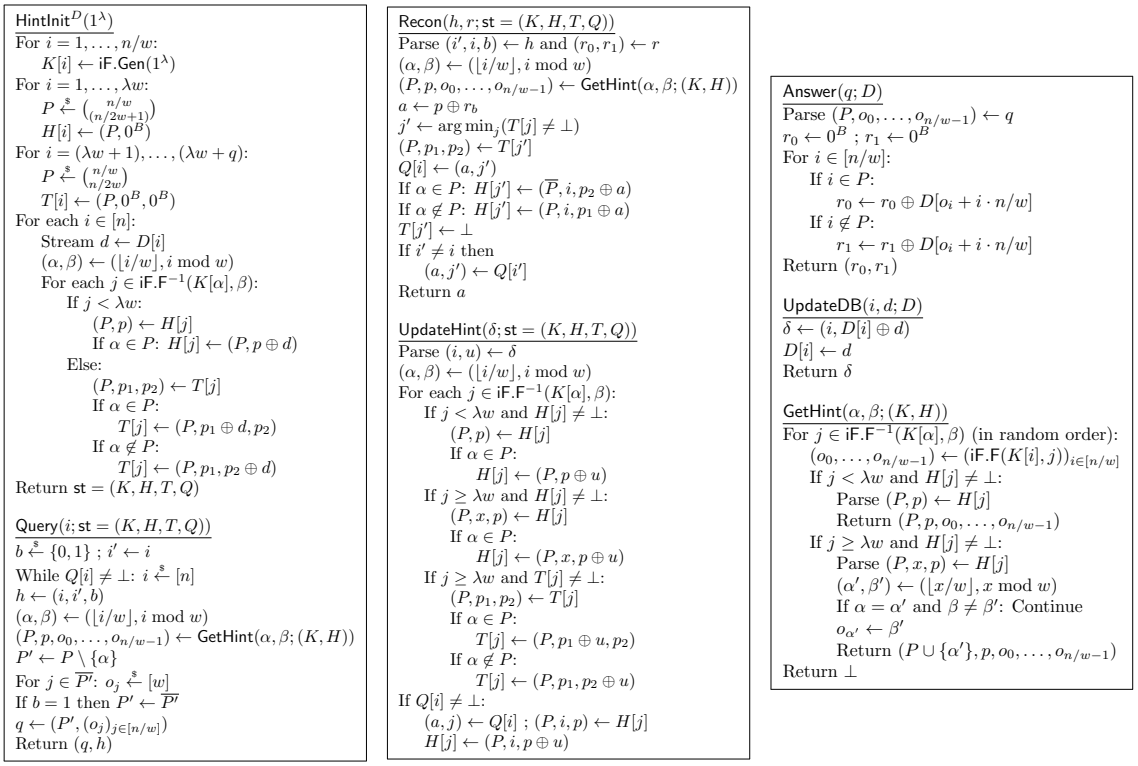

Figure 7: Pseudocode for a variant of RMS [RMS24] with efficient updates by using invertible PRFs. The client uses an iPRF iF from $[\lambda w + q]$ to $[w]$.

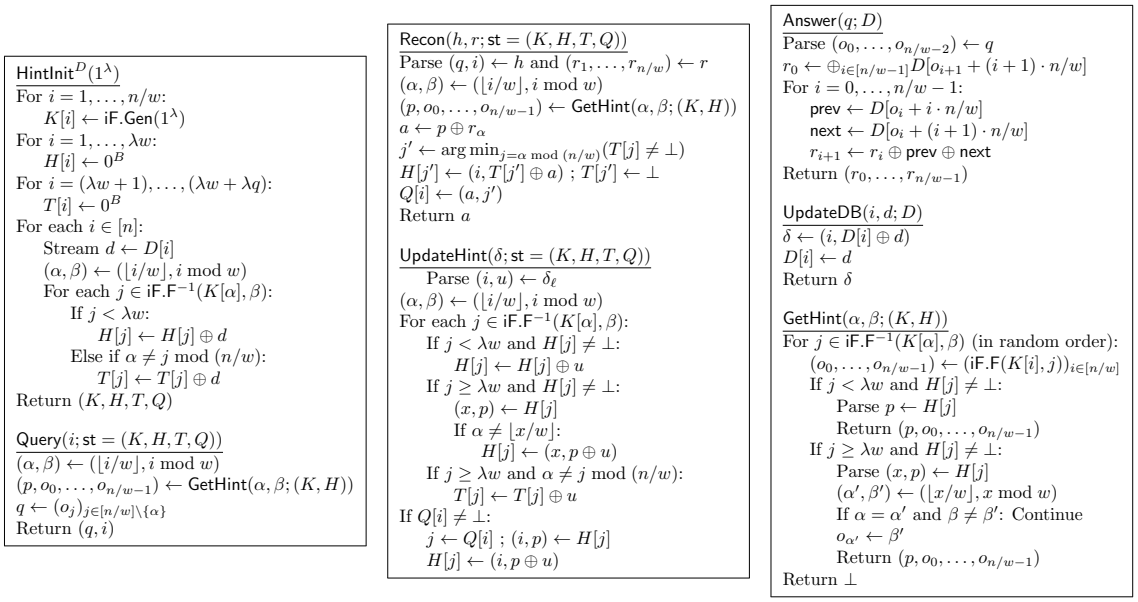

Figure 8: Simplified pseudocode for a variant of Piano [ZPZS24] with efficient updates by using invertible PRFs. The client uses an iPRF iF from $[\lambda w + \lambda q]$ to $[w]$.

one with the queried element. But iPRFs give a more efficient hint searching algorithm. In particular, a simple (efficient) inversion of the iPRF will give pointers to all of the hints that contain the queried element.

Finally, the original Piano paper [ZPZS24] proposed using the Bentley-Saxe transform [BS80] to perform updates more efficiently. Unfortunately, this method requires additional communication between the client and the server and is only efficient when amortized. In the worst-case, an update could require a client to re-run the entire offline phase with all n database items! However, a side-effect of our efficient hint searching algorithm is an efficient way to update the client's hints whenever an update occurs. This method only requires logarithmic communication (the server sending the change and location to the client) and runs in logarithmic time (a single iPRF inversion) for the client. Further, this communication and computation is worst-case and doesn't require amortization.

Security and Correctness. Since our modification of Piano [ZPZS24] is strictly an algorithmic improvement, we omit proofs of correctness and security. The proofs from the original work are sufficient to establish the desired properties. The communication between the client and server is identical across the two versions. As long as the iPRF satisfies its own security, efficiency, and correctness, then the hint selection and distribution will also be preserved.

Note that for the security of the PIR, we only need that the iPRF satisfied *normal PRF* security because the server never observes outputs of the inversion oracle. As long as the output of the client algorithms are identical, security is unaffected by how the algorithms work.