Single Server PIR via Homomorphic Thorp Shuffles

Ben Fisch Arthur Lazzaretti Zeyu Liu Charalampos Papamanthou

Yale University

Abstract

Private Information Retrieval (PIR) is a two player protocol where the client, given some query $x \in [N]$ interacts with the server, which holds a N-bit string DB in order to privately retrieve $\mathsf{DB}[x]$. In this work, we focus on the single server client-preprocessing model, initially idealized by Corrigan-Gibbs and Kogan (EUROCRYPT 2020), where the client and server first run some joint preprocessing algorithm, after which the client can retrieve elements of the server's string DB privately in time sublinear in N.

All known constructions of single server client-preprocessing PIR rely on one of the following two paradigms: (1) a linear-bandwidth offline phase where the client downloads the whole database from the server, or (2) a sublinear-bandwidth offline phase where however the server has to compute a large-depth ($O_\lambda(N)$) circuit under FHE in order to execute the preprocessing phase.

In this paper, we construct a single server client-preprocessing PIR scheme which achieves both sublinear offline bandwidth (the client does not have to download the whole database offline) and a low-depth (i.e. $O_\lambda(1)$), highly parallelizable preprocessing circuit. We estimate that on a single thread, our scheme's preprocessing time should be more than 350x times faster than in prior single server client-preprocessing PIR constructions. Moreover, with parallelization, the latency reduction would be even more drastic. In addition, this construction also allows for updates in $O_\lambda(1)$ time, something not achieved before in this model.

Contents

1 Introduction

Private information retrieval (PIR) [14, 34] is an important cryptographic primitive, which allows a client to fetch a data entry from the database without revealing which entry it is. However, one major practical limitation of PIR is that to fetch an entry, it takes $O(N)$ time, where N is the number of data entries in the database [7]. In 2020, Corrigan-Gibbs and Kogan [18] introduced the notion of online/offline PIR, where during the offline phase one server processes the database, generates a *hint*, and sends it to the client. In the online phase, the client can then use this hint to query the data entry it wants from another server. In this paradigm, the online time can be sublinear in N. Subsequent to [18], many other works have pushed our understanding of this PIR model [33, 41, 37].

One major limitation of the initial idea by [18] is that it requires two non-colluding servers. The client cannot sends its queries to the same server that ran the preprocessing or this breaches privacy. Thus, a series of work study how to get optimal complexities in the single server scenario [17, 59, 36], using fully homomorphic encryption (FHE) and other sophisticated techniques to transform two-server constructions into a single-server construction: the server generates the hint using FHE and thus cannot see the hints.

While these works achieve near-optimal asymptotic complexities, the schemes are far from being practical: the homomorphic circuit computed under encryption by the server requires $O_\lambda(N)$ depth [17, 36, 59], which concretely comes out to millions of levels of multiplication for most if not all PIR use cases.[1] The depth of a circuit is arguably the most impactful metric to optimize for, when constructing a circuit to evaluate under FHE (other than its size, given that all the existing FHE schemes are most efficient in the leveled setting, i.e., working over circuits with a small fixed depth). It is an open question whether we can devise a low-depth (i.e. $O_\lambda(1)$) circuit with $O_\lambda(N)$ gates whose output can be used as the hints for a client-preprocessing PIR scheme.

To circumvent the need for such circuit, another line of work [60, 46] instead proposes to allow the server to stream the whole database to the client during the offline phase. The client computes the hints locally by itself under plaintext (in a streaming fashion, while maintaining $o(N)$ storage at all times). In this way, the server does not see the hints, and FHE is not required. Of course, the major disadvantage of this approach is that there is a $\Omega(N)$ communication cost during the offline phase.

So, the current state for client-preprocessing single server PIR is that one can either download the whole database at the preprocessing step, or have the server evaluate an $O_\lambda(N)$ depth circuit encrypted gate-by-gate using FHE to compute the hints in near-linear time with sublinear bandwidth.

This motivates the following question:

Can we construct a single-server offline/online PIR scheme that achieves (1) sublinear amortized time, (2) sublinear offline communication, and (3) use a polylog/constant depth and linear size circuit to compute the hints?

In this paper, we answer this question in the affirmative, by constructing such a scheme, which additionally has constant update time.

[1] The circuit requires linear depth to compute parities of $O(\sqrt{N})$ subsets of $O(\sqrt{N})$ database indices in near-linear time. Naively, one can achieve $O(1)$ depth by blowing up the size of the circuit to $O(N^2)$, but the amortized time of the PIR scheme in [17] would no longer be sublinear. We observe that it might be possible to achieve $O_\lambda(\sqrt{N})$ depth by devising a circuit specific to some newer PIR schemes [60, 46] (although not explicitly constructed anywhere). However, this is still significantly larger than our target depth.

1.1 A Detailed Overview of our Contributions

A new single server PIR scheme (with constant update time). Our starting point is to construct a new single server client-preprocessing scheme (with linear offline communication following [60, 46], but constant update time), inspired by the two-server-PIR scheme introduced in [35]. The scheme in [35] uses a different technique for its preprocessing phase than previous approaches [46, 37, 18]: instead of sampling independent subsets of $[N]$, it splits the database in Q chunks and permutes each chunk $\mathsf{db}_1, \ldots, \mathsf{db}_Q$ using permutations $p_1, \ldots, p_Q$. Then, it computes hints $h_1, \ldots, h_{N/Q}$, where each $h_j = \bigoplus_{i \in Q} \mathsf{db}[p_i(j)]$ (i.e., XOR the corresponding database entries). Notice that all the hints are now *dependent* (if an element appears in one hint it certainly does not appear in another). Server 0 performs this computation for the client offline. Then online, the client can manipulate the permutations to generate both a query to Server 1 (which will allow it to retrieve its element of interest using h_j), and a 'refresh' query to Server 0 (which allows it to edit the permutations in a way that they look uniform to Server 1 for the next query).

Our scheme is different from [35] in the following aspects:

- **Refined security requirement for permutations:** first, we propose a scheme that does not require pseudorandom permutations secure for adversaries given any number of queries. We observe it is enough to use a pseudorandom permutation that is secure only against adversaries given $q = o(N)$ queries to the pseudorandom permutation. This allows us to substitute the Fisher-Yates permutation used by [35] with a Thorp q-Shuffle [55], a shuffle whose security only guarantees that it is indistinguishable from a permutation for adversaries given at most q queries. The main benefit of using the Thorp q-Shuffle is that the depth of the computation for evaluating it is actually $O(\lambda)$ for $q = o(N)$ [45], and also we can update the hint with constant time (with also sublinear client-side storage).

- **Eliminating the two-server requirement:** after substituting the permutations for Thorp q-Shuffles, we also eliminate the requirement for two servers by first, having the client download the database offline and compute the hints locally, and second, modifying the query stage carefully so it no longer relies on a refresh operation from Server 0 for the scheme's security. The former technique is known [60], but the edits to the query phase require scheme-specific tailoring.

After all these changes, we have a single-server PIR that has (1) sublinear amortized query time; (2) linear offline bandwidth; (3) low-depth preprocessing computation; (4) sublinear client storage; (5) constant update time.

Notice that this scheme already brings benefits with respect to prior works in the same model, as prior works cannot efficiently support updates natively, and require $O_\lambda(\sqrt{N})$ time per update, while our scheme requires only $O_\lambda(1)$ update time.

Improved Thorp Shuffle Bound. We further optimize our scheme by reducing the depth of computation required to achieve a shuffle with our desired security through an improvement in the Thorp Shuffle's mixing time analysis. While the bound shown by [45] already provides depth $O(\lambda)$, the constants are still undesirable and so we set out to optimize them.

Specifically, in [43], with $\sim 4r \log(N)$ rounds of Thorp shuffle, they show that any (adaptive) q-query adversary to an indistinguishability experiment has an advantage of at most $\frac{2q}{r+1} \left(\frac{4qn}{N}\right)^r$ of distinguishing between the Thorp Shuffle and a real permutation. In our new bound, we prove that with only $2(r+1)\log(N)$ rounds, the adversary has an advantage of at most $\frac{2q}{r+1}\left(\frac{2qn}{N}\right)^r$. Practically, for the same advantage, our bound reduces the depth of the permutation's computation by about 2.5x.

Our proof technique refines the Markov chain coupling argument shown in [45]. Specifically, the technique (both in our work and [45]) is to define a coupling between a Thorp Shuffle process and a (carefully picked) uniform process, and show that after T Markov chain updates, the probability that the chains are not coupled is very small. If the chains are coupled, then the Thorp Shuffle is by definition indistinguishable from a uniform permutation. Although we use the same proof setup, we refine the technique by uncovering additional conditions that would result in a coupling, and then calculating the new probability that the chains are coupled after T steps.

Evaluating the Thorp Shuffle under FHE. Given our new scheme and the optimized Thorp q-Shuffle, we now look into eliminating the linear offline bandwidth by having the server compute the Thorp Shuffle obliviously under FHE. Although a circuit representing the Thorp Shuffle has only $O(\lambda)$ depth, there are still two important considerations we need to make when trying to evaluate the Thorp Shuffle homomorphically using FHE and sublinear bandwidth:

1. **How to generate random bits efficiently?** A generic PRG can be very slow when evaluated using FHE. For example, when evaluating AES using FHE, it takes ∼86 seconds to generate 128 bits [58], which is almost a second per bit. This would mean that a Thorp Shuffle that needs to sample 2^{20} bits would take more than a week of computation to simply generating these bits. We also cannot directly encrypt and send the bits since the number of bits required is $\Omega_\lambda(N)$.

2. **How to perform swaps efficiently?** For the best amortized efficiency, we focus on FHE schemes with the SIMD (same instruction multiple data) feature, over finite field (the BFV [13, 23], BGV [12] schemes). For such schemes, a single ciphertext encrypts D plaintext elements at the same time and every operation over the ciphertext applies to all these D encrypted elements. However, for Thorp shuffle, in each round, we need to move element $i < N/2$ to position $2i$ or $2i+1$. This is very challenging for BFV/BGV as each plaintext inside the same ciphertext needs to be moved to a very different position, potentially even different ciphertexts.

With these two points in mind, we propose the following solutions, which may be of independent interest.

A practical FHE-friendly PRG. As a building block towards our homomorphic oblivious permutation, we build a PRG based on the learning with rounding assumption, which is very FHE-friendly. Concretely, its runtime is only about 0.4 ms per bit, compared to 672 ms per bit when homomorphically evaluating a general practical PRG like AES [58]. As mentioned, we focus on the BFV [11, 23] and BGV [12] homomorphic encryption scheme, which render the best amortized efficiency. As suggested in [2, Appendix A.1], BGV/BFV can be used interchangeably. Thus, later in the paper, we use BFV only for simplicity.

Observe that BFV scheme works best (in terms of efficiency) over $\mathbb{Z}_t$ where t is a small prime of size 15-30 bits (we explain why in more detail in Section 4.1). Therefore, to fit this constraint, we make our PRG work over $\mathbb{Z}_t$ as well. A natural idea is thus to build it from a lattice assumption, as lattice assumptions can work over prime fields of such size. Our PRG is thus based on the Learning with Rounding (LWR) problem. At a high level, we compute $R(As)$ where $A \leftarrow_\$ \mathbb{Z}_t^{m\times n}, s \leftarrow_\$ \mathbb{Z}_t^n$

and $R(x) : \mathbb{Z}_t^m \to \mathbb{Z}_2$ checks if $x \in [0, \lceil t/2 \rceil]$ return 0 and otherwise return 1. This gives us a PRG that outputs 0 with probability $\frac{\lceil t/2 \rceil}{t} \approx 1/2$. However, this is not enough. Essentially, we need a PRG that outputs 0 with probability $1/2 + \mathsf{negl}(\lambda)$. To do this, we sample k independent A_i's for $i \in [k]$ and compute $R(A_i s)$, for some k such that $t^{-k} = \mathsf{negl}(n)$. Then, if $A_i s$ is 0 (for each single $\mathbb{Z}_t$ element among the m $\mathbb{Z}_t$ elements), we check $A_{i+1}s$ for $i \in [k-1]$. If all of them are 0, we return 0, but otherwise, we return $R(A_i s)$ where i is the smallest value such that $A_i s \neq 0$. In this case, we obtain a PRG based on LWR.

To make it even more FHE-friendly, we design an alternative $R(\cdot)$ function, that instead maps x to 0 iff $x^{(t-1)/2} = -1$, which can be computed in only $\log(t) - 1$ multiplications. Essentially, this function maps half of $x \in \mathbb{Z}_t^*$ to either 0 and the other half to 1. Furthermore, checking whether $A_i s = 0$ is also easy: $x^{t-1} = 0$ iff $x = 0$. Putting all these together, our PRG only takes $k \cdot \log(t)$ multiplications, and the security relies on a variant of LWR with a special rounding function $R(\cdot)$ described above.

Perfoming swaps with FHE. Now we address the second issue: swapping under FHE. As mentioned, performing the Thorp Shuffle with BFV naively is not very efficient. However, by [20, Lemma 1], n rounds of the Thorp Shuffle are equivalent to n rounds of a butterfly network, and the update rule of the butterfly network is much more BFV-friendly.

In particular, at round $k \in [\log(N)]$, position i is swapped with position $i + 2^{k-1}$. This is easily doable by BFV since all these pairs get swapped have the same interval in terms of positions. Therefore, we implement that instead.

Another difficulty that comes up is that even with our tightened Thorp Shuffle bound, we still require hundreds of levels of multiplication under FHE. If we directly use regular BFV without bootstrapping, the parameters to support this many levels are too large.

In addition, the naive alternative, regular BFV bootstrapping is very slow (hundreds of milliseconds per element). We circumvent both by leveraging a recent advancement in relaxed BFV bootstrapping (which requires only a couple of milliseconds per element bootstrapped) [40]. Although the relaxed bootstrapping only guarantees correctness when each plaintext element is in a predetermined fixed subset of the plaintext space, it works in our use case since we only need encrypted bits.

Putting everything together. Finally, we can combine our PIR scheme with the homomorphic Thorp Shuffle to achieve a client-preprocessing scheme with sublinear bandwidth and low-depth, highly parallelizable server computation in the offline phase. We discuss its performance in Section 5.

1.1.1 Why not sorting networks?

A simple solution to performing a shuffle given a generic FHE scheme and a generic PRG $g : \{0,1\}^\lambda \to \{0,1\}^*$ is for the client to generate a PRG seed s, encrypt it under FHE, and send it to the server. It also sends its public key. The server then homomorphically evaluates $g(s)$ and assigns a random value (encrypted under FHE) to each database entry. It then uses a sorting network to sort the database elements according to the random values, where each gate in the network is encrypted under FHE. We already went into the difficulty of how to evaluate a PRG under FHE (and our proposed solution). However, it is still unclear why we go through the trouble of using a q-Thorp Shuffle when there exists low-depth sorting networks that can be used to solve the problem. The sorting network solution poses two additional *major* drawbacks.

1. Although there exist known sorting networks with optimal $O(N \log N)$ size and $O(\log N)$ depth [1, 51, 15], these sorting networks are notoriously complex and incur huge constant factors in the depth (the best known sorting network with $O(\log N)$ depth has depth about $1800 \log N$). Sorting networks with $O(N \log^2 N)$ gates and $O(\log^2 N)$ depth [6, 49, 54] perform better empirically, but still are asymptotically suboptimal, and furthermore suffer from the drawback below.

2. The ciphertexts randomly assigned by the server to each block must encrypt plaintext values of at least λ bits to ensure the sorting network generates a computationally random permutation (to avoid being assigned to the same random element except with 2^{λ} probability). This restriction imposes an additional λ multiplicative factor to the depth of the FHE computation of a permutation on any sorting network, since comparing between these two elements homomorphically requires a circuit of depth and size $O(\lambda)$. Thus combining with item (1), the total depth is $\Omega(\lambda \log(N))$.[2]

We also compare our solution with this naive solution in Table 1 in terms of both depth and estimated runtime.

1.2 Related Work

1.2.1 Oblivious Shuffles.

Oblivious shuffles have been studied under different definitions and contexts. Informally, we define a shuffle to be *oblivious* if the access pattern of the shuffling does not reveal any information about the shuffle itself. This definition is different than what is seen in [30, 44], whose schemes does not satisfy the definition above. Oblivious permutations (as defined above) have been studied to a large amount in the client server model, where the client wants to shuffle some N encrypted blocks stored at the server according to some pseudorandom permutation σ which should be hidden from the server. Works in this model [47, 50, 3] achieve near-optimal asymptotics with very little client storage usage, however, all known works require the client and server to communicate $O(N)$ information to perform these shuffles. In our work, we want to avoid the $O(N)$ communication, although for scenarios with very large database elements, it might be worth looking into performing an oblivious permutation with linear bandwidth that is perhaps independent of element size.

1.2.2 Permutation Networks.

Permutation networks are networks supposed to shuffle its inputs. There are two best known networks: Butterfly networks (which can equivalently be seen as the Thorp Shuffle [55] or a maximally unbalanced Feistel network), and Benes networks [8]. Known mixing times for the Thorp Shuffle are tighter than those for Benes networks [25], so we focus on the Thorp Shuffle in this work. Another permutation network, proposed by Waksman [56], achieves optimal size and depth, but it requires random access on a permutation matrix of size $N \times N$ which is costly when trying to evaluate the circuit under FHE.

[2] Notice that we cannot simply assign random output to each comparison in the sorting network. The output of the sorting network for this case would not necessarily be equivalent to sampling a permutation uniformly (although it certainly is true that every possible permutation can be output in this method, the distribution of each is not uniform). It is an open question whether there exists a low-depth sorting network that can be used with random comparators to output something indistinguishable from a uniformly sampled permutation.

1.2.3 Private Information Retrieval.

PIR has been studied in a series of different contexts and models. Other than the client preprocessing model studied in this work, another popular model is the server preprocessing model. In the server preprocessing model, the server runs one very expensive preprocessing and stores additional bits at the server which it can then use to respond to queries by *any* client in sublinear time. This has been studied in a series of works using non-standard assumptions or trusted setups [31, 10]. Recently Lin et al. [39] pushed a breakthrough on the server preprocessing model, achieving near-optimality under standard assumptions. Although this model is very appealing, the costs in preprocessing time and server storage are still prohibitive [48], however it is still very recent work and there are likely many more practical improvements to be made. PIR has also been extensively studied with the simplifying assumption of requiring two non-colluding servers. Using the scheme by [27, 9, 28] we can run extremely efficient non-preprocessing PIR at rates very close to optimal, with the caveat of the non-colluding servers assumption.

Other works have studied single server PIR in the context of preprocessing with a smaller client hint [29, 21]. These works still incur linear server time during the query phase, however, all expensive operations are performed offline and the online phase can be viewed as a single scan through the database. Another emerging interesting line of work within PIR is the line of work on Authenticated PIR [16, 22, 57]. In this line of work, the client can be guaranteed that the server followed the protocol honestly, a guarantee not inherent in standard PIR (in the standard PIR definition, although privacy holds against malicious servers, correctness only holds for a server behaving honestly).

2 Preliminaries

Notation. Let $\mathsf{Bern}(p)$ for $0 \leq p \leq 1$ denote the Bernoulli distribution with probability p: i.e., $\mathbf{P}_{x \leftarrow \mathsf{Bern}(p)}(x = 0) = p$ and $\mathbf{P}_{x \leftarrow \mathsf{Bern}(p)}(x = 1) = 1 - p$. $[x] := \{1, \dots, x\}$ for $x \in \mathbb{Z}^+$.

2.1 Hard Problems

Definition 2.1 (Decisional learning with rounding problem)**.** Let n, q, p be parameters dependent on λ, and $R : \mathbb{Z}_q \to \mathbb{Z}_p$ be a function. The learning with rounding (LWR) problem $\mathsf{LWR}_{n,q,p,R}$ states the following: for any $m = \mathsf{poly}(\lambda)$, distinguish $(A, R(\vec{s}A))$ and $(A, R(\vec{b}))$ (with noticeable advantage), where $A \leftarrow_\$ \mathbb{Z}_q^{n \times m}, \vec{s} \leftarrow_\$ \mathbb{Z}_q^n$ and $\vec{b} \leftarrow_\$ \mathbb{Z}_q^m$.

Let $\lfloor\cdot\rceil(x) := \lfloor p \cdot x/q \rceil$ (for the p, q above), then $\mathsf{LWR}_{\cdot,\cdot,\cdot,\lfloor\cdot\rceil}$ is the standard decisional learning with rounding problem introduced in [5].

2.2 Pseudorandom Generator

Definition 2.2. A (t, m, p)-PRG is a deterministic and polynomial-time computable function $f : \mathbb{Z}_t^n \to \{0,1\}^m$, such that for any PPT adversary $\mathcal{A}$, $|\Pr[\mathcal{A}(f(s)) = 1] - \Pr[\mathcal{A}(R)]| \leq \mathsf{negl}(n)$, where $s \leftarrow_\$ \mathbb{Z}_t^n, R \leftarrow_\$ \mathsf{Bern}(p)^m$, and $m > n\lceil\log(t)\rceil$.

A standard PRG is simply a $(t, m, 1/2)$-PRG, and we denote it as (t,m)-PRG and ignore 1/2 for simplicity.

2.3 FHE

Fully Homomorphic Encryption (FHE), introduced by Rivest et al. [53] and first constructed by Gentry [26], enables evaluation of a circuit on encrypted data, such that the result is the encryption of the corresponding output.

BFV FHE scheme. We use the Brakerski/Fan-Vercauteran (BFV) homomorphic encryption scheme [11, 23] in all constructions.

BFV scheme consists of the following PPT algorithms: $\mathsf{GenParams}(1^\lambda), \mathsf{KeyGen}(\mathsf{pp}_{\mathsf{BFV}})$,$\mathsf{Enc}(\mathsf{pp}_{\mathsf{BFV}}$,$\mathsf{pk}, m), \mathsf{Dec}(\mathsf{pp}_{\mathsf{BFV}}, \mathsf{sk}, c)$ as normal PKE schemes. BFV is unconditionally correct and sound. Under the Ring-LWE hardness assumption, it also fulfills the standard definitions of semantic security (IND-CPA) for FHE schemes.

Given a polynomial from the cyclotomic ring $R_t = \mathbb{Z}_t[X]/(X^D + 1)$ (where D is a power-of-two, $t \equiv 1 \mod 2D$), the BFV scheme encrypts it into a ciphertext consisting of two polynomials, each of in a larger cyclotomic ring $R_Q = \mathbb{Z}_Q[X]/(X^D + 1)$ for some $Q > t$. Here, t, Q, and D are called the plaintext modulus, the ciphertext modulus, and the ring dimension, respectively.

Plaintext encoding. In practice, instead of having a polynomial in $\mathcal{R}_t = \mathbb{Z}_t[X]/(X^D+1)$ directly as input, applications usually hold a vector of messages $\vec{m} = (m_1, \ldots, m_D) \in \mathbb{Z}_t^D$. Thus, to encrypt such input messages, BFV first encodes the messages into a polynomial in $\mathcal{R}_t$ (via Inverse Number Theoretic Transform). We say that a BFV ciphertext has D *slots*, each of which is a $\mathbb{Z}_t$ element.

For simplicity, we assume $\mathsf{BFV.Enc}$ takes a vector of form $\mathbb{Z}_t^D$ as an input, and $\mathsf{BFV.Dec}$ outputs a vector of form $\mathbb{Z}_t^D$, and will handle encode and decode implicitly.

Operations. BFV supports the following operations.

- (Additions) For any two BFV ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$, and $\mathsf{ct} \leftarrow \mathsf{ct}_1 {+} \mathsf{ct}_2$, it holds that $\mathsf{BFV.Dec}(\mathsf{ct}) = \mathsf{BFV.Dec}(\mathsf{ct}_1) + \mathsf{BFV.Dec}(\mathsf{ct}_2)$ (element-wise).
- (Multiplication) For any two BFV ciphertexts $\mathsf{ct}_1, \mathsf{ct}_2$, and $\mathsf{ct} \leftarrow \mathsf{ct}_1 \times \mathsf{ct}_2$, it holds that $\mathsf{BFV.Dec}(\mathsf{ct}) = \mathsf{BFV.Dec}(\mathsf{ct}_1) \times \mathsf{BFV.Dec}(\mathsf{ct}_2)$ (element-wise).
- (Rotation) For any BFV ciphertexts ct, and $\mathsf{ct}' \leftarrow \mathsf{BFV.Rotate}(\mathsf{ct}, k)$ for some $k \in [D]$, let $\mathsf{BFV.Dec}(\mathsf{sk}, \mathsf{ct})[i] = \mathsf{BFV.Dec}(\mathsf{sk}, \mathsf{ct}')[i + k \mod D]$.

2.4 Private Information Retrieval

We first formally define correctness and privacy for PIR.

Definition 2.3 (Interaction)**.** A PIR interaction between client and server for PIR scheme = ($\mathsf{Preprocess}$, Query, Answer, $\mathsf{Reconstruct}$) is as follows:

- Client requests the database and runs $\mathsf{Preprocess}$ locally.
- For each query x_t, for $t \in [T]$, client runs $a_1, \ldots$,a_q = $\mathsf{Query}(\mathsf{st}$,$x_t)$ and sends $a_1, \ldots$,a_Q to the server.
- Server returns $\{db_i(a_i)\}_{i \in [Q]}$ to the client.
- Client outputs $\mathsf{Reconstruct}(\mathsf{st}, \{db_i(a_i)\}_{i \in [Q]}, x_t)$.

After T queries, client must re-run the preprocessing to perform the next query. In addition, st is the client's state with which it can access variables across algorithms.

Definition 2.4 (PIR correctness)**.** A PIR scheme (Server,Client) is correct if, for any polynomial-sized sequence of queries $x_1, \ldots, x_Q$, the honest interaction of Client with Server (Definition 2.3) that stores a polynomial-sized database $\mathsf{DB} \in \{0,1\}^N$, returns $\mathsf{DB}[x_1], \ldots, \mathsf{DB}[x_Q]$ with probability $1 - \mathsf{negl}(\lambda)$.

This definition of interaction above assumes that the client runs the preprocessing. We slightly abuse the definition and allow the server to run the preprocessing for one of our theorems. In this case, the client runs an additional algorithm Init whose output it sends to the server, and saves the hints output from the server's preprocessing.

We now define correctness and privacy according to the interactions above.

Definition 2.5 (PIR privacy)**.** A PIR scheme (Server,Client) is **private** if there exists a PPT simulator Sim, such that no PPT adversary $\mathcal{A}$ can distinguish the following experiments with non-negligible probability:

- **Expt**$_0$: Client interacts with $\mathcal{A}$ who acts as Server. At every step t, $\mathcal{A}$ chooses the query index x_t, and Client is invoked with input x_t as its query.
- **Expt**$_1$: Sim interacts with $\mathcal{A}$ who acts as Server. At every step t, $\mathcal{A}$ chooses the query index x_t, and Sim is invoked with no knowledge of x_t.

In the above definition our adversary $\mathcal{A}$ can deviate arbitrarily from the protocol.

2.5 Thorp Shuffle

The Thorp Shuffle is a shuffling algorithm introduced in [55]. It works as follows. We start with the ordered set $[1, \ldots, N]$ for some even N. Then, for every pair of cards $i, i + N/2$, we flip a coin that decides which one will be placed in position $2i$, and which one will be placed in position $2i + 1$ on the next round. This is then repeated for multiple rounds. As the number of rounds t approaches infinity, the output of the Thorp Shuffle approaches the distribution of a uniformly sampled permutation [43]. We describe the algorithm below in Algorithm 1.

Algorithm 1 Thorp Shuffle. Parameters t,$N \in \mathbb{N}$, N even, $n = \lceil \log_2 N \rceil$.

```
procedure THORP SHUFFLE($A_1, \ldots A_N \in [N]$)
    for $t$ steps do
        Let $F_i = A_i$ for $i \in \{1, \ldots, N/2\}$
        Let $S_i = A_{i+N/2}$ for $i \in \{1, \ldots, N/2\}$
        for $j$ in $[1, \ldots, N/2]$ do
            Sample bit $b$ uniformly at random
            if b then
                Let $A_{2j-1} = S_j$, $A_{2j} = F_j$
            else
                Let $A_{2j-1} = F_j$, $A_{2j} = S_j$
    return $A_1, \ldots, A_N$
```

Algorithm 1 describes a 'perfect' version of the Thorp Shuffle, using perfect randomness. We will denote this perfect version of the Thorp Shuffle Th_t, and abuse notation slightly to also allow point queries and inverse queries to the this perfect Thorp Shuffle, which tell us where a specific element landed at the end or where it began. We define these operations in the context of a Thorp Shuffle using *computational randomness* below (using a PRG to seed the bits utilized in each step) below.

Definition 2.6. A Thorp Shuffle Th is a tuple of three algorithms:

- $\mathsf{Gen}(\lambda, N \in \mathbb{N}, q \in [N]) \rightarrow s \in \{0{,}1\}^{O(\lambda)}$: This algorithm takes in a security parameter and a set size N and outputs a PRG seed s to be used to both generate the Thorp Shuffle secure for q queries and also includes the number of rounds t to run for.

- $\mathsf{Eval}(s{,}x \in [N])$: Takes in a seed s and an index $x \in N$ and outputs its Thorp Shuffle evaluation y in $O(\lambda)$ steps using s and a PRG to generate the bits necessary.

- $\mathsf{Inv}(s{,}y \in [N])$: Takes in an index $x \in N$ and outputs its inverse Thorp Shuffle evaluation x in $O(\lambda)$ steps using s (outputs x such that $\mathsf{Th}.\mathsf{Eval}(s{,}x) = y$.

Concretely, the seed s is just a PRG seed of size λ and a description of how many rounds to run the protocol for, which is also a function of λ. We then use the seed s to generate the bits b in each step.

For efficiency reasons, we would like to minimize t, the number of rounds of Thorp Shuffle to run, *especially* if it will be run under FHE. Next, we will look at how to bound the Thorp Shuffle's distance from a true permutation. In this bound, we use $\mathsf{Th}_t(\cdot)$ and $\mathsf{Th}_t^{-1}(\cdot)$ to point queries to the perfect Thorp Shuffle and its inverse for a shuffle run for t steps (i.e., Thorp shuffle using true randomness instead of a PRG). For any $N, \lambda \in \mathbb{N}$, $q \in [N] = o(N)$, $t \geq 1$, [45] prove that for any adaptive q-query CCA adversary $\mathcal{A}$, it follows that:

$$\left|\mathbf{P}\left(\mathcal{A}^{\mathsf{Th}_t(\cdot),\mathsf{Th}_t^{-1}(\cdot)}(\lambda, q) = 1\right) - \mathbf{P}\left(\mathcal{A}^{\pi(\cdot),\pi^{-1}(\cdot)}(\lambda, q) = 1\right)\right| \leq \frac{2q(4n+t)}{4n-4}\left(\frac{4qn}{N}\right)^{t/(4(n-2))}.$$

where $n = \log(N)$ and π is a random permutation. Note that this is the best-known concrete bound on the number of rounds for $q \approx O(\sqrt{N})$. The bound says that approximately every $4n$ rounds of Thorp allows us to reduce the adversary's advantage by a factor of $(4qn/N)$; given $q \approx \sqrt{N}$, this fraction is approximately $1/\sqrt{N}$. Looking ahead, in Section 4 we show how to improve this bound to cut down the number of rounds required to achieve computational indistinguishability and greatly reduce the computation depth needed for the Thorp Shuffle.

3 A Permutation-Based Single Server PIR Scheme

Our starting point is the 2-server client-preprocessing PIR scheme from [35]. For completeness, we include the original scheme in Appendix B (see a high-level summary in Section 1.1). This scheme is relevant to us because in the offline phase, Server 0 samples Q permutations of N/Q elements (seeded by a client-sent seed), computes parities based on these permutations and returns them to the client. The client then uses a second server (Server 1) to perform queries, since sending the queries to the same server that saw its permutations would breach privacy (client still uses Server 0 for refreshing its hints).

Here, we propose a series of changes to the scheme in [35] to construct a single server scheme using the idea of sampling Q permutations of N/Q elements. First, we construct our scheme in the framework of client preprocessing single server PIR that *allows linear bandwidth offline.* In this model, previously explored in [60, 46], the server streams the database to the client in the offline phase, which computes (with sublinear storage) hints that can then be used at query time to perform queries that take $o(N)$ time. Looking ahead, this scheme will serve as a stepping stone to later construct a scheme that does not need linear bandwidth offline, by constructing a primitive that allows the shuffle to be sampled at the server obliviously.

3.1 Building a Single Server PIR Scheme.

Algorithm 2 defines our scheme variant with linear offline bandwidth. Our scheme is inspired by [35], but differs in several important aspects. We will first give a high level intuition and present our scheme. Later we delve into the major differences between our scheme and [35].

3.1.1 Intuition.

At a high level, our scheme works as follows. Initially, the server streams the database to the client one element at a time. For each database element it sees, it finds the appropriate hint (i.e., which chunk of data the element is to-be XOR-ed with) this element belongs to (it can do this using the Thorp Shuffle permutation in $O(\lambda)$ time) and xors this element into the appropriate hint. It repeats this for every element in the database.

For all $j \in [N/Q]$, each hint h_j represents the XOR of $\mathsf{db}_i[\tau_i(j)]$ for all $i \in Q$ (where db_i represents the i-th chunk of db which is divided into Q equally sized chunks). At the end of the preprocessing phase, the client stores the permutations and hints permanently.

Online, for the first query to $x = (q,k) \in ([Q] \times [N/Q])$, the client first computes $j = \tau_q^{-1}(k)$, and uses this j to find all the other database elements it needs to recover $\mathsf{DB}[x]$ from the hint. In more detail, compute $o_i \leftarrow \tau_i(j)$ for $i \neq q$. Then, the client can compute $\mathsf{DB}[x] = h \oplus_{i \neq q} \mathsf{db}_i[\tau_i(j)]$ where h is the hint when getting $\mathsf{db}_i[o_i]$ back by sending $o_i = \tau_i(j)$. These o_i's are indistinguishable from uniformly at random from $[K]$ by the property of the underlying Thorp shuffle, and thus leaks nothing about (q,k). Note that, however, it should also send o_q, as otherwise, the server learns that $x = (q, \cdot)$. Thus, the client samples o_q uniformly at random from $[K]$ and sends it. After querying, the client simply stores $\mathsf{db}_i[o_i]$ in USED_i (some dictionary storing $\mathsf{USED}_i[o_i] = \mathsf{db}_i[o_i]$) for all $i \in [Q]$.

Now, for the following queries, the client first obtains o_i as in the first query. However, note that now the server might have seen o_i already, which may leak some information (e.g., for the same $x = (q,k)$ as the first query, o_q can be different but everything else remains the same). However, not that for the o_i that is already seen, the client know what $\mathsf{db}_i[o_i]$ is as it is stored in USED_i. Thus, instead of directly sampling o_i, resample o_i uniformly at random from $[K] \setminus \mathsf{USED}_i$. Send the o_i's after resampling. For the ones that are not resampled, they are indistinguishable from $[K] \setminus \mathsf{USED}_i$ by the property of the Thorp shuffle. Again, every query stores $\mathsf{db}_i[o_i]$ in USED_i.

With all these intuitions, we show our construction in Algorithm 2, and can show the following theorem.

Theorem 3.1. *The PIR scheme defined in Algorithm 2 is correct (Definition 2.4) and private (Definition 2.5) for $T = o(N)$ queries, and runs with the following complexities:*

Algorithm 2 Single Server PIR scheme

```
procedure PREPROCESS($\mathsf{DB} \in \{0,1\}^N, \lambda$, T)
    Let $K := N/Q$. $\mathsf{DB} = \mathsf{db}_1, \ldots, \mathsf{db}_Q$ where each $\mathsf{db}_i \in \{0,1\}^K$.
    Let $s_1, \ldots, s_Q$ be seeds generated with $\mathsf{Th.Gen}(\lambda, N, T)$.
▷ The number of rounds $t$ for the Thorp shuffle is chosen according to Theorem 3.2, such that the adversary has advantage $\leq 2^{-\lambda}$.
    Initialize $h_1, \ldots, h_k = 0$.
    for $t$ in $[1, \ldots, Q]$ do
        for $j$ in $[1, \ldots, N/Q]$ do
            Let $k = \mathsf{Th.Inv}(s_i, j)$.
            Let $h_k = h_k \oplus \mathsf{db}_i[j]$.
    Initialize empty dictionaries $\mathsf{Used}_i$ for $i \in [Q]$.
    return $\mathsf{st} = (H = (h_1, \ldots, h_{N/Q}), (\mathsf{Used}_i)_{i \in [Q]}, (s_1, \ldots, s_Q))$.
procedure QUERY($\mathsf{st}, x = (q,k) \in [Q] \times [K]$)
    Let $y = \mathsf{Th}_t.\mathsf{Eval}(s_q, k)$ and add $y$ to $\mathsf{st}$.
    Let $D_i$ denote the set of all the keys of $\mathsf{st.Used}_i$ for $i \in [Q]$.
    Let $o_i \leftarrow \mathsf{Th.Eval}(s_i, y)$ for $i \in [Q] \setminus \{q\}$ and $o_q \leftarrow_\$ [K] \setminus D_q$.
    For each $i \in [Q]$: if $o_i \in D_i$ then $o_i \leftarrow_\$ [K] \setminus D_i$.
    return $o_1, \ldots, o_q$.
procedure ANSWER($(\mathsf{DB} = (\mathsf{db}_1, \ldots, \mathsf{db}_Q), (o_1, \ldots, o_Q))$)
    return $(\mathsf{db}_1[o_1], \ldots, \mathsf{db}_Q[o_Q])$.
procedure RECONSTRUCT($\mathsf{st}, \{\mathsf{db}_i[o_i]\}_{i \in [Q]}, pp, x = (q,k)$)
    Let $\mathsf{st.Used}_i[o_i] \leftarrow \mathsf{db}_i[o_i]$ for all $i \in [Q]$.
    $a_i \leftarrow \mathsf{Th.Eval}(s_i, \mathsf{st}.y)$ for $i \in [Q] \setminus \{q\}$.
    Let $\mathsf{DB}[x] = \mathsf{db}_q[k] = \left(\bigoplus_{i \in [Q], i \neq q} \mathsf{st.Used}_i[a_i]\right) \oplus \mathsf{st}.h_y$.
    return $\mathsf{DB}[x]$.
```

- $O_\lambda(N)$ *preprocessing time.*

- $O(N)$ *offline bandwidth.*

- $O_\lambda(Q)$ *online client time.*

- $O(Q \log N)$ *online bandwidth*

- $O(Q)$ *online server time.*

- $O(N/Q + T \cdot Q)$ *client storage.*

- $O_\lambda(1)$ *update time*

We prove this theorem in Appendix A. The proof requires new techniques to adapt to the Thorp Shuffle instantiation only secure for q queries against adaptive adversaries.

Note that after T queries we can just re-run the scheme from scratch to achieve a scheme with the same amortized complexities that runs for unlimited queries.

Parameter setting. For our scheme to be sublinear across all complexities, we must pick Q,T such that $Q \cdot T = o(N)$. For example, for $Q = T = N^{1/3}$, we get a single server PIR scheme with $O(N^{2/3})$ amortized server time, $O(N^{2/3})$ offline and total bandwidth, $O(N^{1/3})$ online bandwidth and query time, and $O(N^{2/3})$ client storage. By using the Thorp Shuffle permutation, we actually circumvent the linear storage constraint present in [35] due to using the Fisher-Yates construction [24]. See Appendix A for why the update time is $O_\lambda(1)$.

3.1.2 Comparison to [35]

The main differences between our scheme and the scheme presented in [35] are as follows:

1. Our permutations are now *q-shuffles* sampled using the Thorp Shuffle [55, 45]. This means, on one hand, that we can only show at most q points of each permutation sampled to the adversary before our shuffle's security no longer holds. However, this also means that we greatly reduce the depth of the computation, since Fisher-Yates is a sequential algorithm with $O(N)$ depth.

2. The server now streams the database to the client upon request. Then, the client computes the hint itself.

3. We eliminate completely the refresh operation. This requires changing the entire query step to work differently. Previous works have ported two-server schemes to single server schemes before by storing backup hints during the offline phase to later replace used hints [17, 36, 59]. This approach does not directly work in this case, since the hint at the client is not comprised of independent subsets of the database. Instead, we take a different approach and store elements seen, performing dummy queries when repeated elements are queried. This requires quite a few modifications to the online phase to ensure that the distribution the server sees is always uniform and independent of the queries.

4. Our scheme needs to be re-run from scratch every T queries. This is also true for previous single server client preprocessing scheme [17, 36, 59] and a consequence of not being able to refresh our state with the help of a second server as is the case for two-server schemes.

Condition (1) causes us to incur an extra λ factor in the preprocessing when compared to [35]. However, it also decreases the depth of the preprocessing computation from $O(N)$ to $O_\lambda(1)$. Conditions (2) and (3) are what allow us to eliminate completely the role of Server 0 in the original scheme and are what allow us to transform it into a single server scheme. Condition (4) is a consequence of eliminating Server 0. Since we can no longer receive fresh random elements to replace the ones we used, we can only support a limited number of queries before having to re-run the preprocessing.

3.2 An Improved Thorp Shuffle Bound

We know that the output of the Thorp Shuffle converges to a uniformly sampled permutation as the number of rounds we shuffle for approaches infinity [43]. More than that, [45] have analyzed the Thorp Shuffle and specifically looked at exactly how well a set of $N = 2^n$ cards are shuffled after t rounds of the the Thorp Shuffle. We define "how well" as the advantage an adversary would have in distinguishing the shuffled elements from a truly random permutation given q queries to it. Specifically, [45] bounds the total variational distance between the q queries of the Thorp Shuffle and q queries from a uniform permutation over N elements. We improve on the bound on [45] that was aforementioned with the following theorem.

Theorem 3.2. *Let $N = 2^n$ and $q \in \{1, \ldots, N\}$, $\{\mathsf{Th} : t \geq 1\}$ be the Thorp Shuffle of $[N]$ after t rounds. Then, for adaptive, q-query PPT adversary $\mathcal{A}$:*

$$\left|\mathbf{P}\left(\mathcal{A}^{\mathsf{Th}_t(\cdot),\mathsf{Th}_t^{-1}(\cdot)}(\lambda, q) = 1\right) - \mathbf{P}\left(\mathcal{A}^{\pi(\cdot),\pi^{-1}(\cdot)}(\lambda, q) = 1\right)\right| \leq \frac{2q(n+t)}{n+1}\left(\frac{2qn}{N}\right)^{t/(2(n+1))}.$$

where π is a uniformly sampled permutation.

Recall Th_t denotes the perfect shuffle run for t rounds and we allow for oracle point queries and inverse queries to the shuffle (for a total of q queries adaptively). Notice that compared to the bound achieved before (stated in Section 2.5), our bound is smaller by about a factor of $(2qn/N)^{t/(2n)}$, which allows to use about half as many rounds while maintaining the same security. This directly reduces the depth of the computation for the shuffle.

At a high level, our technique used, inspired by [45], is to define a Markov chain coupling that correctly models the Thorp Shuffle and a uniform distribution, and try to bound how fast these chains couple together.

Now, we provide an overview of the techniques used in our new bound's proof.

Techniques. The proof in Morris et al. and subsequently ours uses a *coupling argument.* We briefly give some intuition about what this is. A Markov chain can be defined by a transition matrix P and initial state x. We denote $P^t(x)$ to mean the resulting state of applying the transition on initial state x for t times, sequentially. Let π be the stationary distribution over the group, where, for our purposes, the group is the group of all permutations of N cards, and the stationary distribution is a uniform sample from this group (a uniform permutation). The relevant measurement that we would like to minimize is the total variational distance between $P^t(x)$ and π, denoted $\Delta\left(P^t(x), \pi\right)$. In specific, if we can show that the distance is 0, then we can say that after t steps our initial state is indistinguishable from uniform. The first thing to verify is that the Thorp Shuffle, starting from any initial distribution, eventually converges to the stationary distribution after enough steps. This was shown in [43].

The technique to bound the total variational distance is is to define two pair processes, meaning two Markov chains with the same transition matrix P, but different starting states (where one gets the desired x and the other some sample from π). This makes it so such that if for any t, the two pair processes hit the same state, then they are the same from then onwards (since they are both updated in the same way).

In [45], they show that for pair processes $\{X_t\}$ and $\{Y_t\}$ such that $X_1 = x$, $Y_1 \xleftarrow{\$} \pi$, both with transition matrix P, it holds that:

$$\Delta\left(P^t(x), \pi\right) \leq \mathbf{P}(X_t \neq Y_t) = \mathbf{P}(T > t),$$

where T is a random variable, $T = \min\{t : X_t = Y_t\}$. The insight here is that since $Y_1 \xleftarrow{\$} \pi$ and P is a probabilistic transition matrix, $Y_t \xleftarrow{\$} \pi$ for any t. Then, if after T steps, $X_T = Y_T$, we can conclude that X_T is indistinguishable from a uniform sample from the group.

For improved readability, we defer the full proof for Theorem 3.2 to Section 6.

4 Homomorphic Thorp Shuffle

We have already motivated why the Thorp Shuffle is the best candidate shuffling algorithm to perform under FHE. However, we have not seen yet how exactly to deal with the problems alluded to in Section 1. Here, we recall the two main problems one faces when attempting to evaluate the Thorp Shuffle under FHE, and how we propose to deal with them.

The first issue comes from the homomorphic PRG evaluation. Evaluating a commonly used PRG like AES homomorphically is very slow. The state-of-the-art homomorphic AES evaluation takes 86 seconds to generate 128 random bits [58]. Thus, even if only 2^{20} random bits are needed, it would take 704512 seconds to generate these bits using this solution. The second issue is that the homomorphic permutation algorithm needs to be oblivious.

The second issue is the a naive realization of Thorp Shuffle is very FHE-unfriendly and can be very costly in terms of runtime (as it needs to map elements inside the same ciphertext to different locaitons and potentially different ciphertexts).

4.1 LWR-based FHE-friendly PRG

We start by addressing the first issue: constructing a scheme that can efficiently *and* obliviously generate pseudorandom bits under FHE.

Recall that operations over small finite fields are more FHE-friendly, since FHE supports only multiplications and additions over $\mathbb{Z}_t$ for some prime t.[3] Thus, with the motivation of building an FHE-friendly PRG, we focus on lattice-based constructions, as they can indeed work over some finite field $\mathbb{Z}_t$. Our goal is to construct a (t, m, p)-PRG with only multiplications and additions.

Construction with $p = r/t$ for $r \in \mathbb{Z}_t$. We start with a simpler requirement: sampling bits with distribution $\mathsf{Bern}(r/t)$ for $r \in \mathbb{Z}_t$. At a high level, the core idea is to rely on the Learning-with-rounding assumption: the seed is simply a vector $s \leftarrow_\$ \mathbb{Z}_t^n$. Then, the sampling is easy: first randomly sample $A \leftarrow_\$ \mathbb{Z}_t^{w \times n}$ and compute $R_r(As)$, where $R_r : \mathbb{Z}_t \to \{0,1\}$ is defined as

$$R_r(x) = \begin{cases} 1 & \text{if } x \in [0, r-1] \\ 0 & \text{o.w.} \end{cases}$$

[3] Technically, BFV also works with t^r where t is a prime and $r > 1$. However, this reduces the amortized efficiency and we do not consider such parametrization.

Algorithm 3 $(t, m, r/t)$-PRG for $r \in \mathbb{Z}_t$

```
procedure $f_A(s)$                                    $\triangleright\ A \leftarrow_\$ \mathbb{Z}_t^{m\times n}$
    Define
        $R_r(x) = \begin{cases} 1 & \text{if } x \in [0, r-1] \\ 0 & \text{o.w.} \end{cases}$
    return $R_r(As)$
```

Here the R_r can be evaluated via a degree-$(t-1)$ polynomial function (interpolated using R_r). Of course, one restriction is that we need $\mathsf{LWR}_{n,q,2,R_r}$ to be hard, which is assumed to hold as long as r is large enough.

Security of $\mathsf{LWR}_{n,q,2,R_r}$. While the rounding function R_r is not a standard rounding function, for a lot of values of r, we can in fact reduce a regular $\mathsf{LWR}_{n,q,q',\lfloor\cdot\rceil}$ instance for some $q' \geq 2$ to our LWR assumption with this special rounding function. We formalize it with the following lemma.

Lemma 4.1. *For any $n, q > 0$, and $r, q' > 0$ such that $\lfloor (r-1) \cdot (q/q') \rceil = 0$ and $\lfloor r \cdot (q/q') \rceil = 1$, it hold that $\mathsf{LWR}_{n,q,q',\lfloor\cdot\rceil} \leq \mathsf{LWR}_{n,q,2,R_r}$.*

Proof. To prove this lemma, given an adversary $\mathcal{A}$ that breaks $\mathsf{LWR}_{n,q,2,R_r}$, we construct the following adversary that breaks $\mathsf{LWR}_{n,q,q',\lfloor\cdot\rceil}$.

Given an $\mathsf{LWR}_{n,q,q',\lfloor\cdot\rceil}$ sample $(A, \vec{b}) \in \mathbb{Z}_q^{w\times n} \times \mathbb{Z}_p^{w\times 1}$, let $\vec{b}'[i] \leftarrow 1$ if $\vec{b}'[i] \neq 0$ and $\vec{b}[i] \leftarrow 0$ otherwise, for $i \in [w]$. Send $(A, \vec{b}')$ to $\mathcal{A}$ and returns whatever returned from $\mathcal{A}$.

If the input is a valid $\mathsf{LWR}_{n,q,q',\lfloor\cdot\rceil}$ input, and $\vec{b}[i] = 0$, we know that $A\mathsf{sk} \in [0, r-1]$ and otherwise $\vec{b}[i] \neq 0$. This thus gives us a valid $\mathsf{LWR}_{n,q,2,R_r}$ sample. Otherwise, we obtain a random sample. □

With this intuition, we define our algorithm in Algorithm 3.

Theorem 4.2. *For any $n > 0, t > 0$ being a prime, $m > n\lceil\log(t)\rceil$, and $r \in \mathbb{Z}_t$, and $A \leftarrow_\$ \mathbb{Z}_t^{m\times n}$, f_A in Algorithm 3 is a $(t, m, r/t)$-PRG (Definition 2.2), under $\mathsf{LWR}_{n,t,2,R_r}$ is hard.*

Proof. The proof is straightforward. We design a hybrid scheme: replacing f_A with uniformly sampling $u \leftarrow \mathsf{Bern}(r/t)^w$ and return u. The adversary that can distinguish this hybrid and the original scheme breaks $\mathsf{LWR}_{n,t,2,R_r}$. □

Achieving $p = 1/2$. We have achieved values of $p \neq 1/2$, the choices are relatively restricted. The main obstacle to achieving arbitrary p is the finite field size t that we work over: it can be arbitrary and thus may not be compatible with p (i.e., there doesn't exist $r \in \mathbb{Z}_t$ such that $r/t = p$). One trivial solution, of course, is to choose t according to p. However, in the context of FHE, t is subject to certain constraints. Specifically, since we want to use FHE to evaluate the PRG, t is decided by the underlying FHE scheme, which can be constrained by other factors and has to be a prime for best efficiency (also, since $\mathsf{LWR}_{n,t,2,R}$ is trivially insecure for $t = 2$, that is not an option for us).

For our construction, we need to achieve $p = 1/2$ for an arbitrary prime t.

One natural idea is that if t is a prime of size $\Omega(2^\lambda)$, we can pick $r = \lceil t/2 \rceil$, and it holds that $r/t - 1/2 = \mathsf{negl}(\lambda)$. However, if t is too large, it cannot be used as an FHE plaintext modulus in practice, as normally the FHE plaintext modulus $< 2^{30}$ for best efficiency: the noise growth of BFV is linear in t, which means that for a larger plaintext field t, the number of levels that can be supported is much smaller. Furthermore, another issue with this idea is that our PRG circuit has a depth in $\log(t)$, and thus if $\log(t)$ is too large, the PRG circuit also becomes impractical. Thus, to avoid these issues, we devise ways to obtain $p = 1/2$ for any small t.

Achieving $\boldsymbol{p = 1/2}$ with small $\boldsymbol{t}$. We can output bits with distribution arbitrarily close to uniform while maintaining a small t using the following algorithm which takes in a seed $s \in \mathbb{Z}_t^n$:

1. Sample $\vec{a} \leftarrow_{\$} \mathbb{Z}_t^n$. Let $x = \langle \vec{a}, s \rangle$.
2. **If** $x = 0$, go to step 1. **Else** output $x > (t-1)/2$.

To make this algorithm deterministic, we can set a maximum number of k runs. If after k times, $\langle \vec{a}, s \rangle$ is 0 for all k repetitions, return 0. In this case, the probability of sampling $0 \leq 1/2 + t^{-k}$ and thus we can simply let $k = O(\lambda/\log(t))$ to obtain the exact security. With this intuition, we can construct a PRG with $p = 1/2$. For our PRG, it will also be important to make this algorithm oblivious by fixing the number of runs.

A more FHE-friendly circuit. Our goal is to make the sampling process FHE-friendly. Unfortunately, most FHE schemes only support multiplications and additions natively. Therefore, to evaluate a comparison to 0 (as above), requires transforming the circuit into a degree-$(t-1)$ function. Evaluation of this function would cost at least t multiplications for the comparison. Furthermore, when repeating k times as above, we need $k \cdot t$ multiplications.

To reduce such computation cost, we design an alternative function $R : \mathbb{Z}_t \to \mathbb{Z}_2$ that can be evaluated much more efficiently. Observe that for any prime t, $f(x) := x^{(t-1)/2}$ can only be $1, -1, 0$, and it is 0 if and only if $x = 0$. Thus, if $x \in \mathbb{Z}_t^*$, $x^{(t-1)/2}$ is 1 or -1. Moreover, half of the elements in $\mathbb{Z}_t^*$ are mapped to 1 and the other half to -1 [52]. With these observations, we construct the following function: $R(x) = (f(x) + 1) \cdot 2^{-1}$. Using repeated squaring, computing this function only takes $\log(t)$ multiplications. Therefore, even repeating the process $k = O(\lambda/\log(t))$ times, it only takes $k \cdot \log(t) = O(\lambda/\log(t) \cdot \log(t)) = O(\lambda)$ multiplications (with a depth of $O(\lambda/\log(t) + \log(t))$). Moreover, to check whether x is 0 also becomes easy: simply compute $1 - f(x)^2$, which returns 1 if x is 0 and returns 0 otherwise.

With these optimizations, we formalize our construction as in Algorithm 4.

Algorithm 4 $(t, m, 1/2)$-PRG

```
f(x) := $x^{(t-1)/2}$
procedure $f_{A_1,\ldots,A_k}(s)$                                   ▷ $A_i \leftarrow_{\$} \mathbb{Z}_t^{m \times n}$ for $i \in [k]$
    for $i \in [k]$ do
        $\mathsf{tmp}_i \leftarrow f(A_i s_i)$
        $\mathsf{Rnd}_i \leftarrow (\mathsf{tmp}_i + 1) \cdot 2^{-1} \in \mathbb{Z}_t$
        $\mathsf{Ind}_i \leftarrow \mathsf{tmp}_i^2 \in \mathbb{Z}_t$
    $\mathsf{res} \leftarrow \mathsf{Rnd}_1 \cdot \mathsf{Ind}_1 + (1 - \mathsf{Ind}_1)(\mathsf{Rnd}_2 \cdot \mathsf{Ind}_2) + \cdots + (\prod_{i \in [k-1]}(1 - \mathsf{Ind}_i))(\mathsf{Rnd}_k \cdot \mathsf{Ind}_k)$
    return $\mathsf{res}$
```

We will prove the security of Algorithm 4 through a series of hybrid experiments. For each Hybrid i, for $i \in [k]$, we will show indistinguishability from the previous hybrid by showing that $\mathsf{LWR}_{n,t,2,R_i}$ holds, where for each $i \in [k]$, we define $R_i(x)$ as:

$$R_i(x) = \begin{cases} 1 & \text{if } x^{(t-1)/2} = 1 \\ 0 & \text{if } x^{(t-1)/2} = t-1 \\ u_i & \text{otherwise} \end{cases}$$

where $u_i \leftarrow_\$ \mathsf{Bern}(p_i)$ for $p_i = \frac{(t^{k-i}-1)/2+1}{t^{k-i}}$, for $i \in [k-1]$; and $u_i = 0$ for $i = k$. We formalize the theorem as follows.

Theorem 4.3. *For any $n > 0, t > 0$ being a prime, $k > 0$ such that $t^{-k} = \mathsf{negl}(n)$, $m > n\lceil \log(t) \rceil$, then $f_{A_1,\ldots,A_k}$ in Algorithm 3 is a (t,m)-PRG (Definition 2.2) under $\mathsf{LWR}_{n,t,2,R_1}, \ldots, \mathsf{LWR}_{n,t,2,R_k}$ are hard.*

Proof. To prove this security, we design the following hybrids.

Hyb_0 : a rearrangement of our actually scheme, as follows.

1. If $A_1 s_1 \neq 0$: output $(R_1(A_1 s_1), d)$
2. If $A_2 s_2 \neq 0$: output $(R_2(A_2 s_2), d)$
3. ...
4. If $A_{k-1} s_{k-1} \neq 0$: output $(R_{k-1}(A_{k-1} s_{k-1}), d)$
5. output $(R_k(A_k s_k), d)$

Hyb_1 Same as Hyb_0, except that line 5 in Hyb_0 is replaced with: output $(R_k(u), d)$ where $u \leftarrow_\$ \mathbb{Z}_t^w$.

Hyb_2 : Same as Hyb_1, except that line 4 in Hyb_0 is replaced with: output $(R_{k-1}(u), d)$ where $u \leftarrow_\$ \mathbb{Z}_t^w$.

...

Hyb_{k-1} : Same as Hyb_{k-2}, except that line 2 in Hyb_0 is replaced with: output $(R_2(u), d)$ where $u \leftarrow_\$ \mathbb{Z}_t^w$.

Hyb_k : Same as Hyb_{k-1}, except that line 1 in Hyb_0 is replaced with: output $(R_1(u), d)$ where $u \leftarrow_\$ \mathbb{Z}_t^w$.

Clearly Hyb_0 is equivalent to our original scheme. Then, Hyb_i and Hyb_{i-1} for $i \in [k]$ are indistinguishable under $\mathsf{LWR}_{n,t,2,R_i}$ is hard. Furthermore, Hyb_k is by itself sampling from $\mathsf{Bern}(1/2^m)$ except with $O(t^{-k}) = \mathsf{negl}(\lambda)$ probability. Lastly, $k = \mathsf{poly}(\lambda)$, our original scheme is indistinguishable from Hyb_k, our scheme is a (t,m)-PRG. □

4.2 Shuffling via BFV

To perform the Thorp Shuffle obliviously, we will require an FHE scheme. The best performing scheme for this scenario is BFV. BFV encrypts a plaintext of form $\mathbb{Z}_t^D$ where D is the ring dimension. This means that a random vector of D random bits is generated using Random Sampling under BFV. Furthermore, directly realizing Thorp Shuffle is not very efficient using BFV, as it requires mapping an element at slot i to slot $2i$ or $2i+1$ for all $i \in [N/2]$. To do this in BFV, one needs to extract each element from the ciphertext and rotate it accordingly, which is very inefficient. Therefore, instead of directly realizing Thorp Shuffle, we use a butterfly shuffle: for round $\ell \in [n]$, position j is swapped with position $j + 2^{\ell-1}$ for all $j \in [N]$ such that $\lfloor \frac{j}{2^{\ell-1}} \rfloor$ is odd. As discussed in [20, Lemma 1], every n rounds of butterfly shuffle is equivalent to n rounds of Thorp Shuffle.

Then, the swapping becomes easy: for level ℓ, if $2^{\ell-1} < D$, it means that every elements are swapped with another element in the same ciphertext; thus, we separate a single ciphertext into two ciphertexts, use the random bits to swap them, and then recombine them. For example, if $\ell = 1$, we simply separate a ciphertext into two ciphertexts, where the first one contains all the odd slots of the original ciphertext while the second one contains all the even slots. Then, swap the two ciphertexts using the encrypted random bits. Note that one ciphertext contains D elements, but a swap inside a ciphertext requires only $D/2$ random bits. Thus, a ciphertext of random bits containing D bits can serve two swaps for two different ciphertexts. If $2^{\ell} \geq D$, simply swap two ciphertexts using a ciphertext containing D random bits.

The algorithm is formally presented below in Algorithm 5.

We then briefly discuss two optimizations.

Leveraging relaxed bootstrapping. Naively realizing Algorithm 5 requires $(r+1)\log(N)$ levels for shuffling plus $\log(t)+\lambda/\log(t)$ levels for PRG evaluation. This can cause the BFV parameters to be very large without bootstrapping. However, a direct application of regular BFV bootstrapping is very costly. To avoid this issue, we employ the relaxed bootstrapping technique introduced in [40]. Essentially, relaxed bootstrapping says that the correctness of bootstrapping holds *only* when the input is a fixed subset of the plaintext space, and its (amortized) efficiency can be orders of magnitude faster than regular BFV bootstrapping. This is exactly our case: we know that each ciphertext is encrypting a single bit instead of an arbitrary plaintext $\in \mathbb{Z}_t$ after the PRG evaluation is done. Therefore, we can set BFV parameters such that we can evaluate $\log(t) + c$ levels where c is the number of levels that we can compute using BFV after bootstrapping. Then, apply bootstrapping for every c levels for $(r + 1)\log(N)/c$ times.[4]

Encrypting with more than one bit. Instead of performing the shuffle one bit at a time, we can instead encode $p > 1$ bits at a time without increasing the runtime of the shuffle by much. However, note that since we are applying relaxed bootstrapping, we need $p < \log(t)$ (i.e., for relaxed bootstrapping above, we need a fixed subset of the plaintext space needed which is smaller than the real plaintext space). The exact value of p is decided by other BFV parameters as discussed in detail in [40], and we leave it for Section 5 to choose.

4.3 Putting Everything Together for Homomorphic Thorp Shuffle

With all these tools, we now present our homomorphic Thorp Shuffle algorithm in Algorithm 6. Essentially, we use the PRG we presented in Algorithm 4 as the PRG needed for Thorp Shuffle (so sample $s \leftarrow_{\$} \mathbb{Z}_t^n$ as seed, where n is some security parameter). Then, we use this seed to homomorphically generate $(r + 1)\log(N) \cdot N$ bits. Lastly, we use these bits to homomorphically perform the Thorp Shuffle.

Theorem 4.4. *If BFV with ring dimension D and plaintext t and among public parameters $\mathsf{pp}_{\mathsf{BFV}}$ is correct and semantically secure, for any database db of size $N = \mathsf{poly}(\lambda)$ being a power of two and that N/D is even, if $f_{A_1,\ldots,A_k}$ is a (t, m)-PRG (Definition 2.2) where m chosen in line 12, then for any adversary $\mathcal{A}$, any $q < N$, let $\mathsf{pp}, \mathsf{sk} \leftarrow \mathsf{Setup}(1^{\lambda}, q, N)$ and $\mathsf{ct} \leftarrow \mathsf{HomomorphicThorp}(\mathsf{db}, \mathsf{pp})$*

[4] For PRG, we can simply use fresh ciphertexts to evaluate, which can evaluate all the $\log(t) + \lambda/\log(t)$ levels without bootstrapping.

Algorithm 5 Homomorphic Thorp Shuffle via BFV. BFV ring dimension D being a power of two, plaintext modulus t. Shuffling parameters r,$N \in \mathbb{N}$, and N/D is an even integer. $n = \log_2(N)$.

```
procedure $\mathsf{SingleShuffle}_1(\mathsf{pk}_{\mathsf{BFV}}, \mathsf{ct}_{\mathsf{bits}}, \mathsf{ct}_{\mathsf{db}_1}, \mathsf{ct}_{\mathsf{db}_2}, \ell)$        ▷ All operations are under BFV
    Let $\vec{l} \leftarrow 1^L||0^L||1^L||\ldots||0^L \in \mathbb{Z}_t^D$ where $L \leftarrow 2^{\ell-1}$
    $\mathsf{ct}_{\mathsf{db}_1,1} \leftarrow \mathsf{ct}_{\mathsf{db}_1} \cdot \vec{l}$
    $\mathsf{ct}_{\mathsf{db}_1,2} \leftarrow \mathsf{Rotate}(\mathsf{ct}_{\mathsf{db}_1} - \mathsf{ct}_{\mathsf{db}_1,1}, -L)$
    $\mathsf{ct}_{\mathsf{bits},1} \leftarrow \mathsf{ct}_{\mathsf{bits}} \cdot \vec{l}$
    $\mathsf{ct}'_{\mathsf{db}_1,1} \leftarrow \mathsf{ct}_{\mathsf{db}_1,1} \cdot \mathsf{ct}_{\mathsf{bits},1}$
    $\mathsf{ct}'_{\mathsf{db}_1,2} \leftarrow \mathsf{ct}_{\mathsf{db}_1,2} \cdot (1 - \mathsf{ct}_{\mathsf{bits},1})$
    $\mathsf{ct}'_{\mathsf{db}_1} \leftarrow \mathsf{ct}'_{\mathsf{db}_1,1} + \mathsf{ct}'_{\mathsf{db}_1,2} + \mathsf{Rotate}(\mathsf{ct}_{\mathsf{db}_1,1} - \mathsf{ct}'_{\mathsf{db}_1,1} + \mathsf{ct}_{\mathsf{db}_1,2} - \mathsf{ct}'_{\mathsf{db}_1,2}, L)$
    $\mathsf{ct}_{\mathsf{db}_2,1} \leftarrow \mathsf{ct}_{\mathsf{db}_2} \cdot (\vec{1} - \vec{l})$
    $\mathsf{ct}_{\mathsf{db}_2,2} \leftarrow \mathsf{Rotate}(\mathsf{ct}_{\mathsf{db}_2} - \mathsf{ct}_{\mathsf{db}_2,1}, L)$
    $\mathsf{ct}_{\mathsf{bits},2} \leftarrow \mathsf{ct}_{\mathsf{bits}} - \mathsf{ct}_{\mathsf{bits},1}$
    $\mathsf{ct}'_{\mathsf{db}_2,1} \leftarrow \mathsf{ct}_{\mathsf{db}_2,1} \cdot \mathsf{ct}_{\mathsf{bits},2}$
    $\mathsf{ct}'_{\mathsf{db}_2,2} \leftarrow \mathsf{ct}_{\mathsf{db}_2,2} \cdot (1 - \mathsf{ct}_{\mathsf{bits},2})$
    $\mathsf{ct}'_{\mathsf{db}_2} \leftarrow \mathsf{ct}'_{\mathsf{db}_2,1} + \mathsf{ct}'_{\mathsf{db}_2,2} + \mathsf{Rotate}(\mathsf{ct}_{\mathsf{db}_2,1} - \mathsf{ct}'_{\mathsf{db}_2,1} + \mathsf{ct}_{\mathsf{db}_2,2} - \mathsf{ct}'_{\mathsf{db}_2,2}, -L)$
    Return $\mathsf{ct}'_{\mathsf{db}_1}, \mathsf{ct}'_{\mathsf{db}_2}$
procedure $\mathsf{SingleShuffle}_2(\mathsf{pk}_{\mathsf{BFV}}, \mathsf{ct}_{\mathsf{bits}}, \mathsf{ct}_{\mathsf{db}_1}, \mathsf{ct}_{\mathsf{db}_2})$        ▷ All operations are under BFV
    $\mathsf{ct}_{\mathsf{db}_1,1} \leftarrow \mathsf{ct}_{\mathsf{db}_1} \cdot \mathsf{ct}_{\mathsf{bits}}$
    $\mathsf{ct}_{\mathsf{db}_1,2} \leftarrow \mathsf{ct}_{\mathsf{db}_1} - \mathsf{ct}_{\mathsf{db}_1,1}$
    $\mathsf{ct}_{\mathsf{db}_2,1} \leftarrow \mathsf{ct}_{\mathsf{db}_2} \cdot (\vec{1} - \mathsf{ct}_{\mathsf{bits}})$
    $\mathsf{ct}_{\mathsf{db}_2,2} \leftarrow \mathsf{ct}_{\mathsf{db}_2} - \mathsf{ct}_{\mathsf{db}_2,1}$
    $\mathsf{ct}'_{\mathsf{db}_1} \leftarrow \mathsf{ct}_{\mathsf{db}_1,1} + \mathsf{ct}_{\mathsf{db}_2,1}$
    $\mathsf{ct}'_{\mathsf{db}_2} \leftarrow \mathsf{ct}_{\mathsf{db}_1,2} + \mathsf{ct}_{\mathsf{db}_2,2}$
    Return $\mathsf{ct}'_{\mathsf{db}_1}, \mathsf{ct}'_{\mathsf{db}_2}$
procedure $\mathsf{bfvThorp}(\mathsf{db} = (\mathsf{db}[1], \ldots, \mathsf{db}[N]), r, \mathsf{pp}_{\mathsf{BFV}}, \mathsf{pk}_{\mathsf{BFV}}, (\mathsf{ct}_{\mathsf{bits},i,j})_{i \in [(r+1)n], j \in [N/D/2]}))$
    $\mathsf{ct}_{\mathsf{db},j} \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}_{\mathsf{BFV}}, (\mathsf{db}[D \cdot j + 1], \ldots, \mathsf{db}[D \cdot (j+1) - 1])$ for $j \in [0, N/D - 1]$
    for $t$ in $[1, \ldots, r+1]$ do
        for $\ell$ in $[1, \ldots, n]$ do
            if $2^{\ell-1} < D$ then
                for $j$ in $[1, \ldots, N/D/2]$ do
                    $\mathsf{ct}'_{\mathsf{db},2j-1}, \mathsf{ct}'_{\mathsf{db},2j} \leftarrow \mathsf{SingleShuffle}_1(\mathsf{pk}_{\mathsf{BFV}}, \mathsf{ct}_{\mathsf{bits},(t-1)\cdot n+\ell,j}, \mathsf{ct}_{\mathsf{db},2j-1}, \mathsf{ct}_{\mathsf{db},2j}, \ell)$
            else
                $L \leftarrow 2^{\ell-1}/D$
                for $j$ in $[1, \ldots, N/D/2/L]$ do
                    for $k$ in $[1, \ldots, L]$ do
                        $\mathsf{ct}'_{\mathsf{db},2((j-1)L+k)-1}, \mathsf{ct}'_{\mathsf{db},2((j-1)L+k)} \leftarrow \mathsf{SingleShuffle}_2(\mathsf{pk}_{\mathsf{BFV}}, \mathsf{ct}_{\mathsf{bits},(t-1)\cdot n+\ell,(j-1)L+k},$
                                $\mathsf{ct}_{\mathsf{db},(j-1)L+k}, \mathsf{ct}_{\mathsf{db},jL+k})$
        $\mathsf{ct}_{\mathsf{db},\cdot} \leftarrow \mathsf{ct}'_{\mathsf{db},\cdot}$
    return $(\mathsf{ct}_{\mathsf{db},j})_{j \in [N/D]}$
```

Algorithm 6 Oblivious Permutation

```
Database $\mathsf{db}$ size $N \in \mathbb{N}$ if a power of two. $n = \log_2(N)$.
procedure $\mathsf{Setup}(\lambda, q, N)$
    Select BFV parameter $\mathsf{pp}_{\mathsf{BFV}} = (D, t, \dots)$ such that the following computation can be done homomorphically and is semantically secure. ▷ BFV parameters other than $D, t$ are irrelevant to us so we ignore it for simplicity.
    Generate BFV key pairs $\mathsf{pk}_{\mathsf{BFV}}, \mathsf{sk.BFV}$.
    Choose the minimum $k$ such that $t^{-k} = \mathsf{negl}(\lambda)$
    Choose the minimum $n = \mathsf{poly}(\lambda)$ such that $\mathsf{LWR}_{n,t,2,R_i}$ holds for $i \in [k]$.
                                                        ▷ $R_i$ defined in Theorem 4.3
    $s \leftarrow \mathbb{Z}_t^n$                       ▷ The PRG random seed.
    $\mathsf{ct}_s \leftarrow \mathsf{BFV.Enc}(\mathsf{pk}_{\mathsf{BFV}}, s)$
    Choose the minimum $r$ such that $\frac{q}{r+1}\left(\frac{2qn}{N}\right)^r$ is negligible in $\lambda$.
    return $\mathsf{pp} = (\mathsf{pp}_{\mathsf{BFV}}, \mathsf{pk}_{\mathsf{BFV}}, \mathsf{pp}_{\mathsf{ExactSampler}}, \mathsf{ct}_s, r), \mathsf{sk.BFV}$
procedure $\mathsf{HomomorphicThorp}(\mathsf{db}, \mathsf{pp}; d)$
    Let $m = N\log(N)r$
    $A_1, \dots, A_k \leftarrow_d \mathbb{Z}_t^{m \times n}$ (sample using randomness source $d$)
    Homomorphically evaluate $f_{A_1,\dots,A_k}$ to generate $(\mathsf{ct}_{\mathsf{bits},i,j})_{i\in[(r+1)\log(N)],j\in[N/D/2]})$ ciphertexts each containing $N$ random bits.
    $\mathsf{db}_{\mathsf{ct}} \leftarrow \mathsf{bfvThorp}(\mathsf{db}, r, \mathsf{pp}_{\mathsf{BFV}}, \mathsf{pk}_{\mathsf{BFV}}, (\mathsf{ct}_{\mathsf{bits},i,j}))$
    return $\mathsf{db}_{\mathsf{ct}}$
```

(both procedures in Algorithm 6), and $\mathsf{db}_2 \leftarrow \pi(\mathsf{db})$ *where* π *is a truly random permutation, let* $\mathsf{db}_1 \leftarrow \mathsf{BFV.Dec}(\mathsf{sk}, \mathsf{ct})$*, it holds that* $|\Pr[\mathcal{A}^{\mathcal{O}(\mathsf{db}_1,q)}(\mathsf{pp}, \mathsf{ct}, \mathsf{db}) = 1] - \Pr[\mathcal{A}^{\mathcal{O}(\mathsf{db}_2,q)}(\mathsf{pp}, \mathsf{ct}, \mathsf{db}) = 1]| = \mathsf{negl}(\lambda)$*, where* $\mathcal{O}(\mathsf{db}, q)$ *means an oracle access to an arbitrary location of string* db *for at most* q *times adaptively.*

Proof. This proof is trivial it is directly implied by the correctness of BFV, semantic security of BFV, (t, m)-PRG, and Theorem 3.2. □

4.4 Removing the Linear Bandwidth Constraint in Our PIR Scheme

In this subsection, we formalize how to remove the linear bandwidth constraint from our scheme in Algorithm 2. Notice that all we have to do is apply our oblivious permutation construction from Section 4 to the preprocessing phase. Specifically, rather than download the whole database and compute the hints itself, the client can sample public parameters and send these to the server. The server can then obliviously compute the permutation using Algorithm 6. We then define the following theorem.

Theorem 4.5. *The PIR scheme described in Algorithm 2 where the preprocessing is run by the server using Algorithm 6 is correct Definition 2.4 and private Definition 2.5 for* T *queries, and runs with the following complexities:*

- $O_\lambda(N)$ *preprocessing time with* $O_\lambda(1)$ *computation depth.*
- $O(Q + N/Q)$ *offline bandwidth.*

		Prior works [17, 59, 36]	Solution with sorting networks *Section* 1		Our solution *Theorem* 4.5	
			[1, 51, 15]	[6, 49, 54]	Old Bound [45]	Our Bound *Theorem* 3.2
Asymptotic Depth		$O(N)$	$O_\lambda(\log(N))$	$O_\lambda(\log^2(N))$	$O_\lambda(1)$	$O_\lambda(1)$
Concrete Depth		$> 2^{20}$	$> 2^{20}$	> 21,160	930	416
Estimated time (hrs)	w/ AES	1,579,453	N/A	87,703	N/A	N/A
	w/ *Algorithm* 4	140,160	N/A	25,102	892	399

Table 1: Comparison between the preprocessing procedure of our construction (Theorem 4.5) and our naive solution with sorting networks in Section 1, and prior works [17, 59, 36]. All the numbers are estimated with database size $N = 2^{23}$, each with 3 bits, on a single thread CPU. Our depth includes both the Thorp Shuffle depth and the PRG circuit depth.

- $O_\lambda(Q)$ *online bandwidth and query time.*
- $O(N/Q + T \cdot Q)$ *client storage.*
- $O_\lambda(1)$ *update time*

Proof. The proof follows directly from the proof of Theorem 3.1 and Theorem 4.4. □

5 Concrete Efficiency

Here, we discuss concrete efficiency of the techniques shown in the paper.

We estimate the depth and runtime of (1) our construction (Theorem 4.5), (2) our naive solution with sorting networks (Section 1), and (3) prior works and present them in Table 1.

Parameter selection. We estimate times on a database of $N = 2^{23}$ entries, each with 3 bits (for ease of comparison with prior works; we discuss how our runtime scales with the number of bits per entry later). We choose $q = Q = 2^7, K = 2^{16}$ as the parameters for our PIR scheme; and choose r according to Theorem 3.2 such that the adversary querying for q queries have at most 2^{-40} advantage. For the underlying BFV, we choose $D = 2^{15}$ for the ring dimension; 860 bits for the ciphertext modulus; and $t = 65537$ for the plaintext modulus. These parameters guarantee greater than 128 bit security by [19]. With [40], such parameter setting allows a relaxed bootstrapping in about 50 seconds with plaintext space being 3 bits, with about 450 bits noise budget left after bootstrapping. Since there are no concrete LWR security estimators, we heuristically estimate the security of our construction using the LWE estimator [19]. To do this, we make two heuristic assumptions: (1) $\mathsf{LWR}_{n,t,2,R_i}$ is equivalent to $\mathsf{LWR}_{n,t,2,\lfloor\cdot\rceil}$ for any R_i ($i \in [k]$) in Algorithm 4; and (2) $\mathsf{LWR}_{n,t,2,\lfloor\cdot\rceil}$ is equivalent to LWE with secret dimension n, ciphertext modulus q, and error from Gaussian distribution with standard deviation σ such that $\Pr_{e \leftarrow \chi_\sigma}[|e| < t/2] > 1 - \mathsf{negl}(\lambda)$. Under these two heuristic assumptions, we set and $n = 220, t = 65537$ (and we use $\sigma = 128$ which is more than sufficient to satisfy the conditions). Additionally, we set $k = \log_t(2^{80}) = 5$ (i.e., the number of repetitions for random bit sampling), to obtain a statistical security parameter of > 40-bits even after sampling $N\log(N)\lambda = 2^{23} \cdot 23 \cdot 40 \approx 2^{33}$ random bits[5]. With these parameters, we estimate our construction's cost accordingly. Our runtime are based on GCP instance `N2` with CPU Intel Xeon Gold 6268, 64GB RAM. We estimated using the single-thread runtime.

Estimation for other constructions. We estimate the runtime of prior works [17, 59, 36] generously. In particular, we estimate their cost using the depth that could theoretically be achieved by following the observation noted in Section 1. This greatly reduces their runtime. We also estimate the sorting network evaluation's runtime generously by considering a comparison of two λ-bit integers to be just λ homomorphic multiplications, and estimate the sorting network's runtime fixing the depth of the computation to $\log^2(N)/2$ levels (requiring only $N\log^2(N)/2$ multiplications to sort an array of any $n > 0$ bit elements). We do not use the numbers of [32] as this real implementation scales much worse than our estimation (thus giving us more advantages compared to prior works and the naive solution)[6]. Even given generous estimations, our new scheme greatly improves on all previous approaches. We give two separate numbers for every estimate using either AES as the PRG to sample encrypted bits (numbers taken from [58]), or our new FHE-friendly PRG (Algorithm 4).

Depth. As shown in the Table 1, our depth is both asymptotically and concretely better than an prior works by a large margin. This gives two main benefits: (1) to perform our computation over FHE, we need to bootstrap a lot less times, which greatly increases our efficiency; (2) since the circuit is inherently parallelizable.

Runtime. In terms of runtime, we perform orders of magnitude than prior works and than our naive solution using sorting networks, even on a single thread. One may use a GPU to accelerate FHE [4], which results in a ≈ 50x speedup, which means our preprocessing can be finished within 3.6 hours.

Our PRG. To generate 2^{15} random bits, our construction only takes about 13.1 seconds, and for 2^{20} random bits, it only takes about 420 seconds. This is more than 3 orders of magnitudes faster than homomorphically evaluating AES to generate random bits, which takes > 700000 seconds for 2^{20} bits [58].

Scalability with entry size. Our runtime is not greatly affected by increasing the number of bits in each entry. In particular, with 90 bits per entry, the runtime estimation is only increased from a factor of 1.54x, and for 180 bits per entry, the runtime is increased by 2.1x with respect to our number for 3 bits.

6 Proof of the Improved Mixing Time of the Thorp Shuffle

Now we give the proof for Theorem 3.2. The initial setup of the proof follow closely the setup of Morris et al. [45].

Proof. First, we setup some variables and definitions we will need.

Let μ and ν be probability distributions on an event space Ω, where Ω is any finite non-empty set. We say that a pair of random variables $W = (X,Y)$ is a *coupling* of μ and ν if its marginal

[5] More precisely, we need $N\log(N)R/2$ bits, where $R = O(\lambda)$ is the number of levels for the underlying Thorp shuffle. However, this difference does not change this parameter choice since the hidden constant is small, as shown in Table 1.

[6] For example, with 15625 elements, [32] sorts in 295 minutes per 32 thread, and our estimation gives 67 minutes, not to mention larger sizes.

distributions (i.e., distributions of X and Y) and μ and ν respectively. That is, for any $S \subseteq \Omega$ we have $\mathbf{P}(X \in S) = \mu(S)$ and $\mathbf{P}(Y \in S) = \nu(S)$. The total variation distance between μ and ν can be expressed as:

$$\Delta(\mu, \nu) := \max_{S \subseteq \Omega} |\mu(S) - \nu(S)| = \min_{(X,Y):\ X \sim \mu \text{ and } Y \sim \nu} \mathbf{P}(X \neq Y) \tag{1}$$

where $X \sim \mu$ means that X has the distribution μ. The minimum is thus taken over all possible couplings of μ and ν. Further explanation on these statements can be found on standard texts on Markov Chains such as the book by Levin et al. [38, Section 4].

Now, let M_t be the Markov chain representing the Thorp Shuffle with N cards at step t, where **we define a card to be an element in** $C := \{0,1\}^n$ (recall $N = 2^n$ is a parameter)[7]. Then, the state space of M_t is any bijection from C to $\{0,1\}^n$. Let us define $M_t(z)$ to be the position of a card z at time t.

Since the adversary we are considering only ever sees a subset of q elements in the shuffle, we only need to bound the rate at which all q-subsets of the shuffle are indistinguishable from q-subsets of a real permutation. Let $z_1, \ldots, z_q$ be distinct cards. We define X_t to be the vector of random variables for the positions of cards $z_1, \ldots, z_q$ at time t on the Thorp Shuffle M_t. For $j \in \{1, \ldots, q\}$, $X_t(j)$ will represent the position of card z_j at time t.

Recall that we defined our process X_t as the position of a q-tuple vector of cards in M_t after t steps. Now we explicitly define the update rule from X_t to X_{t+1} in a self-contained fashion (without requiring knowledge of other cards' positions). X_1 is defined as the vector of the initial position of the q cards selected. Then, we will update each card's position according to the Thorp Shuffle definition (Algorithm 1). To define X_{t+1} in a self-contained fashion, we have to decide each card's next position in the q-tuple using only the information in X_t. To do this, we define an equivalent rule for generating X_{t+1} from X_t as follows.

We define two cards z_i and z_j **to be matched**[8] at a timestep t if their positions, u_i^t and u_j^t respectively, satisfy $u_i^t = u_j^t \mod (N/2)$.

For every card $z_j, j \in \{1, \ldots, \ell+1\}$, for each timestep t, we sample a coin $c_t^j \xleftarrow{\$} \mathsf{Bern}(1/2)$. We determine the position of card z_j at time $t+1$ as follows.

- **If** card z_j *is not matched to any card* z_i *where* $i < j$, it is moved to position $2(u_j^t \mod N/2) + c_t^j$.
- **Else If** card z_j *is* matched to a card z_i where $i < j$, it is moved to $2(u_j^t \mod N/2) + \neg c_t^i$.

For each card z_j, $j \in \{1, \ldots, \ell+1\}$, and each round t, we will also define a new variable d_t^j, where:

- $d_t^j = c_t^j$ **if** z_j is not matched to any z_i with $i < j$.
- $d_t^j = \neg c_t^i$ **if** z_j is matched to a card z_i with $i < j$, for $i \in \{1, \ldots, \ell+1\}$.

We say that d_t^j is **the coin associated with** card z_j at time t.

[7] More generally, C can be any set of cardinality N, but here we restrict it to $\{0,1\}^n$ for concreteness.

[8] Two cards being matched at timestep t means that they will be placed next to each other in the next round.

Next, we define another random process, U_t, to behave exactly as X_t, except we define U_1 to be q uniform samples without replacement from $\{0,1\}^n$, rather than to be defined from the cards' true initial state as in X_1.

Recall that our goal is to show that the Thorp shuffle (Algorithm 1) is indistinguishable from a real random permutation given only q (parallel) accesses.

Now, in order to bound the total variation distance between q-subsets of the Thorp Shuffle and a uniform permutation using the coupling above, we will introduce a lemma that relates the distance between the distributions to the expected distance betwen two distributions over conditional distributions.

Before introducing the lemma, we introduce some notation. For a distribution ν on distinct q-tuples of Ω, and $(Z_1, \dots Z_\ell) \sim \nu$, we will use the following notation (at the risk of a slight abuse of notation):

$$\nu(u_1, \dots u_\ell) = \mathbf{P}(Z_1 = u_1, \dots, Z_\ell = u_\ell),$$

and

$$\nu(u_\ell \mid u_1, \dots, u_{\ell-1}) = \mathbf{P}(Z_\ell = u_\ell \mid Z_1 = u_1, \dots Z_{\ell-1} = u_{\ell-1}).$$

Note that $\nu(Z_\ell|u_1,\dots,u_{\ell-1})$ is the distribution of Z_ℓ conditioned on $(Z_1 = u_1,...,Z_{\ell-1} = u_{\ell-1})$ and thus $\nu(Z_{\ell+1}|Z_1,\dots,Z_\ell)$ is a random variable over conditional distributions. For two q-tuple distributions μ and ν we will use the notation $\Delta(\mu(Z_{\ell+1}|Z_1,\dots,Z_\ell), \nu(Z_{\ell+1}|Z_1,\dots,Z_\ell))$ to denote the random variable representing the distance between the conditional distributions (*not to be confused with* the distance between the two random variables over conditional distributions), i.e. for any assignment $(u_1,...,u_\ell)$ to $Z_1,...,Z_\ell$ this random variable takes on a real-valued number $\Delta(\mu(Z_{\ell+1}|u_1,\dots,u_\ell), \nu(Z_{\ell+1}|u_1,\dots,u_\ell))$.

Lemma 6.1 (Lemma 2 in [45])**.** *Fix a finite nonempty set Ω and let μ and ν be probability distributions supported on $(\ell+1)$-tuples of elements of Ω, and suppose that $(Z_1, \dots, Z_q) \sim \mu$. Then,*

$$\Delta(\mu, \nu) \leq \sum_{\ell=0}^{q-1} \mathbf{E}\left(\Delta\left(\mu(Z_{\ell+1} \mid Z_1, \dots, Z_\ell), \nu(Z_{\ell+1} \mid Z_1, \dots, Z_\ell)\right)\right). \tag{2}$$

The proof of this lemma is shown in [45] so we omit the details.

Next, we will use this lemma to upperbound the total variation distance between the q-subsets of the Thorp Shuffle and a permutation. Define π to be the distribution of q uniform independent samples without replacement from $\{0,1\}^n$, and τ_t as the *distribution* of X_t. Note that with the update we define previously, for any $t \geq 1$, the distribution of X_t is exactly the same as the marginal distribution of the Thorp Shuffle (Algorithm 1; trivially by how the two processes are defined) over the q-subset. Furthermore, the distribution of U_t is always just π, since U_1 is q uniform samples and the Markov process converges to a uniform distribution with enough time [43]. Thus, by Equation (1), for any $\ell < q$ and any $u_1, \dots, u_\ell$, and $(Z_1, \dots, Z_{\ell+1}) \sim \tau_t$, we have:

$$\begin{aligned}\Delta\left(\tau_t(Z_{\ell+1}|Z_1 = u_1, \dots, Z_\ell = u_\ell), \pi(Z_{\ell+1}|Z_1 = u_1, \dots, Z_\ell = u_\ell)\right) &\leq \mathbf{P}(X_t \neq U_t|Z_1 = u_1, \dots, Z_\ell = u_\ell)) \\ &= \mathbf{P}(T_{\ell+1} > t|Z_1 = u_1, \dots, Z_\ell = u_\ell).\end{aligned}$$

Treating $\mathbf{P}(X_t \neq U_t | Z_1, \ldots, Z_\ell)$ as a random variable over probabilities, we can summarize this equation as:

$$\Delta\left(\tau_t(Z_{\ell+1}|Z_1, \ldots, Z_\ell), \pi(Z_{\ell+1}|Z_1, \ldots, Z_\ell)\right) \leq \mathbf{P}(X_t \neq U_t | Z_1, \ldots, Z_\ell))$$

By definition of our U_t and X_t processes, we have that:

$$\mathbf{P}\left(X_t \neq U_t \mid Z_1, \ldots, Z_\ell\right) = \mathbf{P}(T_{\ell+1} > t \mid Z_1, \ldots, Z_\ell),$$

where we define $T_{\ell+1} = \min\{t : X_t = U_t\}$, and again, the right-hand side is a random variable. With this, we conclude that the *expected* value of the variational distance is less than or equal to the unconditional probability that $T_{\ell+1} > t$, written as follows:

$$\begin{aligned} &\mathbf{E}\left(\Delta\left(\tau_t(Z_{\ell+1} \mid Z_1, \ldots Z_\ell), \pi(Z_{\ell+1} \mid Z_1, \ldots, Z_\ell)\right)\right) \\ &\leq \sum_{(u_1, \ldots, u_\ell) \in \{0,1\}^n} \mathbf{P}\left(T_{\ell+1} > t \mid Z_1 = u_1, \ldots, Z_\ell = u_\ell\right) \mathbf{P}(Z_1 = u_1, \ldots, Z_\ell = u_\ell) \\ &= \mathbf{P}\left(T_{\ell+1} > t\right). \end{aligned} \tag{3}$$

Applying Lemma 6.1 to and Equation (3), we get that:

$$\Delta\left(\tau_t, \pi\right) \leq \sum_{\ell=0}^{q-1} \mathbf{E}\left(\Delta\left(\tau_t(Z_{\ell+1} \mid Z_1, \ldots, Z_\ell), \pi(Z_{\ell+1} \mid Z_1, \ldots, Z_\ell)\right)\right) \leq \sum_{\ell=0}^{q-1} \mathbf{P}(T_{\ell+1} > t), \tag{4}$$

therefore, for the rest of this proof this is what we will attempt to bound.

An important equation. Crucial to our proof will be the following equation [45, Page 8, eq 3], for any $t > n$:

$$\mathbf{P}(z_i \text{ and } z_j \text{ are matched at time } t) \leq 2^{1-n}. \tag{5}$$

Recall that at each step t, for each card z_j, we flip a coin c_j^t for this card. If z_j is not matched to any z_i where $i < j$, then d_j^t, *the card associated with* z_j at round t is defined to be c_j^t; else, we define d_t^j to be $\neg c_t^i$. We will use Equation (5) to bound the probability that $d_t^j \neq c_t^j$ in the following equation.

In this equation, we bound the probability that card that the coin associated with card z_j at time t, d_j^t is not its own coin c_j^t to be less than or equal to 2^{-n}. Formally, for any $t > n$, for any $j \in \{1, \ldots q\}$,

$$\mathbf{P}(d_t^j \neq c_t^j) \leq (j-1)2^{-n} \tag{6}$$

Proof. By definition, whenever z_j is not matched to any coin of smaller index, $c_j^t = d_t^j$. Furthermore, even if z_j is matched to some card of z_i where $i < j$, notice that if $c_i^t = \neg c_j^t$, it is still the case that $d_t^j = c_t^j$. So $d_t^j \neq c_t^j$ if and only if both (1) z_j is matched to some z_i for $i < j$ **and** (2) $c_t^i = c_t^j$. Notice that we can bound the probability of (1) to be less than or equal to $(j-1)2^{-n}$ using a union bound and Equation (5).

Also, the coins c_t^i, c_t^j are sampled from $\mathsf{Bern}(1/2)$, so the probability of (2) is $\frac{1}{2}$. Finally, the the positions of z_i and z_j at timestep t are independent from coins c_t^i, c_t^j, (since their positions are defined by timesteps up to $t-1$). Therefore, the events (A) $c_i^t = c_j^t$ **and** (B) whether z_i and z_j are matched are independent. So the probability of both happening is simply the product of the probability of each which is upper bounded by $(j-1)2^{1-n}\left(\frac{1}{2}\right) = (j-1)2^{-n}$. □

Next, we will look at a second important lemma which, combined with Equation (6), will allow us to bound $P(T_{\ell+1} > t)$ for all $\ell \in \{0, \ldots, q-1\}$. This lemma tells us that a card's position is uniquely defined by that past n coins *associated with* that card, and any coin before that is not necessary to derive its position.

Lemma 6.2. *The past n coins associated with a card z uniquely define its current position u.*

Proof. This follows straight from the update rule defined. Consider a card z with position u_{t+1} at time $t+1$, with coins $d_{t-n+1}, \ldots, d_t$ used to update its position in the previous n timesteps. Then,

$$\begin{aligned} u_{t+1} &= 2((2\left((2(u_{t-n+1} \mod N/2) + d_{t-n+1}\right) \mod N/2) \\ &\qquad\qquad\qquad\qquad + d_{t-n+2}) \ldots \mod N/2) + d_t \\ &= \left(2^n(u_{t-n+1}) + \sum_{i=1}^{n} 2^{n-i} d_{t-n+i}\right) \equiv \sum_{i=1}^{n} 2^{n-i} d_{t-n+i} \mod N. \end{aligned}$$

□

Remark. ▶ An alternative view of Lemma 6.2 using bitwise operations may be help understanding. Given a card's current position as an $n-$bit string and its $t-$th coin d_t, each update chops off the most significant bit of the string, left shifts it by one position, and xors the bit d_t into it. From this perspective, it is clear to see that all the bits of a card's current position will be erased within n updates. ◀

Notice that Lemma 6.2 implies that, if $z_{\ell+1}$ is assigned using the same coins in both the X and U processes for n consecutive timesteps, then it must be that $z_{\ell+1}$ is in the same position in both processes. By what we just saw above, this means that if this happens, then $X_t = U_t$

We define event $A_{a:b}$ to be the probability that for some timestep t in $\{a, \ldots, b\}$, for any $i \in \{1, \ldots, \ell\}$, $z_{\ell+1}$ is associated with a different coin in processes X and U.

Recall from Equation (6), for any $t > n$, we have that $\mathbf{P}(d_t^{\ell+1} \neq c_t^{\ell+1}) \leq \ell \cdot 2^{-n}$. We can then bound that the probability that card $z_{\ell+1}$ *is not assigned* $c_t^{\ell+1}$ in any of the two processes, over any of the $b-a+1$ steps (for $a \geq n$) to be less than or equal to $2\ell \cdot (b-a+1) \cdot 2^{-n}$ (using union bound).

By definition of our Markov processes, once $z_{\ell+1}$ is the same position in both processes, it will remain in the same position in both processes from then onwards. Thus, we can bound:

$$\mathbf{P}(T_{\ell+1} > 2n) \leq \mathbf{P}(A_{n:2n-1}) \leq 2\ell n \cdot 2^{-n}. \tag{7}$$

Recall that $T_{\ell+1} = \min\{t : X_t = U_t\}$.[9]

Next, we will try to amplify this bound by considering what happens after more rounds. To do this, we need some more inequalities. Consider the probability of the cards z_i and z_j being matched for some timestep $t > kn$ given that event $A_{(k-1)n:kn-1}$ has taken place for any $k \geq 2$.

[9] Up to now, our result and argument is very similar to before, with an improvement of a factor of two w.r.t. [45].

Lemma 6.3. *For any $k \geq 2$, $t > kn - 1$:*

$$\mathbf{P}\left(z_i \text{ and } z_j \text{ are matched at time } t \mid A_{(k-1)n:kn-1}\right) \leq 2^{1-n}.$$

Proof. By definition, z_i and z_j are matched if and only if the last $n-1$ coins associated with their positions is the same.

For each timestep $t > kn-1$, conditioned on $A_{(k-1)n:kn-1}$, recall that the coins associated with z_i and z_j's position are $(d^i_{t-1}, \ldots, d^i_{t-n})$ and $(d^j_{t-1}, \ldots, d^j_{t-n})$, respectively. We will then bound the probability of these sets of coins being equal using two cases.

Case 1: $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$ are matched for some time step $\boldsymbol{m \in \{t-1, \ldots, t-n+1\}}$. First, notice that if for any $m \in \{t-1, \ldots, t-n+1\}$, z_i and z_j are matched, then by definition, $d^i_m \neq d^j_m$ and therefore the probability of $(d^i_{t-1}, \ldots, d^i_{t-n})$ being equal to $(d^j_{t-1}, \ldots, d^j_{t-n})$ is 0 conditioned on z_i being matched to z_j at some $m \in \{t-1, \ldots, t-n+1\}$.

Case 2: $\boldsymbol{z_i}$ and $\boldsymbol{z_j}$ are not matched for any time step $\boldsymbol{m \in \{t-1, \ldots, t-n+1\}}$. Let us bound the probability of $d^i_m = d^j_m$ for each step $m \in \{t-1, \ldots, t-n+1\}$ individually. We will show that for every $m \in \{t-1, \ldots, t-n+1\}$, d^i_m and d^j_m are defined by *independent* samples from $\mathsf{Bern}(1/2)$.

W.l.o.g. consider z_i. In each step, z_i is either matched to no cards with index smaller than it, in which case its coin is sampled from $\mathsf{Bern}(1/2)$ or it is matched to some card of smaller index than it, in which case it gets assigned the negation of that card's coin, which is a sample from $\mathsf{Bern}(1/2)$.

The same is true for z_j, but notice that (1) we are restricting to the case that z_i and z_j are not matched, and (2) even if z_i and z_j are both matched to cards of smaller indices, they cannot be matched to the same card.[10] Then, it follows that in this case, d^i_m and d^j_m are always defined by distinct samples of $\mathsf{Bern}(1/2)$ and therefore the probability that they are equal is 1/2.

Let B be the event that at time t, z_i and z_j were matched in any of the past $n-1$ steps. We have shown that:

$$\mathbf{P}\left(z_i \text{ and } z_j \text{ are matched at time } t \mid A_{(k-1)n:kn-1}, \neg B\right) = 2^{1-n}.$$

And we have also shown that:

$$\mathbf{P}\left(z_i \text{ and } z_j \text{ are matched at time } t \mid A_{(k-1)n:kn-1}, B\right) = 0.$$

These two equations suffice to prove our lemma. □

Now, given Lemma 6.3, we can show the following for any $t \geq kn$, for any $j \in \{1, \ldots q\}$:

$$\mathbf{P}\left(d^t_j \neq c^t_j \mid A_{(k-1)n:kn-1}\right) \leq (j-1)2^{-n} \tag{8}$$

This holds by the same argument as in Equation (6), given Lemma 6.3.

Finally, by the same arguments as previously, we can use a union bound to get that for any $k \geq 2$:,

$$\mathbf{P}\left(A_{kn:(k+1)n-1} \mid A_{(k-1)n:kn-1}\right) \leq 2\ell n \left(2^{-n}\right) = p. \tag{9}$$

[10] This follows from the pigeonhole principle. If three cards share the same last $n-1$ bits, then at least two of them must have shared those $n-1$ bits in the previous round, and therefore have a distinct last bit. A contradiction.

Since two coins' adjacency is defined by its coins at the past $n-1$ timesteps, it is also straightforward that any event that happened further than n steps ago does not affect the probability of adjacency between two cards. In other words,

$$\mathbf{P}\left(A_{kn:(k+1)n-1} \mid A_{(k-1)n:kn-1}, A_{(k-2)n:(k-1)n-1}\right) \tag{10}$$
$$= \mathbf{P}\left(A_{kn:(k+1)n-1} \mid A_{(k-1)n:kn-1}\right) \leq p. \tag{11}$$

Applying this more generally, we have that, for any $k > 0$:

$$\begin{aligned}
&\mathbf{P}(A_{n:2n-1} \wedge A_{2n:3n-1} \wedge \ldots \wedge A_{kn:(k+1)n-1}) \\
&= \mathbf{P}(A_{n:2n-1}) \cdot \mathbf{P}(A_{2n:3n-1} \mid A_{n:2n-1}) \cdot \ldots \cdot \mathbf{P}(A_{kn:(k+1)n-1} \mid A_{n:2n-1}, \ldots, A_{(k-1)n:kn-1}) \\
&= \mathbf{P}(A_{n:2n-1}) \cdot \mathbf{P}(A_{2n:3n-1} \mid A_{n:2n-1}) \cdot \ldots \cdot \mathbf{P}(A_{kn:(k+1)n-1} \mid A_{(k-1)n:kn-1}) \leq p^k
\end{aligned}$$

We can go from the second line to the third line by Equation (10). Then, the inequality on the third line follows by Equation (9). Looking back to our original goal of bounding $P(T_\ell > t)$, notice that unless A happens in every n interval, the two processes will be coupled. Therefore, we can now say that for any $r > 0$, for any $\ell \in \{0, \ldots, q-1\}$, $P\left(T_\ell > (r+1)n\right) \leq p^r$.

Finally, we put it all together to get:

$$\begin{aligned}
\Delta\left(\tau_{t=(r+1)n}, \pi\right) \leq \sum_{\ell=0}^{q-1} \mathbf{P}\left(T_{\ell+1} > (r+1)n\right) &\leq \sum_{\ell=0}^{q-1} \left(\frac{2\ell n}{N}\right)^r \\
&\leq \frac{q}{r+1}\left(\frac{2qn}{N}\right)^r \\
&= \frac{q(n+t+1)}{n+1}\left(\frac{2qn}{N}\right)^{t/(n+1)}.
\end{aligned}$$

The first line holds by a union bound, the second line holds because we take the integral (we define $t = (r+1)n$ to facilitate integration) and then just plug in. By [45], the total variational distance between two distributions is exactly equal to the CPA advantage of any stateful non-adaptive adversary distinguishing between them.

Furthermore, given a bound for non adaptive q-query adversaries in the CPA experiment, the result by Maurer et al. [42] tells us that we can enhance the secure to hold for any *adaptive, CCA* adversary with few additional rounds. Specifically, applying [42] to our work gives us that for any adaptive q-query adversary, the probability that it can distinguish a Thorp shuffle over $N = 2^n$ cards with t rounds from a true random permutation in a CCA experiment is less than or equal to

$$\frac{q(2n+t+2)}{n+1}\left(\frac{2qn}{N}\right)^{t/(2(n+1))}.$$

□

References

[1] M. Ajtai, J. Komlós, and E. Szemerédi. An 0(n log n) sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 1–9, New York, NY, USA, Dec. 1983. Association for Computing Machinery.

[2] J. Alperin-Sheriff and C. Peikert. Practical bootstrapping in quasilinear time. In R. Canetti and J. A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 1–20, Santa Barbara, CA, USA, Aug. 18–22, 2013. Springer, Heidelberg, Germany.

[3] G. Asharov, T.-H. H. Chan, K. Nayak, R. Pass, L. Ren, and E. Shi. Bucket Oblivious Sort: An Extremely Simple Oblivious Sort. In *Symposium on Simplicity in Algorithms (SOSA)*, Proceedings, pages 8–14. Society for Industrial and Applied Mathematics, Dec. 2019.

[4] A. A. Badawi, B. Veeravalli, C. F. Mun, and K. M. M. Aung. High-performance FV somewhat homomorphic encryption on GPUs: An implementation using CUDA. *IACR TCHES*, 2018(2):70–95, 2018. `https://tches.iacr.org/index.php/TCHES/article/view/875`.

[5] A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737, Cambridge, UK, Apr. 15–19, 2012. Springer, Heidelberg, Germany.

[6] K. E. Batcher. Sorting networks and their applications. *Proceedings of the April 30–May 2, 1968, spring joint computer conference on - AFIPS '68 (Spring)*, page 307, 1968. Conference Name: the April 30–May 2, 1968, spring joint computer conference Place: Atlantic City, New Jersey Publisher: ACM Press.

[7] A. Beimel, Y. Ishai, and T. Malkin. Reducing the Servers Computation in Private Information Retrieval: PIR with Preprocessing. In M. Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, Lecture Notes in Computer Science, pages 55–73, Berlin, Heidelberg, 2000. Springer.

[8] V. E. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic*. Academic Press, 1965. Google-Books-ID: nQcjAAAAMAAJ.

[9] E. Boyle, N. Gilboa, and Y. Ishai. Function Secret Sharing: Improvements and Extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1292–1303, New York, NY, USA, Oct. 2016. Association for Computing Machinery.

[10] E. Boyle, Y. Ishai, R. Pass, and M. Wootters. Can We Access a Database Both Locally and Privately? pages 662–693, Nov. 2017.

[11] Z. Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO 2012*, LNCS. Springer, Aug. 19–23, 2012.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In *ITCS 2012*. ACM, Jan. 8–10, 2012.

[13] Z. Brakerski and V. Vaikuntanathan. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In P. Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, Lecture Notes in Computer Science, pages 505–524, Berlin, Heidelberg, 2011. Springer.

[14] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. pages 41–41. IEEE Computer Society, Oct. 1995. ISSN: 0272-5428.

[15] V. Chvátal. ON THE NEW AKS SORTING NETWORK. 1992.

[16] S. Colombo, K. Nikitin, H. Corrigan-Gibbs, D. J. Wu, and B. Ford. Authenticated private information retrieval. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, pages 3835–3851, USA, Aug. 2023. USENIX Association.

[17] H. Corrigan-Gibbs, A. Henzinger, and D. Kogan. Single-Server Private Information Retrieval with Sublinear Amortized Time. In *Advances in Cryptology – EUROCRYPT 2022: 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 – June 3, 2022, Proceedings, Part II*, pages 3–33, Berlin, Heidelberg, May 2022. Springer-Verlag.

[18] H. Corrigan-Gibbs and D. Kogan. Private Information Retrieval with Sublinear Online Time. In A. Canteaut and Y. Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, Lecture Notes in Computer Science, pages 44–75, Cham, 2020. Springer International Publishing.

[19] B. Curtis, C. Lefebvre, F. Virdia, F. Göpfert, J. Owen, L. Ducas, M. Schmidt, M. Albrecht, R. Player, and S. Scott. Security estimates for the learning with errors problem.

[20] A. Czumaj and B. Vöcking. Thorp Shuffling, Butterflies, and Non-Markovian Couplings. In *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 344–355, Berlin, Heidelberg, 2014. Springer.

[21] A. Davidson, G. Pestana, and S. Celi. FrodoPIR: Simple, Scalable, Single-Server Private Information Retrieval. *Proceedings on Privacy Enhancing Technologies*, 2023(1), 2023.

[22] M. Dietz and S. Tessaro. Fully Malicious Authenticated PIR, 2023. Publication info: Preprint.

[23] J. Fan and F. Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `https://ia.cr/2012/144`.

[24] R. A. Fisher and F. Yates. *Statistical tables for biological, agricultural and medical research, edited by R.A. Fisher and F. Yates. 6th ed.* Edinburgh: Oliver and Boyd, 1963. Accepted: 2006-06-27T07:57:52Z.

[25] E. Gelman and A. Ta-Shma. The Benes Network is q*(q-1)/2n-Almost q-set-wise Independent. In *DROPS-IDN/v2/document/10.4230/LIPIcs.FSTTCS.2014.327*. Schloss-Dagstuhl - Leibniz Zentrum für Informatik, 2014.

[26] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing*, STOC '09, page 169–178. ACM, 2009.

[27] N. Gilboa and Y. Ishai. Distributed Point Functions and Their Applications. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, Lecture Notes in Computer Science, pages 640–658, Berlin, Heidelberg, 2014. Springer.

[28] S. M. Hafiz and R. Henry. A Bit More Than a Bit Is More Than a Bit Better: Faster (essentially) optimal-rate many-server PIR. *Proceedings on Privacy Enhancing Technologies*, 2019(4):112–131, Oct. 2019.

[29] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. page 27, 2022.

[30] V. T. Hoang, B. Morris, and P. Rogaway. An Enciphering Scheme Based on a Card Shuffle. Technical Report arXiv:1208.1176, arXiv, Nov. 2014. arXiv:1208.1176 [cs] type: article.

[31] J. Holmgren, R. Canetti, and S. Richelson. Towards Doubly Efficient Private Information Retrieval. Technical Report 568, 2017.

[32] S. Hong, S. Kim, J. Choi, Y. Lee, and J. H. Cheon. Efficient Sorting of Homomorphic Encrypted Data With k-Way Sorting Network. *IEEE Transactions on Information Forensics and Security*, 16:4389–4404, 2021. Conference Name: IEEE Transactions on Information Forensics and Security.

[33] D. Kogan and H. Corrigan-Gibbs. Private Blocklist Lookups with Checklist. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 875–892. USENIX Association, 2021.

[34] E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 364–373, Miami Beach, FL, USA, 1997. IEEE Comput. Soc.

[35] A. Lazzaretti, Z. Liu, B. Fisch, and C. Papamanthou. Single Server PIR via Homomorphic Thorp Shuffles. Preprint (under submission), 2024.

[36] A. Lazzaretti and C. Papamanthou. Near-Optimal Private Information Retrieval with Preprocessing. In G. Rothblum and H. Wee, editors, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 406–435, Cham, 2023. Springer Nature Switzerland.

[37] A. Lazzaretti and C. Papamanthou. TreePIR: Sublinear-Time and Polylog-Bandwidth Private Information Retrieval from DDH. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part II*, pages 284–314, Berlin, Heidelberg, Aug. 2023. Springer-Verlag.

[38] D. A. Levin, Y. Peres, and E. L. Wilmer. *Markov Chains and Mixing Times.* American Mathematical Soc., 2009. Google-Books-ID: 6Cg5Nq5sSv4C.

[39] W.-K. Lin, E. Mook, and D. Wichs. Doubly Efficient Private Information Retrieval and Fully Homomorphic RAM Computation from Ring LWE, 2022. Report Number: 1703.

[40] Z. Liu and Y. Wang. Relaxed functional bootstrapping: A new perspective on bgv/bfv bootstrapping. Cryptology ePrint Archive, Paper 2024/172, 2024. `https://eprint.iacr.org/2024/172`.

[41] Y. Ma, Z. Ke, T. Rabin, and S. Angel. Incremental Offline/Online PIR (extended version). In *USENIX Security 2022*, 2022.

[42] U. Maurer, K. Pietrzak, and R. Renner. Indistinguishability Amplification. In A. Menezes, editor, *Advances in Cryptology - CRYPTO 2007*, pages 130–149, Berlin, Heidelberg, 2007. Springer.

[43] B. Morris. Improved mixing time bounds for the Thorp shuffle and L-reversal chain. *The Annals of Probability*, 37(2):453–477, Mar. 2009. Publisher: Institute of Mathematical Statistics.

[44] B. Morris and P. Rogaway. Sometimes-Recurse Shuffle. In P. Q. Nguyen and E. Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, Lecture Notes in Computer Science, pages 311–326, Berlin, Heidelberg, 2014. Springer.

[45] B. Morris, P. Rogaway, and T. Stegers. How to Encipher Messages on a Small Domain. In *Advances in Cryptology - CRYPTO 2009*, volume 5677, pages 286–302. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. Series Title: Lecture Notes in Computer Science.

[46] M. H. Mughees, S. I, and L. Ren. Simple and Practical Amortized Sublinear Private Information Retrieval, 2023. Publication info: Preprint.

[47] O. Ohrimenko, M. T. Goodrich, R. Tamassia, and E. Upfal. The Melbourne Shuffle: Improving Oblivious Storage in the Cloud. In *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 556–567, Berlin, Heidelberg, 2014. Springer.

[48] H. Okada, R. Player, S. Pohmann, and C. Weinert. Towards Practical Doubly-Efficient Private Information Retrieval, 2023. Publication info: Published elsewhere. Minor revision. Financial Cryptography and Data Security 2024.

[49] I. Parberry. The Pairwise Sorting Network. *Parallel Processing Letters*, 2:205–211, Sept. 1992.

[50] S. Patel, G. Persiano, and K. Yeo. CacheShuffle: An Oblivious Shuffle Algorithm Using Caches. *ArXiv*, May 2017.

[51] M. S. Paterson. Improved sorting networks withO(logN) depth. *Algorithmica*, 5(1):75–92, June 1990.

[52] W. Raji. *An Introductory Course in Elementary Number Theory*. July 2013.

[53] R. L. Rivest, L. Adleman, and M. L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, pages 169–179, 1978.

[54] F. Shi, Z. Yan, and M. Wagh. An enhanced multiway sorting network based on n-sorters. *2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*, pages 60–64, Dec. 2014. Conference Name: 2014 IEEE Global Conference on Signal and Information Processing (GlobalSIP) ISBN: 9781479970889 Place: Atlanta, GA, USA Publisher: IEEE.

[55] E. Thorp. Nonrandom Shuffling with Applications to the Game of Faro. *Journal of The American Statistical Association - J AMER STATIST ASSN*, 68:842–847, Dec. 1973.

[56] A. Waksman. A Permutation Network. *Journal of the ACM*, 15(1):159–163, Jan. 1968.

[57] Y. Wang, J. Zhang, J. Liu, and X. Yang. Crust: Verifiable And Efficient Private Information Retrieval with Sublinear Online Time, 2023. Publication info: Preprint.

[58] B. Wei, R. Wang, Z. Li, Q. Liu, and X. Lu. Fregata: Faster homomorphic evaluation of aes via tfhe. In E. Athanasopoulos and B. Mennink, editors, *Information Security*, pages 392–412, Cham, 2023. Springer Nature Switzerland.

[59] M. Zhou, W.-K. Lin, Y. Tselekounis, and E. Shi. Optimal Single-Server Private Information Retrieval. *ePrint IACR*, 2022.

[60] M. Zhou, A. Park, E. Shi, and W. Zheng. Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation, 2023. Publication info: Published elsewhere. Major revision. IEEE S&P 2024.

A PIR Theorem Proof

In this section, we set out to prove Theorem 3.1.

A.1 A Privacy Theorem

We start by showing indistinguisability between two experiments which will exactly model our privacy requirement. We separate this part out (i.e., separate it from correctness and efficiency) since it is the most involved part of the proof. The privacy theorem we need is defined below in Theorem A.1,

Theorem A.1. *For any PPT adversary $\mathcal{A}$, Experiment* 0 *and Experiment* 2 *in Figure 1 are indistinguishable for any $T = o(N)$ (except with negligible probability), as long as the underlying Thorp shuffle is adaptive CCA secure with T queries (except with negligible probability) .*

Proof. It is straightforward that Experiment 0 outputs are independent of $(x_i, y_i)_{i\in[T]}$ thus guaranteeing perfect privacy. It is also straightforward that Experiment 2 is equivalent to our construction in Fig. 1. To prove that these two experiments are indistinguishable to any PPT adversary $\mathcal{A}$, we define Experiment 1.

Note that the only difference between Experiments 1 and 2 is that we change all the independently sampled Thorp shuffles to independently sampled perfect random permutations. Thus, if there exists an $\mathcal{A}$ who can distinguish Experiments 1 and 2, we can construct an adversary $\mathcal{A}'$ breaking the T-CCA-security of the Q Thorp shuffles (i.e., distinguishing Q Thorp shuffles from Q random permutations using T adaptive queries to the shuffle and the inverse of the shuffle) as follows: given a T adaptive oracle accesses of each of Q permutations $P_1, \ldots, P_Q$ (and their inverses), either Q Thorp shuffles or Q random permutations:

- Initialize $\mathsf{USED}_i = \varnothing$ for all $i \in [Q]$.
- For $t \in [T]$:
  - Upon receiving (x_t, y_t) from $\mathcal{A}$,
  - Call the oracle to get $j_t \leftarrow P_{x_t}^{-1}(y_t)$
  - Call the oracle to get $o_i \leftarrow P_i(j_t)$ for $i \in [Q] \setminus \{x_t\}$
  - If $o_i \in \mathsf{USED}_i$, sampled o_i uniformly at random from $[K] \setminus \mathsf{USED}_i$ for $i \in [Q] \setminus \{x_t\}$, and also sampled $o_{x_t} \leftarrow_{\$} [K] \setminus \{x_t\}$

```
Experiment 0:

1. Experiment initializes $\mathsf{USED}_i = \varnothing$ for $i \in [Q]$.
2. For $t \in [T]$ :
    (a) Sample $z_t$ uniformly from $[K] \setminus \mathsf{USED}$.
    (b) Adversary outputs $(\cdot, \cdot) \in [Q] \times [K]$.
    (c) Experiment samples $o_i$ uniformly random from $[K] \setminus \mathsf{USED}_i$ for $i \in [Q]$ and adds $o_i$ to $\mathsf{USED}_i$.
    (d) Output $(o_1, \ldots, o_Q)$.
```

```
Experiment 1

1. Experiment samples $Q$ permutations $p_1, \ldots, p_Q$ uniformly from the set of permutations of $[K]$.
2. Experiment initializes $\mathsf{USED}_i = \varnothing$ for $i \in [Q]$.
3. For $t \in [T]$ :
    (a) Adversary outputs $x_t, y_t \in [Q] \times [K]$
    (b) Let $j_t = p_{x_t}^{-1}(y_t)$.
    (c) Let $o_i \leftarrow p_i(j_t)$ for $i \in [Q] \setminus \{x_t\}$.
    (d) If $o_i \in \mathsf{USED}_i$, sampled $o_i$ uniformly at random from $[K] \setminus \mathsf{USED}_i$ for $i \in [Q] \setminus \{x_t\}$, and also sampled $o_{x_t} \leftarrow_{\$} [K] \setminus \{x_t\}$.
    (e) Add $o_i$ to $\mathsf{USED}_i$ for all $i \in [Q]$.
    (f) Output $(o_1, \ldots, o_Q)$.
```

```
Experiment 2

1. Experiment samples $s_1, \ldots, s_Q$ where each $s_i = \mathsf{Th.Gen}(\lambda, N/Q, T)$ ($Q$ Thorp Shuffles).
2. Experiment initializes $\mathsf{USED}_i = \varnothing$ for $i \in [Q]$.
3. For $t \in [T]$ :
    (a) Adversary outputs $x_t, y_t \in [Q] \times [K]$.
    (b) Let $j_t = \mathsf{Th.Inv}(s_{x_t}, y_t)$.
    (c) Let $o_i \leftarrow \mathsf{Th.Eval}(s_i, j_t)$ for $i \in [Q] \setminus \{x_t\}$.
    (d) If $o_i \in \mathsf{USED}_i$, sampled $o_i$ uniformly at random from $[K] \setminus \mathsf{USED}_i$ for $i \in [Q] \setminus \{x_t\}$, and also sampled $o_{x_t} \leftarrow_{\$} [K] \setminus \{x_t\}$.
    (e) Add $o_i$ to $\mathsf{USED}_i$ for all $i \in [Q]$.
    (f) Output $(o_1, \ldots, o_Q)$.
```

Figure 1: Experiments, where Experiment 0 is the ideal world with no privacy loss, and Experiment 2 is our real construction in Algorithm 2. Experiment 1 is a hybrid we need in the proof.

  - Add o_i to USED_i for all $i \in [Q]$.
  - Feed $(o_1, \dots, o_Q)$ to $\mathcal{A}$

- If $\mathcal{A}$ returns Experiment 0, return that $P_1, \dots, P_Q$ are uniformly random permutations; otherwise, return that they are Thorp shuffles.

It is straightforward that if $P_1, \dots, P_Q$ are uniformly random permutations, $\mathcal{A}$ sees exactly Experiment 1; otherwise, it sees exactly Experiment 2. Thus, $\mathcal{A}'$ distinguishes Q Thorp shuffles with Q uniformly random permutations with the same non-negligible probability as $\mathcal{A}$ distinguishes the two experiments, using T queries to each Thorp shuffle.[11] Lastly, note that by the CCA security of Thorp shuffle, one can distinguish a Thorp shuffle from a uniformly random permutation with $\mathsf{negl}(\lambda)$ probability, and since $Q = \mathsf{poly}(\lambda)$, one can also only distinguishes Q Thorp shuffles with Q uniformly random permutations with $Q \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$ probability, which reach a contradiction. Thus, Experiments 1 and 2 are indistinguishable.

Thus, the only thing left to prove is that Experiments 0 and 1 are indistinguishable. To prove this, we use a sequence of hybrids.

- Hyb_1: same as Experiment 0, except that if $t = 1$, do step 3 in Experiment 1 instead of the step 3 in Experiment 0.

- Hyb_2: same as Hyb_1, except that if $t = 2$, do step 3 in Experiment 1 instead of the step 3 in Experiment 0.

- ...

- Hyb_{T-1}: same as Hyb_{T-2}, except that if $t = T - 1$, do step 3 in Experiment 1 instead of the step 3 in Experiment 0.

- Hyb_T: same as Experiment 1.

Experiment 1 and Hyb_1 are indistinguishable due to the following (recall that $t = 1$): since $p_1, \dots, p_Q$ are independently drawn and all are uniformly random permutation, j_1 is a random element in $[K]$ and indistinguishable from z_1 even given (x_1, y_1), and thus $p_i(j_1)$ is indistinguishable from $p_i(z_1)$ for $i \neq x_1$; thus $(o_1, p_2(j_1), \dots, p_Q(j_1))$ is indistinguishable from $(p_1(z_1), \dots, p_Q(z_1))$, as o_1 is sampled uniformly at random from $[K]$ independent of $p_2(j_1), \dots, p_Q(j_q)$.

Then, Hyb_1 and Hyb_2 are indistinguishable as follows:

- if $(x_2, y_2) = (x_1, y_1)$: then every o_i is sampled from $[K] \setminus \mathsf{USED}_i$

- else if $x_2 = x_1$ but $y_2 \neq y_1$: j_2 is indistinguishable from uniformly random from $K \setminus \{j_1\}$, which means that $p_i(x_2)$ for $i \neq x_2$ is uniformly at random from $[K] \setminus \mathsf{USED}_i$, and furthermore o_{x_2} is uniformly at random from $[K] \setminus \mathsf{USED}_{x_2}$ trivially

- else: j_2 is uniformly at random from $[K]$, and thus $p_i(j_2)$ for $i \neq x_t$ is uniformly at random from $[K]$, which means that o_i is uniformly at random from $[K] \setminus \mathsf{USED}_i$ (since if $o_i \in \mathsf{USED}_i$, it is resampled from the rest), for $i \neq x_t$; and again, o_{x_t} is trivially uniformly at random from $[K] \setminus \mathsf{USED}_{x_t}$

[11] Note that here we are using a Thorp shuffle using PRG-generated randomness, which should be indistinguishable from a truly random Thorp shuffle for any PPT adversary.

Inductively, for a similar argument, it is straightforward to see that Hyb_{i-1} and Hyb_i are indistinguishable for $i \in [3, T]$. Lastly, since Hyb_T and Experiment 1 are indistinguishable, and $T = \mathsf{poly}(\lambda)$, we have Experiments 1 and 0 are indistinguishable. Combining the argument that Experiments 1 and 2 are indistinguishable, we conclude that Experiments 0 and 2 are indistinguishable.

□

A.2 A Proof of Theorem 3.1

Below, we then proof of Theorem 3.1.

Proof. **Correctness:** Follows by construction of the scheme. At each step we either retrieve the element by computing the correct result from our stored hints xored with the relevant elements sent to the server during query time, *or*, we retrieve the element locally if it was seen in a previous query request.

Efficiency: Preprocessing takes $O(\lambda N)$ time since the Thorp Shuffle requires λ rounds with N operations. Our scheme saves at the client N/Q hints plus Q new elements for each of the T queries performed, totaling $N/Q + T \cdot Q$ storage. Since the server streams the database to the client at preprocessing, the bandwidth offline is $O(N)$.

Note that evaluating the Thorp Shuffle or its inverse requires $O(\lambda)$ operations, therefore, to execute our query step, which requires taking one inverse Thorp Shuffle and $Q - 1$ normal Thorp Shuffle evaluations. This takes $O(\lambda Q)$ time at the client, which then sends all the Q elements to the server (which indexes them and returns the elements in $O(Q)$ time). The client gets back the Q database values, totaling $Q \log N + Q$ bandwidth. The reconstruct time requires $O(Q)$ time to xor the elements and store them. Furthermore, performing a 'swap' requires an additional, separate datastructure to keep track of the swap since they are not natively supported by the Thorp Shuffle. Since we perform Q swaps in total, this adds an additional $O(Q)$ storage.

Finally, to update element $(q, k) \in [Q] \times [K]$, the server sends q, k, x, x' where x is the old database element and x' is the new element to be updated. The client computes $j \leftarrow \mathsf{Th.Inv}(s_q, k)$, and compute $\mathsf{st}.h_k = \mathsf{st}.h_k \oplus x \oplus x'$. Then, it also computes $\mathsf{st.Used}_q[k] \leftarrow x'$. This together takes one inverse of Thorp shuffle, two xors, and a data structure update (e.g., a hash table), which together takes only $O_\lambda(1)$ time.

Privacy: Privacy, by Definition 2.5 requires defining an algorithm Sim which runs without knowledge of queries and is indistinguishable from real client queries for any PPT adversary $\mathcal{A}$. First, note that in the preprocessing phase, the client simply downloads the whole database to compute its hints so that cannot leak any information about the permutations it samples. Then, by Theorem A.1, we can replace the online phase of scheme with Experiment 0 in Figure 1 and no adversary can distinguish between interacting with our real scheme or Experiment 0 except with negl probability. □

B 2 Server PIR by [35]

In the two-server scheme by [35], Server 0 runs the hint procedure, after which the client, for each query, sends q_0 to Server 0 and q_1 to Server 1, gets back the answer and reconstructs the desired database value. Notice that access to Server 0 allows the client to swap every element that it shows to Server 1 with a random other element in the permutation, this way refreshing the state completely. We provide the scheme in Algorithm 7.

Algorithm 7 The two-server PIR scheme from [35]. Let $Q,N \in \mathbb{N}$ such that $Q|N$. Let $m \in \mathbb{N} = N/Q$. Let DB be an array of N elements of size w. For $i \in [Q]$, let $\mathsf{DB}_i = \mathsf{DB}[i * m : (i+1)m]$.

```
procedure $\textsc{Hint}(\mathsf{DB})$
    Sample $(P_1,\dots,P_Q)$, permutations of $N/Q$ elements, uniformly at random from the set of all permutations.
    Let $h_1,\dots,h_m = 0$.
    for $j$ in $[1,\dots,m]$ do
        Let $h_j = \bigoplus_{i \in Q} P_i(j)$.
    return $(P_1,\dots,P_Q),(h_1,\dots,h_m)$.
procedure $\textsc{Query}(x = (i^*,j^*) \in ([Q] \times [m]), ck)$
    Find $ind$ such that $P_{i^*}(ind) = j^*$.
    Let $S_1 = [P_j(ind) : j \in [Q]]$. Let $S[i^*] = r^* \xleftarrow{\$} [m]$.
    Sample $r_1,\dots,r_Q \xleftarrow{\$} [m]^Q$ .
    Let $S_0 = [P_i(r_i) : i \in [Q]]$.
    For $i \in [Q], i \neq i^*$, swap $P_i(ind)$ and $P_i(r_i)$.
    return Output $(ck, q_0 = S_0, q_1 = S_1)$.
procedure $\textsc{Answer}((q_b = (a_1,\dots,a_Q))$
    return $A_b = [\mathsf{DB}_i(a_i) : i \in [Q]]$.
procedure $\textsc{Reconstruct}((A_0, A_1, ck, \{h_j\}_{j \in [m]})$
    Let $\mathsf{DB}[x] = \mathsf{DB}_{i^*}[j^*] = h_{j^*} \oplus \left(\bigoplus_{i \in [Q], i \neq i^*} A_1[i]\right)$.
    for $i$ in $[1,\dots,Q]$, $i \neq i^*$ do
        Update $h_{ind} = h_{ind} \oplus A_1[i] \oplus A_0[i]$.
        Update $h_{r_i} = h_{r_i} \oplus A_1[i] \oplus A_0[i]$.
    return $(\mathsf{DB}[x], ck, h)$.
```