Minicrypt PIR for Big Batches

Nico Döttling[1], Jesko Dujmovic[1,2], Julian Loss[1],
Maciej Obremski[3]

[1] CISPA Helmholtz Center for Information Security
jesko.dujmovic,doettling,loss@cispa.de
[2] Saarland University
[3] National University of Singapore
obremski.math@gmail.com

Abstract. We present PIR protocols for offline/online two-server setting where a client C wants to privately retrieve a batch of entries from database of size N by interacting with a servers S_1. The client has interacted with a server S_2 ahead of time, not colluding with S_1. We present simple protocols based on one-way functions that substantially improve on the query complexity or runtime over existing works. Concrete instantiations of our general paradigm lead to batch PIR protocols with the following parameters:

- A protocol for batches of $\sqrt{N}$, where C, S_1, and S_2 each spend a total of $\tilde{O}(N)$ work and exchange $\tilde{O}(\sqrt{N})$ bits of communication. This yields an amortized complexity of $\tilde{O}(\sqrt{N})$ work and $\tilde{O}(1)$ communication per query in the batch.
- A more balanced protocol for batches of size $N^{1/3}$ in which C spends a total of $\tilde{O}(N^{2/3})$ work, S_1 and S_2 spend $\tilde{O}(N)$ work, and the total communication is of size $\tilde{O}(N^{2/3})$.

Our protocols have immediate applications such as Private Set Intersection (PSI) in the two-server setting with preprocessing and unbalanced set sizes.

1 Introduction

Private Information Retrieval (PIR) [CGKS95] is a cryptographic tool that allows a client C to retrieve an entry $\mathsf{DB}[i]$ from a server S holding a database DB without revealing to S the index i which is being retrieved. PIR plays a crucial role in many privacy-sensitive applications such as private certificate lookup, private contact discovery [BDG15, DRRT18], private DNS [SCH+21], and many more. Two of the central performance metrics of PIR protocols are 1) the amount of communication that has to be exchanged between the client C and the server S and 2) the amount of computation that both C and S have to perform. Denoting as N the number of entries stored in DB, one can easily infer various non-triviality requirements with respect to both communication and computation. For example, the communication between C and S must be $o(N)$, as otherwise, there exists a trivial PIR protocol in which the client downloads the entirety of DB. Another folklore result, which was first rigorously proven

by Beimel, Ishai, Malkin [BIM00], states that in any PIR protocol in which the server S stores DB in the clear, S must traverse all of DB in order to ensure privacy (and therefore in incurring $O(N)$ computation). As shown by Beimel et al., this lower bound holds even if the total computation is spread across two or more servers, although in this case, the servers can split the work. Intuitively, this is true because if S does not touch an entry $\mathsf{DB}[i]$ while executing the protocol, it knows that the C could not possibly have queried this entry.

Fortunately, the work of Beimel et al. [BIM00] offers a way out of this quagmire. Namely, their key idea is to split a PIR protocol into two phases, an *offline* phase which is independent of any queries C might ask and an *online* phase. In this manner, it indeed becomes possible to reduce the amount of online time required by the server S to $o(N)$. A recent line of works on offline/oline PIR initiated by the beautiful results of Corrigan-Gibbs and Kogan [CK20] shows how to achieve practical protocols in this setting which incur minimal storage on the server's side. In these works, the client communicates with two non-colluding servers, an offline server S_1 and an online server S_2. Inspired by these advances in offline/online PIR, we present the first two-server protocols for the offline/online setting in which the client efficiently retrieves a *batch* of entries from DB.

1.1 Our Results

While processing a batch of queries instead of a single query at a time is a well-known and thoroughly-explored technique in the setting of PIR, it has not been well explored for the two-server offline/online setting. To close this gap, we present two batch-PIR protocols in the semi-honest server setting with the following properties:

- Our first scheme supports batches of size $\sqrt{N}$ while communicating a total of $\tilde{O}(\sqrt{N})$ communication between the client and the servers over both phases. C has to store $\tilde{O}(\sqrt{N})$ bits in both phases and the work (computation cost) is $\tilde{O}(n)$ for C, S_1 and S_2. These costs translate to polylogarithmic per-query communication and server computation in $\tilde{O}(\sqrt{N})$. Client storage, communication and server computation match the lower bounds of [CHK22, Yeo23] and the upper bound of [CK20] for two-server PIR. We improve upon [CK20] in terms of per-query client computation (but still do not match any known lower bound).
- To obtain a protocol with improved client-side computation, we also give a protocol which achieves more balanced communication and computation parameters. This protocol has batch size $N^{1/3}$, upload communication $\tilde{O}(N^{2/3})$, and download communication of $\tilde{O}(N^{1/3})$. The total work for the client shrinks to $\tilde{O}(N^{2/3})$ while the work for the server stays at $\tilde{O}(N)$.

A key novelty of our work over prior batch-PIR protocols is to set the size of query batches to be a function of the database size N. This allows us to amortize the communication and computation costs of our two phases across the number

of queries in the batch and leads to significant per-query savings over all existing works in this setting. Additionally, we also discuss a database private version of our scheme, i.e. a version of the protocol where the client learns nothing about the database beyond the legitimate PIR outputs.

We will now discuss two intriguing applications of our parameter settings, which further motivate the notion of batch PIR.

Application One: Private Search. In recent years, there has been significant progress in private search [SSLD22, HDCGZ23, ABG+24, ZSF24]. The idea behind private search is compelling—imagine a search engine like Google, but where the server host remains unaware of users' queries.

All private search protocols must address several key challenges. One major difficulty arises from the fact that web pages and search queries exist in distinct high-dimensional spaces. The first challenge, therefore, is to map queries into the same space as the pages, ensuring that proximity between a query and a page reflects relevance.

Additionally, these protocols must efficiently retrieve pages that are close to the query's mapped representation. Recent work has approached this problem using variations of batch private information retrieval (PIR). For instance, [HDCGZ23] employs a PIR scheme that retrieves a single entry of size $\sqrt{N}$, while [ZSF24] uses multiple consecutive batch PIRs with much smaller batch sizes. Larger batch sizes in PIR expand the design space for private approximate nearest neighbor search, offering new possibilities for more efficient private search protocols.

Application Two: Two-Server Unbalanced PSI with Preprocessing. Private Set Intersection (PSI) allows two or more parties to secure compute the intersection of two sets, one held by a sender and one by a receiver, such that the receiver learns the intersection of the two sets but nothing more about the sender's set. PSI is a very active area of research due to its immediate usefulness for many practical applications [KS05, PSZ14, HV17, RR17, KMP+17, PSWW18, ABD+21, ALOS22].

Our first application is a novel two-server private set intersection (PSI) protocol in the two server setting with preprocessing described earlier. Very briefly, in our PSI setting, the servers hold a set $\mathcal{S}_S$ and the receiver holds a set $\mathcal{S}_C$ and we would like to let the receiver learn the intersection $S \cap T$. We are interested in a setting where $|\mathcal{S}_S| \gg |\mathcal{S}_C|$, e.g. if $|\mathcal{S}_C| = O(\sqrt{|\mathcal{S}_S|})$.

All we need to achieve the above construction is the database-private PIR protocol. PIR guarantees privacy of the client i.e. that the server will not learn which entries of the database are retrieved. The database-private PIR has an additional guarantee protecting server- namely that the semi-honest client will not learn anything else about the database except the points he intended to query. Given this protocol, the server S holding $\mathcal{S}_S$ builds database storing 1 or 0 in each cell depending on if the corresponding element is in the set $\mathcal{S}_S$. After the preprocessing phase, all the client C has to do is to send a batch query corresponding to all the elements in $\mathcal{S}_C$.

The database-private version of our first PIR protocol with batch size $|\mathcal{S}_C|$ directly yields a PSI protocol in which C communicates only $|\mathcal{S}_C|$ bits while relying only on minicrypt primitives. This type of PSI protocol (for the semi-honest setting) can immediately be obtained by using our server-private version of batch PIR.

Application Three: Private Statistics on large Sub-Populations. A second, more direct application of our PIR protocols with large batch parameters is to perform statistics on a sub-sample of a very large data pool in a privacy preserving manner. As a simple example, a medical survey may be interested in data points from individuals within a specific group according to gender, age, diet, etc. Such sub-populations may be much smaller than the total number N of all entries within a large medical database, yet still large enough to warrant batch sizes which behave as a function of n, e.g. $\sqrt{N}$.

As a concrete concrete example, consider a medical database with $\approx 10^10$ entries, which is the order of magnitude for one entry per every human on earth. For a database of this scale, our overhead in terms of communication and computation is of order $\sqrt{10^{10}} = 10^5 = 100000$, which is a manageable work- and communication overhead.

1.2 Our Techniques

Offline-Online PIR. Our constructions follow the general approach established by the beautiful work of Corrigan-Gibbs and Kogan [CK20]. In the framework of [CK20] there a two non-communcating servers, an *offline server* S_1 and an *online server* S_2. In the offline-phase a client interacts with the offline-serve and obtains a set of *hints* depending on server's database DB. Importantly, this offline-phase is independent of the client's inputs. Furthermore, the total size of the hints is sublinear in N, the size of the server's database. In the online phase, the client receives his input, namely an index q of a database item he wants to retrieve. The client now retrieves a suitable hint h (depending on q) and uses it to generate a message c to the online server, who processes this message using the database DB and sends the response r back to the client. From this response r as well as additional information provided in the hint h the client can now recover the database item $\mathsf{DB}[q]$.

In terms of security, we require that the message c hides the index q from the online server. Naturally, as the offline-phase is independent of the clients inputs, the offline server *by itself* learns nothing about the clients queries either. However, critically, privacy of the client's queries additionally rests on the assumption that the offline and online servers do not communicate.

It is this non-collusion requirement which facilitates the design of very efficient two-server PIR protocols from symmetric key assumptions, that is without resorting to public-key primitive.

The CK20 Protocol. In [CK20] this framework is instantiated as follows. In the offline-phase, the client generates k pseudorandom sets $\mathcal{S}_1, \ldots, \mathcal{S}_k \subseteq [N]$ of size $\approx \sqrt{N}$, and sends these sets to the offline server. For each set $\mathcal{S}_j$, the offline

server computes the parity $p_j = \oplus_{x \in \mathcal{S}_j} \mathsf{DB}[x]$ and sends $p_1, \ldots, p_k$ back to the client, who stores the pairs $(\mathcal{S}_j, p_j)$. In order to reduce the storage overhead, all the sets $\mathcal{S}_j$ are *random shifts* of a random set $\mathcal{S}$. More concretely, for each index j the client chooses a random shift $\Delta_j \in [n]$ and sets $\mathcal{S}_j = \{x + \Delta_j \mod n \mid x \in \mathcal{S}\}$. Consequently, the client only has to store the set $\mathcal{S}$ as well as the shifts $\Delta_1, \ldots, \Delta_k$.

The online phase now proceeds as follows. Given a a query $q \in [n]$, the client searches for a a pair $(\mathcal{S}_j, p_j)$ in its preprocessing for which $q \in \mathcal{S}_j$. Now the receiver *punctures* $\mathcal{S}_j$ at q, i.e. it computes $\bar{\mathcal{S}} = \mathcal{S}_j \setminus \{q\}$ and sends $\bar{\mathcal{S}}$ to the online server. The online server computes the parity $p = \oplus_{x \in \bar{\mathcal{S}}} \mathsf{DB}[x]$ and sends p back to the client, who can now recover $\mathsf{DB}[q]$ by computing

$$p_j \oplus p = (\oplus_{x \in \mathcal{S}_j} \mathsf{DB}[x]) \oplus (\oplus_{x \in \mathcal{S}_j \setminus \{q\}} \mathsf{DB}[x]) = \mathsf{DB}[q].$$

While this protocol is correct, there is a subtle issue concerning query privacy: The punctured set $\bar{\mathcal{S}}$ actually does leak information about the query q to the online server, namely the fact that the query q *is not in* $\bar{\mathcal{S}}$. Conversely, we can say that the server *knows* that q lies in the *uncertainty set* $[N] \setminus \bar{\mathcal{S}}$. To compensate for this leakage, [CK20] rely on a careful mechanism which introduces *false positives*, i.e. with a certain small probability the client sends a set $\bar{\mathcal{S}}$ for which $q \in \bar{\mathcal{S}}$. This mechanism in turn causes a correctness error, which is compensated by repetition.

There is a growing body of work which extends upon this paradigm, as a non-exhaustive sample e.g. [SACM21, LP23b, ZLTS23, ZPZS24, MIR23, GZS24, LP23a]. Specifically, these works propose alternative mechanisms to represent the sets $\mathcal{S}_j$ and ensure that the punctured set $\bar{\mathcal{S}} = \mathcal{S}_j \setminus \{q\}$ hides the query q from the server.

Our Approach. The starting point of this work is an attempt to construct sets $\mathcal{S}_j$ for which both

1. membership can be tested very efficiently and
2. punctured sets have a compact representation which does not reveal non-trivial information about the punctured point.

To address the first point, we derive the sets $\mathcal{S}_j$ from pseudorandom permutations. Specifically, assume that $N = 2^{2k}$, i.e. we can represent all elements of the $[N]$ as bit-strings of length $2k$. Consequently, we can identify each element $x \in [N]$ as (L, R) where $L, R \in \{0,1\}^k$. Let Π be a pseudorandom permutation on $\{0,1\}^{2k}$. Our approach is to define the sets $\mathcal{S}_j$ as

$$\mathcal{S}_j = \{\Pi_{k_j}(\Delta, R) \mid R \in \{0,1\}^k\},$$

where $k_j \leftarrow \{0,1\}^\lambda$ is a uniformly chosen key for Π and $\Delta \leftarrow \{0,1\}^k$ is chosen uniformly random. Such sets $\mathcal{S}_j$ have a compact representation, namely in the form of the key k_j and Δ. Furthermore, $\mathcal{S}_j$ admits a very fast membership test. Namely, for a candidate element $x \in [N]$ we can compute $(L, R) \leftarrow \Pi_{k_j}^{-1}(x)$ and test whether $L = \Delta$. As a third point, such sets $\mathcal{S}_j$ are *well-spread*, as due to the

pseudorandomness of Π_{k_j} the set $\mathcal{S}_j$ is indistinguishable from uniform (against distinguishers who do not see the key k_j).

However, there is no efficient way to puncture such a set $\mathcal{S}_j$ without unrolling it into its explicit representation. To facilitate puncturing, we include an additional round of a Feistel permutation. Specifically, let f be a pseudorandom function mapping from $\{0,1\}^k$ to $\{0,1\}^k$. We now define $\mathcal{S}_j$ as

$$\mathcal{S}_j = \{\Pi_{k'}(\Delta \oplus f_{k''}(R), R) \mid R \in \{0,1\}^k\},$$

where again $k' \leftarrow \{0,1\}^\lambda$ is a key for the permutation Π and $k'' \leftarrow \{0,1\}^\lambda$ is a key for the pseudorandom function f. Note that the previous properties still holds, i.e. we can test membership of an $x \in [N]$ by computing $(L, R) = \Pi^{-1}_{k'}(x)$ and testing whether $L \oplus f_{k''}(R) = \Delta$.

Moreover, if f is a puncturable PRF [BGI14], we can puncture such a set. Specifically, by puncturing k'' at R, we obtain a punctured set $\bar{\mathcal{S}}_j$ *which does not contain* x, but where indeed all $\Pi(L', R)$ for $L' \in \{0,1\}^k$ are *potential* elements of $\mathcal{S}_j$. That is, from the view of an observer who only sees the punctured key k^*, any of these 2^k values could have been the point we punctured from $\mathcal{S}_j$. We call these $2^k = \sqrt{N}$ values the *uncertainty set* of $\bar{\mathcal{S}}_j$. Note that the size of this uncertainty set given a punctured key is still a far cry away from the size of uncertainty set of the explicit representation of $\bar{\mathcal{S}}_j$. Specifically, for the explicit representation of $\mathcal{S}_j$ the uncertainty set is $[N] \setminus \mathcal{S}_j$, and thus its size is $N - \sqrt{N}$.

Consequently, as for our approach the uncertainty sets are relatively small there is no hope that a *single* punctured key substantially hides the punctured point q.

Batching. This is where our second central theme comes into play: Batching. By bundling together several queries $q_1, \ldots, q_k$ into a single batch-query $(q_1, \ldots, q_k)$ (and suitably shuffling them by e.g. a random permutation), we can leverage that the uncertainty sets of all punctured keys *mutually conceil* their punctured queries. More broadly, if we can make sure that the union of all uncertainty sets is all of $[N]$ then from the view of the online server the punctured keys could (at least qualitatively) correspond to an arbitrary batch-query. Since each uncertainty set is a (pseudorandom) set of size $\sqrt{N}$, this event will happen with near certainty once $k \geq \lambda\sqrt{N}$.

However, making this basic idea stochastically precise turns out to be the most challenging aspect about our new technique.

To illustrate some of the challenges we face, consider the following naive approach to process a batch-query. Assume the client has a *preprocessing pool* $\mathcal{P}$ consisting of $m \geq k$ pairs of keys and hints $\mathcal{P} = \{(k_1, h_1), \ldots, (k_m, h_m\})$. Assume further the client has k queries $q_1, \ldots, q_k$. In this naive approach the client processes the queries q_i one-by-one. That is, for each query q_i the client retrieves a preprocessing (k_j, h_j) for which $q_i \in \mathcal{S}_{k_j}$ from the pool $\mathcal{P}$ *and removes this preprocessing from* $\mathcal{P}$. After m iterations of this procedure, the client has obtained preprocessings for each query. It then randomly shuffles these queries and sends them to the online server.

While at first glance this procedure seems "reasonable", a moment of reflection reveals a grave issue with this approach: Consider just the first query q_1. Initially the sets $\mathcal{S}_{k_i}$ in the pool are all independent of the queries. However, once we take a set $\mathcal{S}_j$ *which contains* q_1 out of the pool $\mathcal{P}$, then the remaining sets are ever so slightly *biased away* from q_1. In a bit more detail, first recall that the sets $\mathcal{S}_i$ are pseudorandom sets of size $\sqrt{N}$. Hence, for a fixed query q_1 initially the expected number of sets $\mathcal{S}_i$ in $\mathcal{P}$ is which contain q_1 is $k \cdot \sqrt{N}/N = k/\sqrt{N}$. However, after removing a set $\mathcal{S}_j$ from $\mathcal{P}$ which is guaranteed to contain q_1, the expected number of sets in $\mathcal{P}$ which contain q_1 drops to $k/\sqrt{N} - 1$, which is a noticeable difference.

Batch Unbiased Sampling. Hence, we need is a mechanism which matches preprocessings with queries without introducing a bias into the pool of remaining samples. Our core-mechanism to accomplish this will be *sampling with replacement*, a technique inspired by the broken hint technique used in [GZS24]; every time we take a set $\mathcal{S}_j$ out of the pool $\mathcal{P}$ based on the criterion that it contains q_i, we sample a fresh key k under the condition that $\mathcal{S}_k$ contains q_i and insert k back into the pool $\mathcal{P}$. However, as this set $\mathcal{S}_k$ is generated during the online-phase, the client does not know a hint h corresponding to it. Hence, it inserts the *dummy set* $\mathcal{S}_k$ with the *empty hint* $\perp$ into $\mathcal{P}$, i.e. it inserts $(k, \perp)$ into $\mathcal{P}$. This will ensure that the pool $\mathcal{P}$ remains unbiased.

However, this now introduces the issue that the population of dummy keys without hints in $\mathcal{P}$ steadily increases, hence a query q_i may be matched up with a useless dummy set. To compensate for this issue without reintroducing a bias, we need to (potentially) pair up each query with several keys, which introduces a small amount of redundancy.

To deal with these technical issues, our actual batch unbiased sampling mechanism provided in Section 4 is somewhat more involved than described in this outline.

At a very high level, our sampling mechanism proceeds in two stages. In the first stage, which we refer to as the splitting stage, we distribute preprocessed keys to bins corresponding to the queries in a way very similar to the one described above. After puncturing the keys in each bin, the punctured keys are again *dissociated* from the corresponding queries in a merging stage, which indeed results in a list of punctured keys which computationally hides the queries.

The analysis of our batch unbiased sampling mechanism constitutes the technical core of this work, and we think this notion may be interesting in its own right.

Optimizations. We further discuss different trade-offs between batch size, online-client effort, and per-query server effort.

To get different trade-offs between batch size, online-client effort, and per-query server effort, we change our set representations slightly. In particular, we explore a setting where the client computation is sublinear while the server cost remains linear in the database size N. Thus, we want both the set and the uncertainty set to be of size $\tilde{\theta}(N^{2/3})$. To achieve this task, we have to change our set representations.

We use the same pseudorandom permutation g as before, but with a split of $[N]$ into $[N^{2/3}] \times [N^{1/3}]$ using the mapping $\Pi_k(\Delta \oplus L \oplus f_{k'}(R), R)$. With this, our set would be of size $N^{2/3}$ but the uncertainty set is now of size only $N^{1/3}$. Using the approach from above, this would force us to use batches of size $N^{2/3}$ in order for the uncertainty sets in the batch to cover all of $[N]$. Recall that the client needs to search, on average $N^{1/3}$ of them per query q, until it fins a set which contains q. Thus, this results again in $O(N)$ work for the client and we are back to where we started.

Therefore, we instead puncture the key of f at $N^{1/3}$ many random points. Note that due to the way that our sets are represented, each such punctured point increases the uncertainty set by $N^{1/3}$ many points. Thus, overall, the uncertainty sets in this approach grow to size $N^{2/3}$. To keep these sets functional, all but one of the $N^{1/3}$ punctured point already need to be fixed in the pre-processing phase and transmitted to the server during the online phase. Overall, the work and storage requirements of the client reduce, per query, to $O(N^{1/3})$ in both phases and thus to a total of $O(N^{2/3})$ in a query batch of size $O(N^{1/3})$. The communication in this approach increases, however, to $O(N^{1/3})$ per query and thus to $O(N^{2/3})$ in the batch, as the client needs to transmit all of the punctured points to the server.

All the parameters we instantiated in this paragraph can be chosen flexibly. We discuss the exact trade-offs in the Section 3.

Database Privacy. Additionally, we will provide a small modification which will augment our scheme with server privacy, also known as database privacy. That is, a semi-honest client will learn nothing about the database DB beyond the legitimate query responses $\mathsf{DB}[q_1], \ldots, \mathsf{DB}[q_k]$. We formalize this via a standard simulation-based notion, c.f. Definition 7.

Our base scheme is markedly *not* database private; the sets $\mathcal{S}_k$ together with the hints $h = \bigoplus_{i \in \mathcal{S}_k} \mathsf{DB}[i]$ reveal parities about DB.

The transformation to make our scheme server private is very natural. Besides the database DB the offline and online servers also hold a joint PRF key K. The basic idea is now to associate each set key k_i with a randomly chosen nonce $\mathsf{nce}_i \leftarrow \{0,1\}^\lambda$ which is sent alongside k_i to the servers. Both online and offline servers now mask their response with $\mathsf{PRF}_K(\mathsf{nce}_i)$. That is, the offline server computes $h_i = \mathsf{PRF}_K(\mathsf{nce}_i) \oplus \bigoplus_{j \in \mathcal{S}_{k_i}} \mathsf{DB}[j]$ and the online server computes $p_i = \mathsf{PRF}_K(\mathsf{nce}_i) \oplus \bigoplus_{j \in \bar{\mathcal{S}}_{k_i^*}} \mathsf{DB}[j]$. The client can still recover $\mathsf{DB}[q_i]$ by computing

$$\begin{aligned} h_i \oplus p_i &= \mathsf{PRF}_K(\mathsf{nce}_i) \oplus \bigoplus_{j \in \mathcal{S}_{k_i}} \mathsf{DB}[j] \oplus \mathsf{PRF}_K(\mathsf{nce}_i) \oplus \bigoplus_{j \in \bar{\mathcal{S}}_{k_i^*}} \mathsf{DB}[j] \\ &= \mathsf{DB}[q_i]. \end{aligned}$$

For dummy queries, we will set the nonces nce_i to fresh random value, consequently the client will not learn anything from the corresponding masked parities.

1.3 Related Work

To get an excellent overview of single-query PIR we recommend reading the excellent related work sections of [CK20, ZPZS24].

Since the emergence of [CK20] there have been a few iterations on their template. We give a short overview over the two server protocols is only based on one-way functions. The set representation of TreePIR [LP23b] has a size that is polylogarithmic in n, but the punctured representation does not allow the server to uniquely determine the punctured set. Therefore, the server has to communicate $\sqrt{N}$ bits per query. In [ZPZS24, MIR23, HPPY24] the server can uniquely determine the punctured set. Therefore, the server-to-client communication is constant but the punctured set representation is of size $\sqrt{N}$, so the per query communication is still $\sqrt{N}$. [GZS24] interpolate between the two settings with a protocol where the punctured set has a representation size $N^{1/4}$ and the server can compute $N^{1/4}$ candidates of punctured set, where one of them will be correct. This way they arrive at a protocol that has client-to-server communication of $N^{1/4}$ and a server-to-client communication of $N^{1/4}$. While each of the protocols has advantages of their own, [GZS24] has the best communication complexity as a two-server offline-online private information retrieval protocol, which is why we compare against them. [GZS24] works particularly well in our setting because after $\sqrt{N}$ queries they have to redo the preprocessing and in our setting this is where the protocol end. Therefore, no trick is required to amortize the preprocessing.

In this section, we will focus on batch PIR with big batches. We will assume a batch size of $\sqrt{N}$ for the rest of this section, though the same points can be made about $N^{1/3}$.

There are a few approaches to batch PIR. One approach is to simply run multiple instances of a single-query PIR protocol. When applying this approach to non-preprocessing PIR schemes the server computation becomes unbearable, as the server computation per query is linear in the database size. However, for protocols where the server computation is sublinear in the database size, this approach can be viable if the preprocessing can be reused. If we consider batch sizes of size $\sqrt{N}$, then any PIR with $\sqrt{N}$ per-query communication complexity is beaten out by the trivial solution of downloading the entire database.

Another approach is to apply a batch code or a weaker variant called probabilistic batch code [IKOS04, SWP09, LS15, SG16, ACLS18] to the database and then use the single-query PIR protocol on smaller databases. Simplfied, a (probabilistic) batch code has four parameters (n, N, k, m) it take as string $x \in \{0,1\}^n$ and encodes it into m strings $y_1, \ldots, y_m \in \{0,1\}^*$. It has the propery that whp. one can recover any k-sized subsequence $x_{i_1}, \ldots, x_{i_k}$ by querying one element from each y_i. Finally, N is the size of all y_i together. For our setting we use a $(n, O(n), O(\sqrt{N}), \sqrt{N})$ batch code, which splits the database DB into $O(\sqrt{N})$ many strings $y_1, \ldots, y_{\sqrt{N}}$ and then run a single-query PIR protocol on each of them. The batch code then guarantees that whp. one can recover any $\sqrt{N}$ entries of DB by querying one element from each y_i. One can combine batch codes with any single-query PIR protocol. While this approach improves the parameters of

a single-query PIR protocol, the parameters of our solution are better in communication or computation complexity unless one is willing to use public-key assumptions.

A notable example is in the offline/online setting are programmable distributed point functions (pDPFs) [BGIK22] which achieve a similar query complexity and client storage as we do. For $1/\mathsf{poly}$ privacy error the scheme is quite efficient, however, their amplification to a negligible privacy error is quite inefficient. This amplification causes makes the complexity of the programmable distributed point function to be at least be quadratic in its codomain. This problem only gets slightly reduced if we combine them with batch codes [IKOS04] to adapt this protocol to the batch PIR setting. For a database of 2^{35} bits or roughly 4GB, batch size of $\sqrt{2^{35}}$, statistical security parameter $\kappa = 40$ and computational security parameter $\lambda = 128$, their client has a runtime of ca. 2^{119} bit operations, which far from practical. In the same scenario our client needs to make ca. 2^{30} bit operations. These high computational costs are not only concrete but also asymptotic, see Figure 1.

One can also use specific structure of PIR protocols improve their batching. For example [GKL10] modify the single-query PIR of [GR05] to improve its batching. Similarly, when using a fully homomorphic encryption [Gen09, BV11, GSW13] scheme to implement PIR, one can use a single circuit to make all $\sqrt{N}$ queries at once reducing the server-sided computation from $O(N)$ per query to $O(N)$ total. None of these protocols are particularly practical, as they heavily rely on public-key cryptography. Finally, in a recent breakthrough Lin, Mook and Wichs [LMW23] provided the first single server PIR protocol with preprocessing which achieves sublinear, in fact polylogarithmic online overhead for the server. Their scheme is based on the Ring-LWE assumption with superpolynomial modulus-to-noise ratio [LPR10].

2 Preliminaries and Definitions

We introduce some basic definitions and notations. We define $[n] = \{1, \ldots, n\}$. Usually denote a negligible function by negl, which means that for every positive polynomial $p(.)$ and all but finitely many n we have $\mathsf{negl}(n) < 1/p(n)$. Throughout, let λ denote the security parameter. We use calligraphic letters $\mathcal{S}, \mathcal{R}$ to denote sets and write $x \leftarrow_{\$} \mathcal{S}$ to denote that x is sampled uniformly from $\mathcal{S}$. We write $\mathbb{P}_{=\ell}(\mathcal{R})$ for the set of all subset of $\mathcal{R}$ of size ℓ. We assume the standard notions of PPT algorithms and write algorithms using serif-free letters, i.e., A. We write $y \leftarrow \mathsf{A}(x)$ to denote that (probabilistic) algorithm A returns y on input x. If A has access to an oracle O at runtime, we write A^{O}.

Multiplicative Chernoff Bound. Suppose $X_1, \ldots, X_n$ be n independent random variables with a range $\{0, 1\}$. We call their sum X and the expected value of the sum μ. Then for $0 < \delta < 1$ we can bound

$$\Pr[X \geq (1+\delta)\mu] \leq \left(\frac{e^{\delta}}{(1+\delta)^{1+\delta}}\right)^{\mu}$$

	Online Work	Online Communication	Storage	Assum.
Batch Size $\sqrt{N}$				
PIR + BC*	Client: $(\lambda+t)\sqrt{N}$ Server:Nt	$(\lambda+t)\sqrt{N}$	0	PIR
[LMW23]*	Client: $\sqrt{N}t$ Server: $\sqrt{N}t$	$\sqrt{N}t$	0	RLWE
[CK20] Thm. 14 + BC	Client: $\kappa(N^{3/4}\lambda+t)/\log N$ Server: $N^{3/4}\lambda\log N$	$\kappa\sqrt{N}(\lambda+t/\log N)$	$\kappa N^{3/4}t$	OWF
[BGIK22]+BC	Client: $\kappa N^{3/2}(\log N)^{2\log\kappa}t\lambda$ Server: $\kappa N^{3/2}(\log N)^{2\log\kappa}t\lambda$	$\sqrt{N}t\lambda$	$\sqrt{N}t$	OWF
[GZS24]* Thm. 5.1	Client: $\kappa(N\lambda+N^{3/4}t)$ Server: $\kappa(N\lambda+N^{3/4}t)$	$\kappa N^{3/4}(\lambda+t)$	$\kappa\sqrt{N}t$	OWF
Ours	Client: $\kappa\sqrt{N}(\lambda+t)$ Server: $\kappa(N\lambda+\sqrt{N}t)$	$\kappa\sqrt{N}(\log N\lambda+t)$	$\kappa\sqrt{N}t$	OWF
Batch Size $N^{1/3}$				
PIR + BC*	Client: $(\lambda+t)N^{1/3}$ Server: Nt	$(\lambda+t)N^{1/3}$	0	PIR
[LMW23]*	Client: $N^{1/3}t$ Server: $N^{1/3}t$	$N^{1/3}t$	0	RLWE
[CK20] Thm. 14 + BC	Client: $\kappa(N^{2/3}\lambda+t)/\log N$ Server: $N^{2/3}\lambda\log N$	$\kappa N^{1/3}(\lambda+t/\log N)$	$\kappa N^{2/3}t$	OWF
[BGIK22]+BC	Client: $\kappa N^{5/3}(\log N)^{2\log\kappa}t\lambda$ Server: $\kappa N^{5/3}(\log N)^{2\log\kappa}t\lambda$	$N^{1/3}t\lambda$	$\sqrt{N}t$	OWF
[GZS24]* Thm. 5.1	Client: $\kappa(N^{5/6}\lambda+N^{7/12}t)$ Server: $\kappa(N^{5/6}\lambda+N^{7/12}t)$	$\kappa N^{7/12}(\lambda+t)$	$\kappa\sqrt{N}t$	OWF
Ours	Client: $\kappa(N^{2/3}\lambda+N^{1/3}t)$ Server: $\kappa(N\lambda+N^{1/3}t)$	$\kappa(N^{2/3}\lambda\log N+N^{1/3}t)$	$\kappa N^{1/3}t$	OWF

Fig. 1. We compare our parameters of to other state of the art PIR schemes. N is the database size, λ is the computational security parameter, κ is the statistical security parameter and t is the size of a database entry. All values are given in terms of O. We combine all one-time preprocessing PIR schemes with optimal batch codes (BC) that split the database into $\sqrt{N}$ or $N^{1/3}$ blocks to adapt them to batch PIR. Choosing blocksizes significantly deviating from this will blow worsen the weak points, storage size for [CK20] and running times for [BGIK22]. The protocols with a * only require one server. The choice of cell color is purely subjective and is just meant as support to interpret the table.

and

$$\Pr[X \leq (1-\delta)\mu] \leq \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu}$$

Definition 1 ((Puncturable) Pseudorandom Function ((P)PRF)). *A* pseudorandom function (PRF) *with domain $\mathcal{D}$ and range $\mathcal{R}$ is a tuple of algorithms $f = (\mathsf{Gen}, \mathsf{Eval})$ with the following properties.*

- Gen *is randomized and takes in the security parameter λ. It outputs a a key k.*
- Eval *is deterministic and takes in a key k and $x \in \mathcal{D}$. It outputs $y \in \mathcal{R}$.*

We use the abbreviated notation $f_k(x) := \mathsf{Eval}(k, x)$. Moreover, for a subset $\mathcal{S} \subset \mathcal{D}$, we write $f_k(\mathcal{S}) := \{f_k(x) \mid x \in \mathcal{S}\}$. For puncturable pseudorandom functions (PPRF)*, we require the existence of an additional algorithm* $\mathsf{Puncture}$ *with the following properties.*

- $\mathsf{Puncture}$ *is deterministic and takes in a key k and a set $\mathcal{S} \subset \mathcal{D}$. It outputs a punctured key k^*.*

Definition 2 (Security and Correctness for (P)PRFs). *Let $f = (\mathsf{Gen}, \mathsf{Eval})$ be a PRF with domain $\mathcal{D}$ and range $\mathcal{R}$. We define the following security and correctness properties:*

- ***Pseudorandomness of Outputs.*** *f is said to be* pseudorandom *if for all PPT algorithms A, we have that*

$$\left| \Pr_{k \leftarrow \mathsf{Gen}(\lambda)}[b \leftarrow \mathsf{A}^{f_k(\cdot)}] - \Pr_{\substack{k \leftarrow \mathsf{Gen}(\lambda) \\ \sigma \leftarrow_{\$} \Sigma}}[b \leftarrow \mathsf{A}^{\sigma} \mid k \leftarrow \mathsf{Gen}(\lambda)] \right|$$

  is negligible and where Σ denotes the set of functions with domain $\mathcal{D}$ and range $\mathcal{R}$.

For PPRFs, we additionally require that:

- ***Pseudorandomness of Punctured Values.*** *f is said to have pseudorandomness for punctured keys if, for all sets $\mathcal{S} \subset \mathcal{D}$, and PPT algorithms A,*

$$\left| \Pr_{k \leftarrow \mathsf{Gen}(\lambda)}[b \leftarrow \mathsf{A}(\mathsf{Puncture}(k, \mathcal{S}), f_k(\mathcal{S}))] - \Pr_{\substack{k \leftarrow \mathsf{Gen}(\lambda) \\ r \leftarrow_{\$} \mathcal{R}^{|\mathcal{S}|}}}[b \leftarrow \mathsf{A}(\mathsf{Puncture}(k, \mathcal{S}), r)] \right|$$

  is negligible.
- ***Correctness of Puncturing.*** *f is said to be* correct *if for all $k \in \mathsf{Gen}(\lambda)$ and all $\mathcal{S} \subset \mathcal{D}$, the punctured key $k^* = \mathsf{Puncture}(k, \mathcal{S})$ satisfies $f_{k^*}(x) = f_k(x)$, for all $x \in \mathcal{D} \setminus \mathcal{S}$.*

Definition 3 (Pseudorandom Permutation (PRP)). *A* pseudorandom function (PRP) *on $\mathcal{D}$ is a triple of algorithms $p = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Inv})$ with the following properties.*

- Gen *is randomized and takes in the security parameter λ. It outputs a a key k.*
- Eval *is deterministic and takes in a key k and $x \in \mathcal{D}$. It outputs $y \in \mathcal{D}$. We require that $\mathsf{Eval}(k, \cdot)$ be bijective.*
- Inv *is deterministic and takes in a key k and $y \in \mathcal{D}$. It outputs the inverse x of y, i.e., such that $\mathsf{Eval}(k, x) = y$.*

We use the abbreviated notations $p_k(x) := \mathsf{Eval}(k, x)$ and $p_k^{-1}(y) := \mathsf{Inv}(k, y)$. Moreover, for a subset $\mathcal{S} \subset \mathcal{D}$, we write $p_k(\mathcal{S}) := \{p_k(x) \mid x \in \mathcal{S}\}$ and $p_k^{-1}(\mathcal{S}) := \{p_k^{-1}(y) \mid y \in \mathcal{S}\}$.

Definition 4 (Security for PRPs with Poly-Sized Domains). *Let $p = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Inv})$ be a PRP on $\mathcal{D}$. We say that p is* pseudorandom for polynomial domain sizes *if $\mathcal{D}$ is of polynomial size and for all ppt algorithms A, we have that*

$$\left| \Pr_{k \leftarrow \mathsf{Gen}(\lambda)} [b \leftarrow \mathsf{A}(p(\mathcal{D}))] - \Pr_{\substack{k \leftarrow \mathsf{Gen}(\lambda) \\ \pi \leftarrow_{\$} \Pi}} [b \leftarrow \mathsf{A}(\pi(\mathcal{D}))] \right|,$$

is negligible and where Π denotes the set of permutations on $\mathcal{D}$.

Definition 5 (Pseudorandom Puncturable Set (PPRS)). *A* pseudorandom puncturable set (PPRS) *with range $\mathcal{R}$ and size ℓ, and uncertainty set size ℓ' is tuple of algorithms $p = (\mathsf{Gen}, \mathsf{Expand}, \mathsf{IntersectSet}, \mathsf{Puncture}, \mathsf{InUncertaintySet})$ with the following properties.*

- Gen *is randomized and takes in the security parameter λ. It outputs a a key k representing a set $\mathcal{S}_k \subset \mathcal{R}$ of size ℓ.*
- Expand *is deterministic and takes in a key k. It outputs a set and outputs the set $\mathcal{S}_k$ the key represents.*
- $\mathsf{IntersectSet}$ *is deterministic and takes as input a key k representing a set $\mathcal{S}_k$ and a set $\mathcal{Q}$ and outputs their intersection.*
- $\mathsf{Puncture}$ *is deterministic and takes as input a key k representing a set $\mathcal{S}_k$ and an element $q \in \mathcal{R}$. It returns $\perp$ if $q \notin \mathcal{S}_k$. Otherwise, it returns a punctured key k^* representing the punctured set $\mathcal{S}_k \setminus \{q\}$.*
- $\mathsf{InUncertaintySet}$ *is deterministic and takes as input a punctured key k^* and an element q. It outputs 0 (accept) or 1 (reject). We refer to the set of values q for which $\mathsf{InUncertaintySet}(k, q) = 1$ as the* uncertainty set *$\tilde{\mathcal{S}}_{k^*}$ of k^* and require that $\tilde{\mathcal{S}}_{k^*}$ be of size ℓ' for all syntactically well-formed punctured keys k^*.* [4]

[4] Intuitively, $\tilde{\mathcal{S}}_{k^*}$ represents all possible values at which the puncturing leading to the key k^* could have occurred.

Definition 6 (Security and Correctness for PPRS). *Let* $(\mathsf{Gen}, \mathsf{Expand}, \mathsf{IntersectSet}, \mathsf{Puncture}, \mathsf{InUncertaintySet})$ *be a PPRS with range* $\mathcal{R}$*, size* ℓ*, and uncertainty set size* ℓ'*. We define the following security and correctness properties.*

- ***Pseudorandomness of Outputs.*** *p is said to be* pseudorandom *if for all PPT algorithms* A*, we have that*

$$\left| \Pr_{k \leftarrow \mathsf{Gen}(\lambda)} [b \leftarrow \mathsf{A}(\mathsf{Expand}(k))] - \Pr_{\mathcal{S} \leftarrow_{\$} \mathbb{P}_{=\ell}(\mathcal{R})} [b \leftarrow \mathsf{A}(\mathcal{S})] \right|$$

  is negligible.
- ***Pseudorandomness of Uncertainty Sets.*** *p is said have* pseudorandom uncertainty sets *if for all PPT algorithms* A*, we have that*

$$\left| \Pr_{\substack{k \leftarrow \mathsf{Gen}(\lambda) \\ q \leftarrow_{\$} \mathcal{S}_k \\ k^* \leftarrow \mathsf{Puncture}(k,q)}} [b \leftarrow \mathsf{A}(\tilde{\mathcal{S}}_{k^*})] - \Pr_{\mathcal{S} \leftarrow_{\$} \mathbb{P}_{=\ell'}(\mathcal{R})} [b \leftarrow \mathsf{A}(\mathcal{S})] \right|$$

  is negligible.
- ***Pseudorandomness of Punctured Elements.*** *For a PPT algorithm* A*, an element* $q \in \mathcal{R}$*, and* $b \in \{0,1\}$*, we define the experiment* $\mathbf{PPRS} - \mathbf{KEY}^{\mathsf{A}}_{q,b}$ *below as follows:*
  - *If* $b = 0$*, sample a key* k *as* $k \leftarrow \mathsf{Gen}(\lambda)$ *conditioned on* $q \in \mathcal{S}_k$ *and compute the punctured key* k^* *as* $k^* = \mathsf{Puncture}(k, q)$*.*
  - *If* $b = 1$*, sample a key* k *as* $k \leftarrow \mathsf{Gen}(\lambda)$ *and sample the element* q' *as* $q' \leftarrow \mathcal{S}_k$*. Compute the punctured key* k^* *as* $k^* = \mathsf{Puncture}(k, q')$*, conditioned on* $q \in \tilde{\mathcal{S}}_k$*.*
  - *Run* A *on input* k^* *and return* A*'s output* b'*.*

  p is said to have pseudorandomness for punctured keys *if for all* A *and all* $q \in \mathcal{R}$*,*

$$\left| \Pr[\mathbf{PPRS} - \mathbf{KEY}^{\mathsf{A}}_{q,0} = 1] - \Pr[\mathbf{PPRS} - \mathbf{KEY}^{\mathsf{A}}_{q,1} = 1] \right|$$

  is negligible.

We define private information retrieval for the specific communication pattern our protocol has. Generally, private information retrieval does not have to follow this specific pattern.

Definition 7 (Two-Server Batch Private Information Retrieval (PIR)). *A two server batch PIR scheme* $\mathsf{2sbPIR}$ *consists of 6 PPT algorithms* $(\mathsf{client}_{1,1}, \mathsf{server}_1, \mathsf{client}_{1,2}, \mathsf{client}_{2,1}, \mathsf{server}_2, \mathsf{client}_{2,2})$ *with the following syntax:*

- $\mathsf{client}_{1,1}(\lambda, N, B)$*: Takes as input a security parameter* λ*, a database size* N *and a batch parameter* B *and outputs a message* c_1 *and a state* st_1*.*

- $\mathsf{server}_1(\mathsf{DB}, c_1, K)$: *Takes as input a database* $\mathsf{DB} \in \{0,1\}^N$*, a client message* c_1 *as well as optionally a key* $K \in \{0,1\}^\lambda$ *and outputs a message* s_1
- $\mathsf{client}_{1,2}(s_1)$: *Takes as input a server message* s_1 *and a state* st_1 *and outputs a preprocessing* H*.*
- $\mathsf{client}_{2,1}(H, \mathcal{Q} \in [N]^B)$: *Takes as input a preprocessing* H *as well as a list of queries* Q *and outputs a message* c_2 *as well as a state* st_2*.*
- $\mathsf{server}_2(\mathsf{DB}, c_2, K)$: *Takes as input a database* DB*, a client message* c_2 *as well as optionally a key* K*. Outputs a message* s_2
- $\mathsf{client}_{2,2}(\mathsf{st}, s_2)$: *Takes as input a state* st *and a server message* s_2 *and outputs a list* out*.*

The optional joint input K for the two servers is only needed if server privacy is desired. In this case $K \leftarrow \{0,1\}^\lambda$ is a uniformly random key.

We require the following properties:

- ***Statistical Correctness.*** *For all databases* $\mathsf{DB} \in \{0,1\}^N$ *and query sets* $\mathcal{Q} \subset [N]$*, where* $|\mathcal{Q}| = B$ *we get that* $\mathsf{client}(\lambda)$ *interacting with* $\mathsf{server}_1(\mathsf{DB})$ *then* $\mathsf{client}(H, \mathcal{Q})$ *interacting with* $\mathsf{server}_2(\mathsf{DB})$ *outputs query-database-entry pairs* $(q_1, \mathsf{DB}[q_1]), \ldots, (q_B, \mathsf{DB}[q_B])$*, where* $q_1, \ldots, q_B$ *are the* B *distinct elements of* $\mathcal{Q}$ *with all but negligible probability.*
- ***Client Privacy.*** *We require that for all PPT distinguishers* D *and all databases* DB *there exists a PPT simulator* Sim_C *such that for all query-sets* $\mathcal{Q}$ *the following holds:*

$$\left|\Pr[\mathsf{D}(c_2, K) = 1] - \Pr[\mathsf{D}(\mathsf{Sim}_C(1^\lambda, N, B, \mathsf{DB}, K), K)]\right| \leq \mathsf{negl}(\lambda),$$

  where $K \leftarrow \{0,1\}^\lambda$ *is chosen uniformly random and* c_2 *is sampled by*
  - $c_1 \leftarrow \mathsf{client}_{1,1}(\lambda, N, B)$
  - $s_1 \leftarrow \mathsf{server}_1(\mathsf{DB}, c_1, K)$
  - $H \leftarrow \mathsf{client}_{1,2}(s_1)$
  - $(c_2, \mathsf{st}) \leftarrow \mathsf{client}_{2,1}(H, \mathcal{Q})$
- ***Server Privacy.*** *We require that for all PPT-distinguishers* D *and all query-sets* $\mathcal{Q}$ *there exists a PPT simulator* Sim_S *such that for all databases* DB *the following holds:*

$$\left|\Pr[\mathsf{D}(\mathsf{view}_{\mathsf{client}}(\mathcal{Q}, \mathsf{DB})) = 1] - \Pr[\mathsf{D}(\mathsf{Sim}_C(1^\lambda, N, B, \mathcal{Q}))]\right| \leq \mathsf{negl}(\lambda),$$

  where $\mathsf{view}_{\mathsf{client}}(\mathcal{Q}, \mathsf{DB})$ *encompasses the entire view of a semi-honest client (including all random coins) in a run of the protocol* 2sbPIR *with client input* $\mathcal{Q}$ *and server input* DB*.*

3 Puncturable Pseudorandom Sets

In our protocol, we use subsets of $[N]$, where we will assume, for simplicity, that N is a power s.t. we can easily represent elements in binary. Our goal is to puncture these sets at one of their elements. To implement this functionality, we construct a type of pseudorandom set with two different types of representations.

We call these representations *pseudorandom* and *punctured*. At the core of our set representation is the following keyed permutation

$$g_{k',k'',\Delta}(L,R) = p_{k'}(L \oplus f_{k''}(R) \oplus \Delta, R)$$

where $p_{k'}$ is a pseudorandom permutation and f is a puncturable pseudorandom function. It is fairly easy to see that g is a permutation by seeing that $p_{k'}$ is a permutation and $(L,R) \mapsto (L \oplus f(R) \oplus \Delta, R)$ is a permutation.

The basic idea is that the set is described by the key k' for the permutation, the key k'' for the function and a set $\mathcal{V}$. Then the set

$$S_{k',k'',\Delta,\mathcal{V}} = \{g_{k',k''}(0,x) | x \in [N^{2/3}] \setminus \mathcal{V}\}$$

In our pseudorandom representation of S we have that V is also pseudorandom, i.e. $\mathcal{V} = G(s)$ for some PRG G that takes as input a seed $s \in \{0,1\}^\lambda$ and outputs a subset of $[N^{2/3}]$ of size $N^{1/3} - 1$. So to be exact the pseudrandom representation is

$$S_{k',k'',\Delta,s} = \{g_{k',k''}(0,x) | x \in [N^{2/3}] \setminus G(s)\}$$

When we puncture a set $S_{k',k'',\Delta,s}$ at an element q that means we want to remove that element for which we know it is contained. We do this by computing $p^{-1}_{k'}(q) = (L,R)$. Because we know that q is in the set we derive that $L = f_{k''}(R) \oplus \Delta$. We then puncture k'' at points $\mathcal{V} \cup \{R\}$ and get a new punctured key $\bar{k}$. The description now becomes $k', \bar{k}, \Delta, \mathcal{V} \cup \{R\}$ which represents the set

$$S_{k',\bar{k},\Delta,\mathcal{V}\cup\{R\}} = \{g_{k',\bar{k}}(0,x) | x \in [N^{2/3}] \setminus (\mathcal{V} \cup \{R\})\}$$

This clearly doesn't fully hide q from an adversary who gets to see $(k', \bar{k}, \Delta, \mathcal{V} \cup \{R\})$. Indeed, the adversary learns that the punctured point q is in the following set

$$\tilde{S}_{k',\bar{k},\Delta,\mathcal{V}\cup\{R\}} = \{p_{k'}(L',R') | L' \in [N^{1/3}], R' \in \mathcal{V} \cup \{R\}\}$$

We prove that this is all the adversary learns about the punctured as this will be important in the analysis of our protocol.

Construction 1 *We describe the algorithms required for the set more formally and generally, as there are different trade-off when choosing a different split for L and R and choosing the size of V. I.e. we split $[N]$ into $[N/\alpha] \times [\alpha]$ and parameterize the size of V as β, where $\beta < \alpha$. Let G be a PRF with domain $\{0,1\}^\lambda$ and range $\mathcal{R} \subset [\alpha]$ of size β and let f denote a PPRF with domain $[\alpha]$, range $[N/\alpha]$, and puncturing algorithm $\mathsf{Puncture}_f$. Moreover, let p, g denote PRPs on $[N]$. In the following, we find it convenient to view g and p as functions $[N/\alpha] \times [\alpha] \longrightarrow [N]$.*

- ***Algorithm*** Gen*: On input the security parameter λ, sample $k', k'', s \leftarrow_\$ \{0,1\}^\lambda$ and $\Delta \leftarrow_\$ [N/\alpha]$. Return $k = (k', k'', \Delta, s)$.*
- ***Algorithm*** Expand*: On input a key $k = (k', k'', \Delta, s)$, set $\mathcal{V} \leftarrow G(s)$ and $\tilde{V} = \{0\} \times ([\alpha] \setminus \mathcal{V})$. Then compute $\mathcal{S}_k \leftarrow g_{k',k'',\Delta}(\tilde{\mathcal{V}})$ and return $\mathcal{S}_k$. Clearly,* Expand *runs in time runs $\tilde{O}(\alpha)$ and returns $\mathcal{S}_k$ of size $\alpha - \beta$.*

– ***Algorithm*** $\mathsf{IntersectSet}$*: On input a key* $k = (k', k'', \Delta, s)$ *and a set* $\mathcal{Q}$*, compute* $\mathcal{V} \leftarrow G(s)$*, and initialize* $\mathcal{I} = \emptyset$*. For each* $q \in \mathcal{Q}$*, compute* $(L', R') \leftarrow p_{k'}^{-1}(q)$ *and add* q *to* $\mathcal{I}$ *if* $R' \notin \mathcal{V}$ *and* $L' = f_{k''}(R') \oplus \Delta$*. Return* $\mathcal{I}$*.* $\mathsf{IntersectSet}$ *runs in time* $\tilde{O}(\beta + |\mathcal{Q}|)$*, which is faster than computing* $\mathsf{Expand}(k) \cap \mathcal{Q}$ *when* $|\mathcal{Q}| + \beta \in o(\alpha)$*.*
– ***Algorithm*** $\mathsf{Puncture}$*: On input a key* $k = (k', k'', \Delta, s)$ *and an element* q*, return* $\perp$ *if* $q \notin \mathcal{S}_k$*. Otherwise, compute* $\mathcal{V} \leftarrow G(s)$ *and* $(L, R) = p_{k'}^{-1}(q)$*. Set* $\bar{k} \leftarrow \mathsf{Puncture}_f(k'', \mathcal{V} \cup \{R\})$ *and return* $(k^* = (k', \bar{k}, \Delta, \mathcal{V} \cup \{R\}))$*. The above algorithm runs in time* $\tilde{O}(\beta)$*.*
– ***Algorithm*** $\mathsf{InUncertaintySet}$*: On input a punctured key* $k^* = (k', \bar{k}, V \cup \{R\}))$ *and an element* q*, let* $(L', R') \leftarrow p_{k'}(q)$*. Return 1 if* $R' \in V \cup \{R\}$ *and 0 otherwise. This algorithm runs in time* $\tilde{O}(1)$*.*

For the sake of completeness we detail Expand *and* $\mathsf{IntersectSet}$ *also for punctured keys, but the only difference will be that* $\mathcal{V} \cup \{R\}$ *is provided as an input rather than obtaining it from* s *via* G*. (Clearly, this only decreases the running times).*

– ***Algorithm*** Expand*: On input a punctured key* $(k^* = (k', \bar{k}, \Delta, \mathcal{V}' = \mathcal{V} \cup \{R\}))$*, set* $\tilde{V} = \{0\} \times ([\alpha] \setminus \mathcal{V}')$*. Then compute* $\mathcal{S}_k \leftarrow g_{k', k'', \Delta}(\tilde{\mathcal{V}})$ *and return* $\mathcal{S}_{k^*}$*.*
– ***Algorithm*** $\mathsf{IntersectSet}$*: On input a punctured key* $k^* = (k', \bar{k}, \Delta, \mathcal{V}' = V \cup \{R\}))$ *and a set* $\mathcal{Q}$*, initialize* $\mathcal{I} = \emptyset$*. For each* $q \in \mathcal{Q}$*, compute* $(L', R') \leftarrow p_{k'}^{-1}(q)$ *and add* q *to* $\mathcal{I}$ *if* $R' \notin \tilde{V}'$ *and* $L' = f_{k''}(R') \oplus \Delta$*. Return* $\mathcal{I}$*.*

3.1 Security

Theorem 1. *The pseudorandom set in construction 1 is pseudorandom at punctured points with respect to definition 6.*

Proof. Assume there is a PPT-distinguisher $\mathcal{D}$ which distinguishes between $b = 0$ and $b = 1$ with non-negligible advantage ϵ. Consider the following sequence of hybrids.

– Hybrid H_0: This is the experiment with $b = 0$.
  • Do
    ∗ Sample $k = (k', k'', \Delta, s) \leftarrow \mathsf{Gen}(1^\lambda)$.
    ∗ Sample a random β-sized set $\mathcal{V} \subset [\alpha]$ using random coins $G(s)$.
  • Until $q \in \{p_{k'}(f_{k''}(x) \oplus \Delta, x) | x \in [\alpha] \setminus \mathcal{V}\}$.
  • Let $(L, R) \leftarrow p_{k'}^{-1}(q)$
  • Let $\mathcal{W} \leftarrow \mathcal{V} \cup \{R\}$.
  • Puncture k'' at $\mathcal{W}$ and call the result $\bar{k}$.
  • Let $k^* \leftarrow (k', \bar{k}, \Delta, \mathcal{W})$.
– Hybrid H_1: This is a bridging step and distributed identically to H_0.
  • In H_1, we sample $\mathcal{V}$ such that $q \in \{p_{k'}(f_{k''}(x) \oplus \Delta, x) | x \in [\alpha] \setminus \mathcal{V}\}$ and set then sample $(L, R) \leftarrow p_{k'}^{-1}(q)$.
  • In H_2 we first sample $\mathcal{V}$, sample $(L, R) \leftarrow p_{k'}^{-1}(q)$ and condition on $R \notin V$ and $L = f_{k''}(R) \oplus \Delta$.

Hence, H_2 is given as follows:

- Do
  - Sample $k = (k', k'', \Delta, s) \leftarrow \mathsf{Gen}(1^\lambda)$.
  - Let $(L, R) \leftarrow p_{k'}^{-1}(q)$.
  - Sample a random β-sized set $\mathcal{V} \subset [\alpha]$ using random coins $G(s)$.
  - Let $\mathcal{W} \leftarrow \mathcal{V} \cup \{R\}$.
  - Puncture k'' at $\mathcal{W}$ and call the result $\bar{k}$.
  - Let $k^* \leftarrow (k', \bar{k}, \Delta, \mathcal{W})$.
- Until $R \notin \mathcal{V}$ and $L = f_{k''}(R) \oplus \Delta$.

– Hybrid H_2: This is identical to H_1, except that we choose $\mathcal{V}$ as a truly uniform set of size β instead of pseudorandomly. Computational indistinguishability follows routinely from the pseudorandomness of $G(s)$.

– Hybrid H_3: In this hybrid, we replace $f_{k''}(R)$ by a uniformly random value. Computational indistinguishability follows routinely from the puncturing securtity of f, as we puncture k'' on $\mathcal{W}$ and $R \in \mathcal{W}$. Consequently, we can drop the loop exit condition $L = f_{k''}(R) \oplus \Delta$ as $f_{k''}(R) \oplus \Delta$ is now distributed uniformly and independently of all other values. Furthermore, we can include the condition $R \notin \mathcal{V}$ directly into the sampling process for $\mathcal{V}$. Hence we can write H_3 as follows:

- Sample $k = (k', k'', \Delta, s) \leftarrow \mathsf{Gen}(1^\lambda)$.
- Let $(L, R) \leftarrow p_{k'}^{-1}(q)$.
- Sample a random β-sized set $\mathcal{V} \subset [\alpha] \setminus \{R\}$.
- Let $\mathcal{W} \leftarrow \mathcal{V} \cup \{R\}$.
- Puncture k'' at $\mathcal{W}$ and call the result $\bar{k}$.
- Let $k^* \leftarrow (k', \bar{k}, \Delta, \mathcal{W})$.

We remark that now $\mathcal{W}$ is now a uniformly random subset of size $\beta+1$ of $[\alpha]$ subject to there being an $L \in [N/\alpha]$ and an $R \in \mathcal{W}$ such that $p_{k'}(L, R) = q$.

– Hybrid H_4: In this hybrid, we sample $\mathcal{W}$ differently, namely

- Sample a random β-sized set $\mathcal{V} \subset [\alpha]$.
- Sample R' uniformly at random from $[\alpha] \setminus \mathcal{V}$.
- Let $\mathcal{W} \leftarrow \mathcal{V} \cup \{R'\}$.
- Reject and resample if $R \notin \mathcal{W}$, where $p_{k'}(L, R) = q$.

Note that a set $\mathcal{W}$ sampled this way is uniformly random under the condition that $R \in \mathcal{W}$. By the above remark, H_3 and H_4 are identically distributed. Hence, we can write H_4 as

- Do
  - Sample $k = (k', k'', \Delta, s) \leftarrow \mathsf{Gen}(1^\lambda)$.
  - Sample a random β-sized set $\mathcal{V} \subset [\alpha]$.
  - Sample R' uniformly at random from $[\alpha] \setminus \mathcal{V}$.
  - Let $\mathcal{W} \leftarrow \mathcal{V} \cup \{R'\}$.
  - Let $(L, R) \leftarrow p_{k'}^{-1}(q)$.
  - Puncture k'' at $\mathcal{W}$ and call the result $\bar{k}$.
  - Let $k^* \leftarrow (k', \bar{k}, \Delta, \mathcal{W})$.
- Until $R \in \mathcal{W}$.

– Hybrid H_6: This hybrid is identically distributed to hybrid 4, except that we sample set set $\mathcal{V}$ pseudorandomly using coins $G(s)$, computational indistinguishability again follows routinely from the pseudorandomness of $G(s)$.
– Hybrid H_5: In this hybrid we replace the condition $R \in W$ with $q \in \{p_{k'}(L,R)|L \in [N/\alpha], R \in \mathcal{W}\}$. Note that the two conditions are equivalent as $(L,R) \leftarrow p_{k'}^{-1}(q)$. Furthermore, instead of sampling R' uniformly random from $[\alpha] \setminus \mathcal{V}$, we sample q' uniformly random from $\{p_{k'}(f_{k''}(x) \oplus \Delta, x)|x \in [\alpha] \setminus \mathcal{V}\}$ and set $(L', R') \leftarrow p_{k'}^{-1}(q')$. Note that that these two procedures produce the same distribution for R'. We can write H_5 as
  • Do
    * Sample $k = (k', k'', \Delta, s) \leftarrow \mathsf{Gen}(1^\lambda)$.
    * Sample a random β-sized set $\mathcal{V} \subset [\alpha]$ using coins $G(s)$.
    * Sample $q' \leftarrow_\$ \{p_{k'}(f_{k''}(x) \oplus \Delta, x)|x \in [\alpha] \setminus \mathcal{V}\}$
    * Set $(L', R') \leftarrow p_{k'}^{-1}(q')$
    * Let $\mathcal{W} \leftarrow \mathcal{V} \cup \{R'\}$.
    * Puncture k'' at $\mathcal{W}$ and call the result $\bar{k}$.
    * Let $k^* \leftarrow (k', \bar{k}, \Delta, \mathcal{W})$.
  • Until $q \in \{p_{k'}(L,R)|L \in [N/\alpha], R \in \mathcal{W}\}$.

  This is identically distributed to the experiment with bit $b = 1$. Hence this concludes the proof.

Theorem 2. *The pseudorandom set in construction 1 is pseudorandom according to Definition 6.*

Proof. The key k has the structure (k', k'', Δ, s) and the set is described by $S_{k',k'',\Delta,s} = \{g_{k',k''}(0,x)|x \in [\alpha] \setminus G(s)\}$, where $g_{k',k'',\Delta}(L,R) = p_{k'}(L \oplus f_{k''}(R) \oplus \Delta, R)$, $p_{k'}$ is a pseudorandom permutation and f is a puncturable pseudorandom function.

In a standard hybrid argument we can argue that $p_{k'}$ is indistinguishable from a random permutation using its pseudorandomness. Because permutations are a group and g is just a composition of p and $(L,R) \mapsto (L \oplus f_{k''}(R) \oplus \Delta, R)$ we can also argue that g is computationally indistinguishable from a random permutation.

Evaluating a random permutation $\alpha - \beta$ many fixed points is the same as choosing $\alpha - \beta$ many distinct elements at random.

Theorem 3. *The uncertainty set in construction 1 is pseudorandom according to Definition 6.*

Proof. Note first that puncturing a set $\tilde{S}_k$ at a random point in $q \leftarrow \tilde{S}_k$ is equivalent to sampling a uniformly random $R \leftarrow [\alpha]$ and puncturing the PRF key k'' at R.

By the PRP security of p we can replace $p_{k'}$ by a uniformly random permutation π. Consequently,

$$\tilde{S}_{k^*} = \{\pi(L', R') \mid L' \in [n/\alpha], R' \in V \cup \{R\}\}$$

is a uniformly random set of size $(\beta + 1) \cdot N/\alpha$.

4 Batch Unbiased Sampling

In the upcoming PIR protocol, we need to carefully sample pseudorandom sets in two different ways. First, we have a list of sets with hints and we need pick out the ones that are useful to us while not introducing any unpredictable bias. Then, we modify these useful set and integrate them back into a population of dummy sets, that are independent of the queries, without disturbing the distribution. The sampling procedures are entirely independent of the application and might be of independent interest.

Split Sampling. In the following, we detail a process that allows us to, from a list of unbiased samples, collect samples into lists such that the lists follow the underlying distribution under the condition that they fulfill certain predicates.

Let $\varphi_1, \ldots, \varphi_n$ be predicates on some finite domain X. Define the following algorithm $\mathsf{SplitSamp}$, which takes as input κn samples $(a_i)_{i\in[\kappa n]}$ from X with associated auxiliary information $(\mathsf{aux}_i)_{i\in[\kappa n]}$ and outputs n lists $(\mathsf{A}'_i)_{i\in[n]}$.

$\mathsf{SplitSamp}((a_i, \mathsf{aux}_i)_{i\in[\kappa n]})$:

- Let $\mathsf{A}^{(0)} \leftarrow ((a_l, \mathsf{aux}_l))_{l\in[\kappa n]}$ be a copy of the input list of pairs with auxiliary information.
- Let $t \leftarrow 0$.
- For $i \in [n]$:
  - Let A'_i be a output list of tuples. We will fill this list up in the process.
  - For $l \in [\kappa n]$:
    - If $\varphi_i(a_l^{(t)})$ holds, where $a_l^{(t)}$ is the l-th entry of $\mathsf{A}^{(t)}$:
      - Append $(a_l^{(t)}, \mathsf{aux}_l^{(t)})$ to A'_i.
      - Sample $a^* \leftarrow \chi$ conditioned on $\varphi_i(a^*)$ holding
      - Let $\mathsf{A}^{(t+1)}$ be a copy of $\mathsf{A}^{(t)}$, where we replace $(a_l^{(t)}, \mathsf{aux}_l^{(t)})$ by $(a^*, \perp)$.
      - Let $t \leftarrow t + 1$.
- Output $(\mathsf{A}'_i)_{i\in[n]}$.

On a high level the algorithm goes through the list of samples and puts them into the lists A'_i where each of these lists corresponds to a predicate φ_i. To keep the distribution the same between $\mathsf{A}^{(t)}$ and $\mathsf{A}^{(t+1)}$ we replace the samples that we take out of $\mathsf{A}^{(t)}$ by ones that follow the same distribution.

We prove two theorems about this algorithm. The first one captures behaviour that is independent of specifics about the conditions $(\varphi_i)_{i\in[n]}$.

Theorem 4. *Let $n \in \mathbb{N}$. Fix a distribution χ supported on X. Assume now that $(a_i)_{i\in[\kappa n]}$ are κn independent samples of χ. Now let $(\mathsf{A}'_i)_{i\in[n]}$ be the output of $\mathsf{SplitSamp}((a_i, \mathsf{aux}_i)_{i\in[\kappa n]})$ and parse $\mathsf{A}'_i = (a'_{i,j}, \mathsf{aux}'_{i,j})_{i\in[n], j\in[l_i]}$, where l_i is the length of A'_i. For all $i \in [n]$ the following holds: $(a'_{i,j})_{j\in[l_i]}$ is a list of independent samples of χ under the condition that φ_i holds for each sample.*

Lemma 1. *For all t with $\mathsf{A}^{(t)} = ((a_i^{(t)}, \mathsf{aux}_i^{(t)}))_{i\in[\kappa n]}$ we have that $(a_i^{(t)})_{i\in[\kappa n]}$ follows the distribution of κn independent samples from χ.*

Proof (Proof of Lemma 1). $(a_i^{(0)})_{i\in[\kappa n]}$ follow the distribution χ by definition. For any $t \geq 0$ the only difference is that we sample $a_i^{(t)}$ as $a_i^{(t+1)}$ where $\varphi_j(a_i^{(t+1)}) = 1 = \varphi_j(a_i^{(t)})$ for some $i \in [\kappa n]$, $j \in [n]$. Therefore $(a_i^{(t)})_{i\in[\kappa n]}$ and $(a_i^{(t+1)})_{i\in[\kappa n]}$ are identically distributed.

Proof (Proof of Theorem 4). Fix an $i \in [n]$. We only have to argue about the i-th run of the outer loop of $\mathsf{SplitSamp}$ as all operations with regard to A'_i happen in that run of the loop. Let $t_{i,l}$ be the value that t has at the beginning of the i-th run of the outer loop and the l run of the inner loop. As the inner loop in the l-th run only ever does operations on the l-th element of $\mathsf{A}^{(t_{i,l})}$ we have for all $l \in [\kappa n]$ that $a_l^{(t_{i,l})} = a_l^{(t_{i,0})}$. By Lemma 1 we get that $a_l^{(t_{i,l})}$ follows the distribution χ for all $l \in [\kappa n]$. $a_l^{(t_{i,l})}$ gets added to A'_i if $\varphi_i(a_l^{(t_{i,l})}) = 1$. Therefore, if it is in A'_i it follows the distribution χ under the condition that φ_i holds.

This next theorem captures the behaviour of $\mathsf{SplitSamp}$ if there are certain relation between the predicates and the distribution χ.

Theorem 5. *Let $n \in \mathbb{N}$. Fix a distribution χ supported on X as well as predicates $(\varphi_i)_{i\in[n]}$ on X. Assume that there are constants $c, c' > 0$ such that following holds. Let $x \leftarrow \chi$, then for all $i \in [n]$*

1. *the probability (over the choice of $x \leftarrow \chi$) that $\varphi_i(x) = 1$ is less than c/n*
2. *the probability (over the choice of $x \leftarrow \chi$) that $\varphi_i(x) = 1$ and for all $j \neq i$ $\varphi_j(x) = 0$ is at least c'/n*

Assume now that $(a_i)_{i\in[\kappa n]}$ are κn independent samples of χ and let $(\mathsf{aux}_i)_{i\in[\kappa n]}$ be their associated auxiliary information. Now let $(\mathsf{A}'_i)_{i\in[n]}$ be the output of $\mathsf{SplitSamp}((a_i, \mathsf{aux}_i)_{i\in[\kappa n]})$. For all $i \in [n]$ the following holds:

- *The size of A'_i at most $\kappa 3c/2$, except with negligible in λ probability over all random choices.*
- *there are at least $c'\kappa/2$ distinct pairs $(a_1^*, \mathsf{aux}_1^*), \ldots, (a_{c'\kappa/2}^*, \mathsf{aux}_{c'\kappa/2}^*)$ in A'_i such that each of these pairs was in the input list $(a_i, \mathsf{aux}_i)_{i\in[\kappa n]}$, except with negligible probability over all random choices.*

Proof. The proof follow directly from the upcoming lemmas 2 and 3

Lemma 2. *For all $i \in [n]$ there are at least $c'\kappa/2$ distinct sample-auxiliary-information pairs $(a_1^*, \mathsf{aux}_1^*), \ldots, (a_\kappa^*, \mathsf{aux}_\kappa^*)$ in A'_i such that each of these pairs was in the input list $(a_i, \mathsf{aux}_i)_{i\in[\kappa n]}$, except with negligible probability over all random choices.*

Proof. For all $i \in [n]$, by the structure of $\mathsf{SplitSamp}$, we get that if for an element (a, aux) of A $\varphi_i(a) = 1$ holds, but for all $j < i$ we have $\varphi_j(a) = 0$, then (a, aux) will end up in A'_i. Therefore, it is enough to analyse the number of elements (a, aux) in the initial list A such that $\varphi_i(a) = 1$ and but $\varphi_j(a) = 0$ for $j \neq i$.

By precondition 2 of Theorem 5 we know that a random sample of χ fulfills only φ_i with probability $> c'n^{-1}$. A contains κn independent samples of χ.

So, in expectation it contains $c'\kappa$ such samples. Let S be the random variable that describes the number of such samples. By multiplicative Chernoff bound we get that $\Pr[S \leq \kappa c'/2] \leq \left(\frac{e^{-1/2}}{(-1/2)^{-1/2}}\right)^{c'\kappa} < 0.9^{c'\kappa}$, which is negligible in κ. Therefore, the number of samples in A'_i which satisfy only φ_i is $> c'\kappa/2$ with all but negligible probability.

Lemma 3. *For all $i \in [n]$, the length l_i of list A'_i satisfies $l_i \leq \kappa 3c/2$ with all but negligible probability.*

Proof. By the structure of $\mathsf{SplitSamp}$ for each $i \in [\kappa n]$, $j \in [n]$ we take a pair (a, aux) and add (a, aux) to A'_j if $\varphi_j(a) = 1$. By Lemma 1 it holds that for all $i \in [\kappa n]$, $t \in \mathbb{N}$ $a_i^{(t)}$ is identically distributed to χ. Therefore, it is enough to analyse the number of elements in a freshly sampled list A to determine how many are put into A'_i.

A has κn elements and the first coordinate of an element has a probability of fulfilling φ_i with probability $< cn^{-1}$ by precondition 1 of Theorem 5. Therefore, the expected number of elements where the first coordinate fulfills φ_i is $c\kappa$. Let S be the random variable that describes the number of such samples. By multiplicative Chernoff bound we get that $\Pr[S \geq \kappa 3c/2] \leq \left(\frac{e^{1/2}}{(3/2)^{3/2}}\right)^{c\kappa} < 0.9^{c\kappa}$, which is negligible in κ. Therefore, the number of samples in A'_i is $\leq \kappa 3c/2$ with all but negligible probability.

Merge Sampling In the following, we detail a sampling procedure that takes as inputs lists of samples, where each list follows the distribution χ, except that they are conditioned on fulfilling different predicates. In each list some of the samples are "useful" ($\mathsf{aux} \neq \bot$). Our procedure outputs a list of samples, where each output follows the distribution χ and there are useful samples from each input list.

Let $\varphi_1, \ldots, \varphi_n$ be predicates on some finite domain X. Define the following algorithm $\mathsf{MergeSamp}$, for some constant $0 < c, c^\dagger$, which takes as input n lists of samples $(\mathsf{A}'_i)_{i\in[n]}$, with each entry being a pair of values from X with associated auxiliary information aux and outputs a list A of pairs from the same domains.

$\mathsf{MergeSamp}((\mathsf{A}'_i)_{i\in[n]})$:

- For $i \in [2c\kappa n/c^\dagger]$:
  - Sample $a_i^{(0)}$ from χ.
  - Let $\mathsf{aux}_i^{(0)}$ be $\bot$.
  - For $j \in [n]$:
    - If $\varphi_j(a_i^{(j-1)})$ holds and $l_j \neq 0$, where l_j is the length of A'_j:
      - Let $(a_i^{(j)}, \mathsf{aux}_i^{(t)}) \leftarrow (a'_{j,l_j}, \mathsf{aux}_{j,l_j})$.
      - Remove $(a'_{j,l_j}, \mathsf{aux}_{j,l_j})$ from A'_j.
    - Else:
      - Let $(a_i^{(j)}, \mathsf{aux}_i^{(j)}) \leftarrow (a_i^{(j-1)}, \mathsf{aux}_i^{(j-1)})$
- Output $\mathsf{A} = ((a_i^{(n)}, \mathsf{aux}_i^{(n)}))_{i\in[2c\kappa n/c^\dagger]}$.

Remark 1. Note that $\mathsf{MergeSamp}$ always outputs a list of *fixed length* $2c\kappa n/c^\dagger$. (This will be a crucial feature in the proof of Theorem 9)

About this algorithm too we prove two different theorems. The first one is independent of specifics of the predicates.

Theorem 6. *Let $n \in \mathbb{N}$. Fix a distribution χ on X as well as predicates $(\varphi_i)_{i\in[n]}$ on X. Parse entries of the lists in the following ways $\mathsf{A}'_i = ((a'_{i,j}, \mathsf{aux}_{i,j}))_{j\in[l_i]}$, where l_i is the length of A'_i. Assume for each $i \in [n], j \in [l_i]$ we have that $a'_{i,j}$ is a sample from χ under the condition that $\varphi_i(a'_{i,j}) = 1$ holds.*

We get that all elements $(a_i)_{i\in[2c\kappa n/c^\dagger]}$, where $((a_i, \mathsf{aux}_i))_{i\in[2c\kappa n/c^\dagger]}$ are the elements of $\mathsf{A} \leftarrow \mathsf{MergeSamp}((\mathsf{A}'_i)_{i\in[n]})$, are distributed like independent samples from χ.

Proof. Follows directly from the upcoming Lemma 4 and the fact that $\mathsf{A} = ((a_i^{(n)}, \mathsf{aux}_i^{(n)}))_{i\in[2c\kappa n/c^\dagger]}$.

Lemma 4. *For all $j_1, \ldots, j_{2c\kappa n/c^\dagger} \in [n] \cup \{0\}$ we have $(a_i^{(j_i)})_{i\in[2c\kappa n/c^\dagger]}$ are independent samples from χ.*

Proof. We prove by induction. For $j_1 = \cdots = j_{2c\kappa n/c^\dagger} = 0$ the claim follows because $(a_i^{(0)})_{i\in[n]}$ are all fresh independent samples of χ. Fix an $i \in [2c^\dagger \kappa n]$. Now we prove the induction step from j_i to $j_i + 1$. If $\varphi_{j_i+1}(a_i^{(j_i)}) = 1$ and A'_{j_i} is not empty then $a_i^{(j_i+1)}$ is a new independent sample of χ under the condition that φ_{j_i+1} holds. It is independent because A'_{j_i+1} are independent samples of χ under the condition that φ_{j_i+1} hold and we only use them because we remove them from A'_{j_i+1} upon use. Further, replacing $a_i^{(j_i)}$ by $a_i^{(j_i+1)}$ because they are both samples of χ under the condition that φ_{j_i+1} holds. The statement follows by induction.

Theorem 7. *Let $n \in \mathbb{N}$. Fix a distribution χ on X as well as predicates $(\varphi_i)_{i\in[n]}$ on X. Assume that there are constants $c, c', c^\dagger, c^* > 0$ such that the following holds.*

1. *Let $x \leftarrow \chi$, then for all $i \in [n]$ the probability (over the choice of $x \leftarrow \chi$) that $\varphi_i(x) = 1$ is $> c^\dagger n^{-1}$*
2. *Let $x \leftarrow \chi$ under the condition that $\varphi_i(x) = 1$, then for all $i \in [n]$ the probability (over the choice of $x \leftarrow \chi$) for all $j \neq i$ $\varphi_j(x) = 0$ is less than c^*.*

Parse entries of the lists in the following ways $\mathsf{A}'_i = ((a'_{i,j}, \mathsf{aux}_{i,j}))_{j\in[l_i]}$, where l_i is the length of A'_i. Assume there are $\leq c\kappa$ elements in each A'_i and $\geq c'\kappa$ elements in A'_i such that $\mathsf{aux}_{i,j} \neq \perp$ Further, assume for each $i \in [n], j \in [l_i]$ we have that $a'_{i,j}$ is a sample from χ under the condition that $\varphi_i(a'_{i,j}) = 1$ holds.

Now let $\mathsf{A} \leftarrow \mathsf{MergeSamp}((\mathsf{A}'_i)_{i\in[n]})$ and parse $\mathsf{A} = ((a_i, \mathsf{aux}_i))_{i\in[2c^\dagger\kappa n]}$. We get that for all $i \in [n]$ there is an element (a'_i, aux_i) from A'_i with $\mathsf{aux}_i \neq \perp$ in A with all but negligible probability.

Lemma 5. *If all the precondition of Theorem 7 are met, at the end of the sampling process for all $j \in [n]$ we have that A'_j is empty.*

Proof (Proof of Lemma 5). For each $i \in [2c\kappa n/c^\dagger]$ we remove the last element from A'_j if $\varphi_j(a)$ holds and A'_j is not empty yet. By Lemma 4 we know that a follows the distribution χ. We know that A'_j is no longer than $c\kappa$. By precondition 1 of Theorem 7 of $\mathsf{MergeSamp}$ we know that the probability of a sample of χ fulfilling φ_j is $> c^\dagger n^{-1}$. Therefore, the expected number of elements that fulfill φ_j is $2c\kappa$. Let S denote the number of times φ_j gets checked in $\mathsf{MergeSamp}$ and holds true. By multiplicative Chernoff bound we get that $\Pr[S \leq c\kappa] \leq \left(\frac{e^{-1/2}}{(1/2)^{1/2}}\right)^{2c\kappa} < 0.9^{2c\kappa}$, which is negligible in κ. Therefore, the number of times φ_j gets checked in $\mathsf{MergeSamp}$ and holds is $> \kappa c$ with all but negligible probability.

Proof (Proof of Theorem 7). For each $j \in [n]$ a precondition of Theorem 7 states the number of elements of A'_j for which $\mathsf{aux} \neq \perp$ is $\geq c'\kappa$. By Lemma 5 all of them are moved from A'_j at some point.

By the structure of $\mathsf{MergeSamp}$ once a sample-auxiliary information pair $(a'_{j,i}, \mathsf{aux}_{j,i})$ becomes $(a_i^{(j)}, \mathsf{aux}_i^{(j)})$ for some i it ends up in A unless $\varphi_{j'}(a_{j,i}) = 1$ for some $j' > j$. By precondition 2 of Theorem 7 this happens with probability $< c^*$. Because there are $\geq c'\kappa$ pairs such that the auxiliary information is $\neq \perp$ and the processes are independent the probability that no pair $(a_{j,i}, \mathsf{aux}_{j,i})$ with $\mathsf{aux} \neq \perp$ is in A is $< {c^*}^{c'\kappa}$, which is negligible in κ.

5 Two-Server Batch PIR

In the following we show how to use puncturable pseudorandom sets from Construction 1 with parameters $\alpha = N/B$ and $\beta = N/B^2$ and batch unbiased sampling to create a two server batch PIR protocol.

Construction 2 *In the following we describe the client and server algorithms of the PIR protocol. Let $p = (\mathsf{Gen}, \mathsf{Expand}, \mathsf{IntersectSet}, \mathsf{Puncture}, \mathsf{InUncertaintySet})$ be a PPRS with range $[N]$ and size ℓ.*

- ***Offline Phase/Preprocessing:***
  - ***Algorithm*** $\mathsf{client}_{1,1}$*: On input the computational security parameter λ, statistical security parameter κ, the database size N, and the batch parameter B, generate keys k_i via $\mathsf{Gen}_f(\lambda)$ for all $i \in [\kappa B]$ the client generates $k_i \leftarrow \mathsf{Gen}(\lambda)$ and chooses a uniformly random nonce $\mathsf{nce}_i \leftarrow \{0,1\}^\lambda$. It outputs a list $c_1 = (k_i, \mathsf{nce}_i)$ and a state $\mathsf{st}_1 = c_1$.*
  - ***Algorithm*** server_1*: On input the database DB, a list $c_1 = (k_i, \mathsf{nce}_i)$ and a key K, where the keys k_i represent sets $\mathcal{S}_{k_i}, i \in [\kappa B]$, compute the hint*

$$h_i = \mathsf{PRF}_K(\mathsf{nce}_i) \oplus \bigoplus_{j \in S_{k_i}} \mathsf{DB}[j]$$

for all $i \in [\kappa B]$ and output them as a list s_1.

- ***Algorithm*** $\mathsf{client}_{1,2}$*: On input a list* s_1 *of hints and a state* st_1 *consisting of the keys* $h_i, i \in [\kappa B]$*, create and returns, as the preprocessing, the list of pairs* $H = ((k_i, h_i, \mathsf{nce}_i))_{i\in[\lambda B]}$ *together with the database size* N *and the batch size* B*.*

– ***Online Phase:***

- ***Algorithm*** $\mathsf{client}_{2,1}$*: On input a preprocessing* H *and a list of queries* $\mathcal{Q} \in [N]^B$ *do as follows:*
  - *Let* $(\mathsf{A}'_j)_{j\in[B]} \leftarrow \mathsf{SplitSamp}((a_i = k_i, \mathsf{aux}_i = (h_i, \mathsf{nce}_i))_{i\in[\lambda B]})$ *with the predicates* $(\varphi_j)_{j\in[B]}$ *where* $\varphi_j(k)$ *computes* $q_j \in \mathsf{IntersectSet}(k, \mathcal{Q})$
  - *For each* $j \in [B]$ *parse* A'_j *as* $(a_{i,j} = k_{i,j}, \mathsf{aux}_{i,j} = (h_{i,j}, \mathsf{nce}_{i,j}))_{i\in[|\mathsf{A}'_j|]}$
  - *For each* $j \in [B], i \in [|\mathsf{A}'_j|]$ *let* $k^*_{i,j} \leftarrow \mathsf{Puncture}(k_{i,j}, q_j)$*.*
  - *For each* $j \in [B]$ *set* $\mathsf{A}''_j = \{k^*_{i,j}, (h_{i,j}, q_j, \mathsf{nce}_{i,j})\}_{i\in[|\mathsf{A}'_j|]}$
  - *Let* $\mathsf{A} \leftarrow \mathsf{MergeSamp}(\{\mathsf{A}''_j\}_{j\in[B]})$ *with the predicates* $\{\varphi_j\}_{j\in[B]}$ *where* $\varphi_j(k)$ *computes* $\mathsf{InUncertaintySet}(k, q_j)$
  - *Parse* $\mathsf{A} = \{(a_i = k^*_i, \mathsf{aux}_i = (q_i, h_i, \mathsf{nce}_i))\}_{i\in[|\mathsf{A}|]}$
  - *For all* $i \in [|\mathsf{A}|]$ *for which* $\mathsf{nce}_i = \perp$ *reset* $\mathsf{nce}_i \leftarrow \{0,1\}^\lambda$ *to a uniformly random value.*
  - *Output* $c_2 = \{k^*_i, \mathsf{nce}_i\}_{i\in[|\mathsf{A}|]}$ *and* $\mathsf{st}_2 = \mathsf{A}$*.*
- ***Algorithm*** server_2*: On input a list of punctured keys* $c_2 = \{(k^*_i, \mathsf{nce}_i)\}_{i\in[|\mathsf{A}|]}$ *and a database* DB *and a PRF key* K*, compute, for each* $i \in [|\mathsf{A}|]$*, the parity*

$$p_i = \mathsf{PRF}_K(\mathsf{nce}_i) \oplus \bigoplus_{j\in S_{k^*_i}} \mathsf{DB}[j]$$

  and output the list $s_2 = \{p_i\}_{i\in[|\mathsf{A}|]}$*.*
- ***Algorithm*** $\mathsf{client}_{2,2}$*: On input a list of parities* s_2 *and a state* $\mathsf{st}_2 = \mathsf{A} = \{(a_i = k^*_i, \mathsf{aux}_i = (q_i, h_i, \mathsf{nce}_i))\}_{i\in[|\mathsf{A}|]}$*, initialize the empty output list* out*. For* $i \in [|\mathsf{A}|]$ *do:*
  - *If* $(h_i, q_i, \mathsf{nce}_i) \neq \perp$ *and* W *does not contain an element with first component* q_i *then include* $(q_i, p_i \oplus h_i)$ *into the list* out*.*

  Output the list out*.* [5]

Theorem 8. *The PIR in construction 2 satisfies computational correctness (see Definition 7) if the punctured pseudorandom sets have pseudorandom outputs and pseudorandom uncertainty sets (see Definition 6).*

Proof. Theorem 5 gives us a tool to upper bound the number of elements in each bin A'_i, as well as a *lower bound* on the number of *useful* elements in each bin, that is the elements that are endowed with a hint that comes from the preprocessing H.

Specifically, the distribution χ in Theorem 5 corresponds to the distribution $\mathcal{S}_k$ for a fresh key k, The parameter n corresponds to B, and φ_j corresponds to $q_j \in \mathcal{S}_k$, which is equivalent to $q_j \in \mathsf{IntersectSet}(k, \mathcal{Q})$.

However, in order to apply Theorem 5, we first have to establish its preconditions. Specifically, we need to bound the following probabilities:

[5] Since our algorithm returns the replies to the queries in $\mathcal{Q}$ in unordered fashion, we add the database position q_i to the replies.

– For each $q \in \mathcal{Q}$ we need to upper-bound the probability that q is in a freshly generated $\mathcal{S}_k$
– For each $q \in \mathcal{Q}$ we need to lower-bound the probability that $q \in \mathcal{S}_k$, but no other $q' \in \mathcal{Q} \setminus \{q\}$ is in $\mathcal{S}_k$ (again for a freshly sampled k)

The sets $\mathcal{S}_k$ follow rather complicated and thus hard to analyze distributions, which makes it hard to compute the above probabilities directly for the $\mathcal{S}_k$. However, we can use the fact that these distributions are pseudorandom (Theorem 2) to bound these two probabilities. Specifically, bounding these probabilities for truly random sets is rather routine.

By construction, the sets $\mathcal{S}_k$ have size exactly $\alpha - \beta$, which we call ℓ, in the following we only need that there exist constants $\tilde{c}, c > 0$

$$\tilde{c}N/B < \ell < cN/B.$$

which follows from the choice of parameters $\alpha = N/B$ and $\beta = N/B^2$.

Now fix an element $q \in \mathcal{Q}$. With the above we have $\Pr[q \in \mathcal{S}_k] < c\frac{N}{BN} = c/B - \mathsf{negl}(\kappa)$. This establishes the first precondition of Theorem 5.

Fix a $q \in \mathcal{Q}$. We will now bound the probability that a $\mathcal{S}_k$ contains q but no other $q' \in \mathcal{Q}$. Let $\mathcal{T} \subseteq [N]$ be a uniformly chosen set of size ℓ. We can bound this probability that $\mathcal{T}$ contains q but doesn't contain any $q' \in \mathcal{Q} \setminus \{q\}$ by

$$\begin{aligned}
\Pr[q \in \mathcal{T} \text{ and } \mathcal{T} \cap (\mathcal{Q} \setminus \{q\}) = \emptyset] &= \Pr[q \in \mathcal{T}] \cdot \Pr[\mathcal{T} \cap (\mathcal{Q} \setminus \{q\}) = \emptyset | q \in \mathcal{T}] \\
&= \frac{\ell}{N} \cdot \prod_{i \in [B-1]} \left(1 - \frac{\ell - 1}{(N-i)}\right) \\
&> \tilde{c}\frac{N}{BN} \cdot \left(1 - \frac{cN/B}{N-B}\right)^B \\
&> \tilde{c}/B \cdot \left(1 - \frac{2c}{B}\right)^B \qquad (1) \\
&> \frac{\tilde{c} \cdot e^{-4c}}{B} \qquad (2)
\end{aligned}$$

The inequality 1 holds for $N > 2B$, which we can assume without loss of generality because our protocol becomes worse than the trivial solution for $2B \geq N$, and inequality 2 holds as $B \geq 4c$ using the first-order approximation $1 - x \geq e^{-2x}$ for $x \in [0, 1/2]$.

Consequently, by Theorem 2 we also get that

$$\begin{aligned}
\Pr[q \in \mathcal{S}_k \text{ and } \mathcal{S}_k \cap (\mathcal{Q} \setminus \{q\}) = \emptyset] &\geq \Pr[q \in \mathcal{T} \text{ and } \mathcal{T} \cap (\mathcal{Q} \setminus \{q\}) = \emptyset] - \mathsf{negl}(\kappa) \\
&> \frac{\tilde{c} \cdot e^{-4c}}{B} - \mathsf{negl}(\kappa).
\end{aligned}$$

We can now invoke Theorem 5 and conclude that for each $j \in [B]$ we have A'_j is at most $\lambda 3c/2$ long and contains at least $c'\lambda/2$ elements with hints that come from H.

Theorem 7 gives us a tool guarantee that A contains at least one element that is endowed with a hint for q that comes from the preprocessing H for all $q \in \mathcal{Q}$.

Specifically, the distribution χ in Theorem 7 corresponds to the distribution $\tilde{\mathcal{S}}_{k^*}$ for a fresh key k, uniform $q \leftarrow_\$ \mathcal{S}_k$, and $k^* \leftarrow \mathsf{Puncture}(k, q)$. The parameter n corresponds to B, and φ_j corresponds to $q_j \in \tilde{\mathcal{S}}_{k^*}$.

Next, we want to invoke Theorem 7. To establish the preconditions we need to bound the following probabilities:

- For each $q \in \mathcal{Q}$ we need to lower-bound the probability that q is in $\tilde{\mathcal{S}}_{k^*}$ for a freshly generated k, uniformly random $q' \leftarrow_\$ \mathcal{S}_k$ and $k^* \leftarrow \mathsf{Puncture}(k, q')$
- For each $q \in \mathcal{Q}$, we want to lower-bound the probability that $(\mathcal{Q} \setminus \{q\}) \cap \tilde{\mathcal{S}}_{k^*} = \emptyset$ for a freshly generated k conditioned $q \in \mathcal{S}_k$ with $k^* \leftarrow \mathsf{Puncture}(k, q)$

As above, we will rely on the fact that the sets $\tilde{S}_{k^*}$ are pseudorandom (Theorem 2) to bound these two probabilities.

By construction, the sets $\tilde{\mathcal{S}}_{k^*}$ have size exactly $(N/\alpha) \cdot (\beta + 1)$, which we call ℓ', in the following we only need that there exist constants $\tilde{c}, c > 0$

$$\tilde{c}N/B < \ell' < cN/B.$$

which follows from the choice of parameters $\alpha = N/B$ and $\beta = N/B^2$.

Now fix an element $q \in \mathcal{Q}$. With the above we have $\Pr[q \in \tilde{\mathcal{S}}_{k^*}] > \tilde{c}\frac{N}{BN} = \tilde{c}/B - \mathsf{negl}(\kappa)$. This establishes the first precondition of Theorem 7.

Fix a $q \in \mathcal{Q}$. We will now bound the probability that for $k \leftarrow \mathsf{Gen}(\lambda)$ under the condition that $q \in \mathcal{S}_k$ and $k^* \leftarrow \mathsf{Puncture}(k, q)$ we have for no other $q' \in \mathcal{Q}$ that $q' \in \tilde{S}_{k^*}$. Let $\mathcal{T} \subseteq [N]$ be a uniformly chosen set of size ℓ' under the condition that it contains q. We can bound this probability that $\mathcal{T}$ doesn't contain any other $q' \in \mathcal{Q}$ by

$$\begin{aligned}
\Pr[\mathcal{T} \cap (\mathcal{Q} \setminus \{q\})] &= \prod_{i \in |[B-1]|} 1 - \frac{\ell - 1}{(N - i)} \\
&> \left(1 - \frac{cN/B}{N - B}\right)^B \\
&> \left(1 - \frac{2c}{B}\right)^B \qquad (3) \\
&> e^{-4c} \qquad (4)
\end{aligned}$$

The inequality 3 holds for $N > 2B$, which we can assume without loss of generality because our protocol becomes worse than the trivial solution for $2B \geq N$, and the inequality 2 holds as $B \geq 4c$ by using the first-order approximation $1 - x \geq e^{-2x}$ for $x \in [0, 1/2]$.

Consequently, by Theorem 1 and Theorem 3 we also get that

$$\Pr_{\substack{k\leftarrow\mathsf{Gen}(\lambda)|q\in\mathcal{S}_k\\ k^*\leftarrow\mathsf{Puncture}(k,q)}}[\tilde{\mathcal{S}}_{k^*}\cap(\mathcal{Q}\setminus\{q\})=\emptyset]\geq\Pr[\mathcal{T}\cap(\mathcal{Q}\setminus\{q\})=\emptyset]-\mathsf{negl}(\kappa)$$
$$>e^{-4c}-\mathsf{negl}(\kappa).$$

This establishes the second precondition of Theorem 7.

We can now invoke Theorem 5 and conclude that for each $q\in\mathcal{Q}$ we have A contains at least one element $(k_i^*,(q_i^*,h_i,\mathsf{nce}_i))$ such that $q=q_i^*$, $k_i^*=\mathsf{Puncture}(k,q)$ where (k_i,h_i,nce_i) is in H with all but negligible probability. Therefore, we have that $h_i=\mathsf{PRF}_K(\mathsf{nce}_i)\oplus\bigoplus_{j\in S_k}\mathsf{DB}[j]$, and from the correctness of the puncturable pseudorandom set we get that $S_{k_i^*}=S_{k_i}\setminus\{q\}$. For this element the client gets a response $p=\mathsf{PRF}(\mathsf{nce}_i)\oplus\bigoplus_{j\in S_{k_i^*}}\mathsf{DB}[j]$, which means $p\oplus h=\bigoplus_{j\in S_k}\mathsf{DB}[j]\oplus\bigoplus_{j\in S_{k^*}}\mathsf{DB}[j]=\mathsf{DB}[q]$. Thus the protocol is computationally correct.

Theorem 9. *The PIR construction 2 satisfies computational privacy (see Definition 7).*

We prove this statement using the properties of $\mathsf{SplitSamp}$, the puncturable pseudorandom sets, and $\mathsf{MergeSamp}$ in Appendix A.1.

Theorem 10. *Given that* PRF *is a pseudorandom function, the Construction 2 satisfies server privacy.*

Proof. Our simulator is given as follow:

- When interacting with the offline-phase, the for each (k_i,nce_i) the simulator chooses a fresh uniformly random bit $r_i\leftarrow\{0,1\}$ and sets $h_i=r_i$. It further stores the tuple (nce_i,k_i,r_i)
- For each pair (k_i^*,nce_i) in the online phase, the simulator first checks if there is a tuple (nce_j,k_j,r_j) stored in the step phase. If so it sets $\{q_i\}=\mathcal{S}_{k_j}\setminus\mathcal{S}_{k_i^*}$ and sets its response to $p_i=r_j\oplus\mathsf{DB}[q_i]$. Otherwise if not such tuple (nce_j,k_j,r_j) was stored it sets p_i to a uniformly random value.compute

Consider the following hybrids:

- Hybrid 0: This is the real experiment.
- Hybrid 1: This is identical to the real experiment, except that the experiment aborts if two different queries use the same nonce. By a birthday union bound this event has only negligible probability $O(N\cdot 2^{-\lambda/2})$, hence hybrid 0 and hybrid 1 are statistically close
- Hybrid 2: In this hybrid we replace the pseudorandom function PRF_K by a uniformly random function R. Computational indistinguishability follows routinely from the PRF security of PRF.
- Hybrid 3: In this hybrid we compute all h_j via $h_j=r_j$ and all corresponding p_i via $p_i=r_j\oplus\mathsf{DB}[q_i]$. This is identically distributed to hybrid 2, as by the condition introduced in hybrid 1 no nonce nce is used twice.

– Hybrid 4: In this hybrid we undo the modification of hybrid 1. Hence the statistical distance between hybrids 3 and 4 is at most $O(N \cdot 2^{-\lambda/2})$. This is the ideal experiment.

This concludes the proof.

Remark 2. The protocol has the following runtimes and communication sizes:

– Offline client computation: $\tilde{\theta}(B)$
– Offline server computation: $\tilde{\theta}(N)$
– Offline communication: $\tilde{\theta}(B)$
– Client storage between offline and online stage: $\tilde{\theta}(B)$
– Online client computation: $\tilde{\theta}(N/B + B^2)$
– Online server computation: $\tilde{\theta}(N)$
– Online communication: $\tilde{\theta}(N/B)$

where $\tilde{\theta}$ ignores all log factors and λ.

We prove the remark in Appendix A.2.

References

ABD+21. Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part III*, volume 13044 of *LNCS*, pages 94–125, Raleigh, NC, USA, November 8–11, 2021. Springer, Cham, Switzerland.

ABG+24. Hilal Asi, Fabian Boemer, Nicholas Genise, Muhammad Haris Mughees, Tabitha Ogilvie, Rehan Rishi, Guy N Rothblum, Kunal Talwar, Karl Tarbe, Ruiyu Zhu, et al. Scalable private search with wally. *arXiv preprint arXiv:2406.06761*, 2024.

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.

ALOS22. Diego F. Aranha, Chuanwei Lin, Claudio Orlandi, and Mark Simkin. Laconic private set-intersection from pairings. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 111–124, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

BDG15. Nikita Borisov, George Danezis, and Ian Goldberg. DP5: A private presence service. *PoPETs*, 2015(2):4–24, April 2015.

BGI14. Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In Hugo Krawczyk, editor, *PKC 2014*, volume 8383 of *LNCS*, pages 501–519, Buenos Aires, Argentina, March 26–28, 2014. Springer, Berlin, Heidelberg, Germany.

BGIK22. Elette Boyle, Niv Gilboa, Yuval Ishai, and Victor I. Kolobov. Programmable distributed point functions. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part IV*, volume 13510 of *LNCS*, pages 121–151, Santa Barbara, CA, USA, August 15–18, 2022. Springer, Cham, Switzerland.

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

BV11. Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106, Palm Springs, CA, USA, October 22–25, 2011. IEEE Computer Society Press.

CGKS95. Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, 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, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 3–33, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

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

DRRT18. Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling private contact discovery. *PoPETs*, 2018(4):159–178, October 2018.

Gen09. Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178, Bethesda, MD, USA, May 31 – June 2, 2009. ACM Press.

GKL10. Jens Groth, Aggelos Kiayias, and Helger Lipmaa. Multi-query computationally-private information retrieval with constant communication rate. In Phong Q. Nguyen and David Pointcheval, editors, *PKC 2010*, volume 6056 of *LNCS*, pages 107–123, Paris, France, May 26–28, 2010. Springer, Berlin, Heidelberg, Germany.

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*, volume 3580 of *LNCS*, pages 803–815, Lisbon, Portugal, July 11–15, 2005. Springer, Berlin, Heidelberg, Germany.

GSW13. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Berlin, Heidelberg, Germany.

GZS24. Ashrujit Ghoshal, Mingxun Zhou, and Elaine Shi. Efficient pre-processing PIR without public-key cryptography. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VI*, volume 14656 of *LNCS*, pages 210–240, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

HDCGZ23. Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with tiptoe. In *Proceedings of the 29th symposium on operating systems principles*, pages 396–416, 2023.

HPPY24. Alexander Hoover, Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. Plinko: Single-server PIR with efficient updates via invertible PRFs. Cryptology ePrint Archive, Report 2024/318, 2024.

HV17. Carmit Hazay and Muthuramakrishnan Venkitasubramaniam. Scalable multi-party private set-intersection. In Serge Fehr, editor, *PKC 2017, Part I*, volume 10174 of *LNCS*, pages 175–203, Amsterdam, The Netherlands, March 28–31, 2017. Springer, Berlin, Heidelberg, Germany.

IKOS04. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In László Babai, editor, *36th ACM STOC*, pages 262–271, Chicago, IL, USA, June 13–16, 2004. ACM Press.

KMP+17. Vladimir Kolesnikov, Naor Matania, Benny Pinkas, Mike Rosulek, and Ni Trieu. Practical multi-party private set intersection from symmetric-key techniques. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1257–1272, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

KS05. Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257, Santa Barbara, CA, USA, August 14–18, 2005. 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 ACM STOC*, 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 *Theory of Cryptography Conference*, pages 406–435. Springer, 2023.

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

LPR10. Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 1–23, French Riviera, May 30 – June 3, 2010. Springer, Berlin, Heidelberg, Germany.

LS15. Helger Lipmaa and Vitaly Skachek. Linear batch codes. In *Coding Theory and Applications: 4th International Castle Meeting, Palmela Castle, Portugal, September 15-18, 2014*, pages 245–253. Springer, 2015.

MIR23. Muhammad Haris Mughees, Sun I, and Ling Ren. Simple and practical amortized sublinear private information retrieval. Cryptology ePrint Archive, Report 2023/1072, 2023.

PSWW18. Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient circuit-based PSI via cuckoo hashing. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 125–157, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

PSZ14. Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 797–812, San Diego, CA, USA, August 20–22, 2014. USENIX Association.

RR17. Peter Rindal and Mike Rosulek. Malicious-secure private set intersection via dual execution. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1229–1242, Dallas, TX, USA, October 31 – November 2, 2017. 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, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 641–669, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.

SCH+21. Sudheesh Singanamalla, Suphanat Chunhapanya, Jonathan Hoyland, Marek Vavrusa, Tanya Verma, Peter Wu, Marwan Fayed, Kurtis Heimerl, Nick Sullivan, and Christopher A. Wood. Oblivious DNS over HTTPS (ODoH): A practical privacy enhancement to DNS. *PoPETs*, 2021(4):575–592, October 2021.

SG16. Natalia Silberstein and Anna Gál. Optimal combinatorial batch codes based on block designs. *Designs, Codes and Cryptography*, 78(2):409–424, 2016.

SSLD22. Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. Private approximate nearest neighbor search with sublinear communication. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 911–929. IEEE, 2022.

SWP09. DR Stinson, Ruizhong Wei, and Maura B Paterson. Combinatorial batch codes. *Advances in Mathematics of Communications*, 3(1):13–27, 2009.

Yeo23. Kevin Yeo. Lower bounds for (batch) PIR with private preprocessing. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, 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, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 395–425, Lyon, France, April 23–27, 2023. Springer, Cham, Switzerland.

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 (SP)*, pages 4296–4314. IEEE, 2024.

ZSF24. Mingxun Zhou, Elaine Shi, and Giulia Fanti. Pacmann: Efficient private approximate nearest neighbor search. *Cryptology ePrint Archive*, 2024.

A Proofs for Construction 2

A.1 Proof of Theorem 9

Here, we prove Theorem 9, which establishes the security of Construction 2.

Proof. The simulator for the $\mathsf{2sbPIR}$ scheme given in Construction 2 is very simple: It generates c_2 by generating $2c\lambda n/c^\dagger$ fresh keys, each punctured at a uniformly random point.

$\mathsf{Sim}(1^\lambda, N, B)$:

- For $i \in [2c\lambda n/c^\dagger]$:
  - Let $k \leftarrow \mathsf{Gen}(\lambda)$
  - Sample $q \leftarrow_\$ \mathcal{S}_k$ uniformly at random
  - Let $k_i^* \leftarrow \mathsf{Puncture}(k, q)$

– Output $c_2 = (k_i^*)_{i \in [2c\lambda n/c^\dagger]}$.

We will now argue that the message c_2 generated by $\mathsf{client}_{2,1}$ in a real run of the protocol is computationally indistinguishable from the output of $\mathsf{Sim}(1^\lambda, N, B)$ in the view of the second server.

– Hybrid 0: This is the real experiment, i.e. we compute c_2 as in the real run of the protocol. Specifically, c_2 is generated as follows.
  1. $(\mathsf{A}'_j = (a_{i,j} = k_{i,j}, \mathsf{aux}_{i,j} = h_{i,j})_{i \in [|\mathsf{A}_j|]})_{j \in [B]} \leftarrow \mathsf{SplitSamp}((a_i = k_i, \mathsf{aux}_i = h_i)_{i \in [M]})$ with the predicates $(\varphi_j)_{j \in [B]}$ where $\varphi_j(k)$ computes whether $q_j \in \mathsf{IntersectSet}(k, \mathcal{Q})$
  2. For each $j \in [B], i \in [|\mathsf{A}'_j|]$ let $k^*_{i,j} \leftarrow \mathsf{Puncture}(k_{i,j}, q_j)$.
  3. For each $j \in [B]$ set $\mathsf{A}''_j = (k^*_{i,j}, h_{i,j})_{i \in [|\mathsf{A}'_j|]}$
  4. $\mathsf{A} \leftarrow \mathsf{MergeSamp}((\mathsf{A}''_j)_{j \in [B]})$ with the predicates $(\varphi_j)_{j \in [B]}$ where $\varphi_j(k)$ computes $\mathsf{InUncertaintySet}(k, q_j)$
  5. Parse $\mathsf{A} = ((a_i = k^*_i, \mathsf{aux}_i = (q_i, h_i)))_{i \in [|\mathsf{A}|]}$
  6. Set $c_2 = (k^*_i)_{i \in [|\mathsf{A}|]}$
– Hybrid 1: This hybrid is identical to hybrid 0, except that we modify step 2 in the generation of c_2. Specifically, instead of setting $k^*_{i,j} \leftarrow \mathsf{Puncture}(k_{i,j}, q_j)$ we do the following:
  - Sample a fresh key k as $k \leftarrow \mathsf{Gen}(\lambda)$.
  - Sample an element q' as $q' \leftarrow \mathcal{S}_k$.
  - Compute the punctured key k^* as $k^* = \mathsf{Puncture}(k, q')$, conditioned on $q \in \tilde{\mathcal{S}}_k$ (i.e. reject an resample if this condition does not hold)
  - Set $k^*_{i,j} = k^*$

  We claim that hybrid 0 and hybrid 1 are computationally indistinguishable. First, observe that by Theorem 5 it holds that for each $j \in [B]$ the keys $(k_{i,j})_{i \in [\mathsf{A}'_j]}$ in hybrid 0 are all uniform conditioned on $q_j \in \mathsf{IntersectSet}(k_{i,j}, \mathcal{Q})$ holding, which is equivalent to $q_j \in \mathcal{S}_{k_{i,j}}$.
  Hence, the punctured keys $k^*_{i,j}$ in hybrid 0 follow the same distribution as in the *pseudorandomness of punctured elements* game (Definition 6) for $b = 0$. On the other hand, in hybrid 1 all keys $k^*_{i,j}$ follow the same distribution as in the *pseudorandomness of punctured elements* game (Definition 6) for $b = 1$. Thus, a sequence of λB sub-hybrids going over all elements in all bins $j \in [B]$, we can use the pseudorandomness of punctured elements property to establishes computational indistinguishability of hybrids 0 and 1.
– Hybrid 2: In this hybrid we choose $c_2 = \mathsf{Sim}(1^\lambda, N, B)$, i.e. this is the ideal experiment.
  We claim that hybrid 1 and hybrid 2 are identically distributed.
  First note that in hybrid 1 for all $j \in [B]$ A''_j is a list of tuples $(k^*_{i,j}, \mathsf{aux}_{i,j})$ such that each $k^*_{i,j}$ is uniform under the condition that $q_j \in \mathcal{S}_{k_{i,j}}$. Hence the preconditions of Theorem 6 are satisfied, which establishes that the output $\mathsf{A} = ((a_i = k^*_i, \mathsf{aux}_i = (q_i, h_i)))_{i \in [|\mathsf{A}|]}$ of $\mathsf{MergeSamp}((\mathsf{A}''_j)_{j \in [B]})$ is such that all k^*_i are uniformly chosen keys punctured at uniformly random points in their domain.
  Note that it always holds that $|\mathsf{A}| = 2c\lambda n/c^\dagger$. This means that $c_2 = (k^*_i)_{i \in |\mathsf{A}|}$ follows exactly the distribution generated by $\mathsf{Sim}(1^\lambda, N, B)$.

This concludes the proof.

A.2 Proof of Remark 2

Here, we prove Remark 2 that establishes the runtime and communication of Construction 2.

Proof. For database of size N and batch size B we choose parameters for the puncturable pseudorandom set $\alpha = N/B$ and $\beta = N/B^2$. In the first phase, client generates λB new keys and receives λB responses. Generating a new key takes time $\tilde{\theta}(1)$. Therefore, the runtime is $\tilde{\theta}(B)$. The server has to compute λB inner products where the non-zero entries are elements of the pseudorandom sets. These sets have size $\alpha - \beta \in \tilde{\theta}(N/B)$. Therefore the runtime of server_1 is $\tilde{\theta}(N)$. Overall, the communication between client and server_1 is λB many keys and one bit-responses totalling to $\tilde{\theta}(B)$ communication.

The list A has length $\tilde{\theta}(B)$. server_2 has to compute an inner product for each uncertainty set. The number of non-zero entries are the number of elements in the uncertainty set, which have size $(N/\alpha)(\beta + 1) = B(N/B^2 + 1) = \tilde{\theta}(N/B)$. Therefore, the computation of server_2 is $\tilde{\theta}(\lambda N)$. The punctured representation of these sets are of size $\tilde{\theta}(\lambda + \beta)$. So the communication in the second phase is $\tilde{\theta}(\lambda B(N/B^2)) = \tilde{\theta}(\lambda N/B)$.

We know analyse the runtime of the client in the second phase. It operates in three stages: $\mathsf{SplitSamp}$, puncturing, and $\mathsf{MergeSamp}$. First we analyse $\mathsf{SplitSamp}$.

In $\mathsf{SplitSamp}$ has to compute $\mathsf{IntersectSet}(k, \mathcal{Q})$ and sample a new set under the condition that a certain φ holds $\tilde{\theta}(\lambda B)$ many times because these operations only have to be done once per key in $(\mathsf{A}'_j)_{j\in[B]}$. In Theorem 8 we prove that $\sum_{j\in[B]} |\mathsf{A}'_j| \in \tilde{\theta}(\lambda B)$. Because $\mathsf{IntersectSet}(k, \mathcal{Q})$ takes time $\tilde{\theta}(N/B^2 + B)$ and sample a new set under the condition that a certain φ holds his in $\tilde{\theta}(1)$ this part takes time $\tilde{\theta}(N/B + B^2)$. Once $\mathsf{IntersectSet}(k, \mathcal{Q})$ is computed for all the relevant sets it is a constant time operation to look up $q \in \mathsf{IntersectSet}(k, \mathcal{Q})$ for a specific q and k. This operation has to be done λB^2 many times in $\mathsf{SplitSamp}$. Therefore, the runtime of $\mathsf{SplitSamp}$ is $\tilde{\theta}(N/B + B^2)$.

Then each key in $(\mathsf{A}'_j)_{j\in[B]}$ has to be punctured. Puncturing costs time $\tilde{\theta}(N/B^2)$ and has to be done for $\tilde{\theta}(B)$ many elements. Therefore, the runtime for this step is in $\tilde{\theta}(N/B)$.

In $\mathsf{MergeSamp}$ the inner loop gets run $\tilde{\theta}(B^2)$ many times and the operation therein are in $\tilde{\theta}(1)$. Thus, the computation of the client is in $\tilde{\theta}(N/B + B^2)$.