Random-Index PIR and Applications

Craig Gentry[1], Shai Halevi[1], Bernardo Magri[2,3], Jesper Buus Nielsen[2,3]*, and Sophia Yakoubov[3]**

[1] Algorand Foundation, USA, craigbgentry@gmail.com,shaih@alum.mit.edu
[2] Concordium Blockchain Research Center, Aarhus University, Denmark
[3] Aarhus University, Denmark, {magri,jbn}@cs.au.dk,sophia.yakoubov@gmail.com

Abstract. Private information retrieval (PIR) lets a client retrieve an entry from a database without the server learning which entry was retrieved. Here we study a weaker variant that we call *random-index PIR* (RPIR), where the retrieved index is an *output* rather than an input of the protocol, and is chosen at random. RPIR is clearly weaker than PIR, but it suffices for some interesting applications and may be realized more efficiently than full-blown PIR.

We report here on two lines of work, both tied to RPIR but otherwise largely unrelated. The first line of work studies RPIR as a primitive on its own. Perhaps surprisingly, we show that RPIR is in fact equivalent to PIR when there are no restrictions on the number of communication rounds. On the other hand, RPIR can be implemented in a "noninteractive" setting (with pre-processing), which is clearly impossible for PIR. For two-server RPIR we even show a truly noninteractive solution, offering information-theoretic security without any pre-processing.

The other line of work, which was the original motivation for our work, uses RPIR to improve on the recent work of Benhamouda *et al.* (TCC'20) for maintaining secret values on public blockchains. Their solution depends on a method for selecting many random public keys from a PKI while hiding most of the selected keys from an adversary. However, the method they proposed is vulnerable to a double-dipping attack, limiting its resilience. Here we observe that a RPIR protocol, where the client is implemented via secure MPC, can eliminate that vulnerability. We thus get a secrets-on-blockchain protocol (and more generally large-scale MPC) which is resilient to any fraction $f < 1/2$ of corrupted parties, resolving the main open problem left from the work of Benhamouda *et al.*

As the client in this solution is implemented via secure MPC, it really brings home the need to make it as efficient as possible. We thus strive to explore whatever efficiency gains we can get by using RPIR rather than PIR. We achieve more gains by using *batch RPIR* where multiple indexes

* Partially funded by The Concordium Foundation; The Danish Independent Research Council under Grant-ID DFF-8021-00366B (BETHE); The Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM).

** Funded by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO).

are retrieved at once. Lastly, we observe that this application can make do with a weaker security guarantee than full RPIR, and show that this weaker variant can be realized even more efficiently. We discuss one protocol in particular that may be attractive for practical implementations.

Keywords. Private information retrieval, Batch PIR, Random PIR, Large-scale MPC, Secrets on blockchain, Random ORAM.

1 Introduction

A Private Information Retrieval (PIR) scheme lets a client fetch an entry from a database held by a server, without the server learning which entry was retrieved. The database is typically modelled as an n-bit string $DB \in \{0,1\}^n$, known in full to the server. The client has an input index $i \in [n]$, and its goal is to retrieve the bit $DB[i]$. A PIR scheme is secure if the server cannot distinguish between any two possible input indexes i, i' for the client, and it is *nontrivial* if the server sends to the client less than n bits. PIR was introduced by Chor et al. [6] who described a solution with multiple non-colluding servers; a single-server solution was first described by Kushilevitz and Ostrovsky [14].

1.1 Random-Index PIR (RPIR)

In this work we consider a similar setting, but with a twist. Rather than a specific index, in our setting the client wishes to retrieve *a random index* from the database, without the server learning which index was retrieved. Namely, instead of the index i being an input to the protocol, we consider it an output, and require that it be random. We call such a scheme random-index PIR (RPIR). While clearly a weaker variant of PIR, we show below that RPIR suffices for some interesting applications. Of course, RPIR can be easily implemented by having the client choose i at random and then run a PIR protocol. But being a weaker variant, we could hope that RPIR is easier and more efficient to implement than full blown PIR. Such improved efficiency could be critical for some applications, including our motivating application of large-scale secure MPC (which is described below).

One measure of efficiency is the number of communication rounds. We show that, unlike PIR, RPIR can be implemented in a "noninteractive" fashion. Namely, after a pre-processing stage in which the client sends to the server some string whose length depends only on the security parameter κ, we only allow server-to-client communication and we want to perform arbitrarily many RPIR executions. It is clear that no such nontrivial PIR protocols exist, since there is no way for such protocols to incorporate the client's input. However, we show that some existing interactive PIR protocols can be adapted to yield noninteractive RPIR protocols. (These can be based on fully homomorphic encryption or one-way trapdoor permutations.) Moreover, for the two-server setting we show that a nontrivial noninteractive protocol is possible even without any pre-processing.

Other examples of settings where RPIR is more efficient than PIR are discussed in Section 1.3 below.

On the other hand, we show that such efficiency gains are necessarily limited, since every RPIR protocol can be converted into a PIR protocol with only slightly more communication and rounds. Specifically, given a r-round RPIR protocol with server communication $m < n$, we show how to construct:

- A $((r+1)\log n)$-round PIR with server communication $1 + m \log n$; or
- A $(r+2)$-round PIR with server communication $O(\sqrt{mn})$.

We note that the latter transformation relies on a long client-to-server message. We also describe a simple variant with a short client-to-server message, where the server communication is $m + \frac{n}{2}$.

1.2 Applications

Computing on Public Blockchains Our initial motivation for studying RPIR came from a recent work of Benhamouda *et al.* [2] (BGG+) about maintaining secret values on public blockchains. In that work they construct a scalable evolving-committee proactive secret-sharing (ECPSS) scheme, that allows dynamically-changing small committees to maintain a secret over a public blockchain. The main challenge in that work was to choose a small committee from within a large population in such a way that (a) everyone can send messages to committee members, and yet (b) a mobile adversary does not learn who they are and therefore cannot target them for corruption. Once chosen, such committees can execute the proactive secret sharing protocol (or more generally any secure-MPC protocol).

A drawback of the BGG+ scheme is that, in order to guarantee an honest majority within the committees, it can only tolerate up to about 1/4 corruptions overall. The reason is that committee-selection is done by individual parties, who "nominate" members to the new committee by drawing their public keys from a list and then re-randomizing them. This nomination style enables a double-dipping adversarial strategy: corrupted parties can always nominate other corrupted parties, while honest parties nominate randomly selected parties (so they too sometimes nominate corrupted parties by chance).

To do better, we can try to delegate the nomination task to previous committees, who would emulate an honest nominator via secure MPC. Roughly, the function computed by the committee-selection procedure of [2] is

$$\mathsf{Nominate}(n \text{ public keys, randomness}) = k \text{ re-randomized keys.}$$

We can let previous committees compute that randomized function, without the adversary learning anything about who the honest nominees are, hence depriving it of the double-dipping strategy above. The problem with this solution, however, is that it scales poorly with the total number of parties: The circuit of the $\mathsf{Nominate}$ function above has input of size linear in n, hence a naive secure-MPC protocol for it would have complexity more than n.

This is where RPIR comes in. The only role that the input plays in the Nominate function is of a database from which we choose $k \ll n$ random entries. We therefore employ a variant of MPC-in-the-head, letting previous committees play the role of the RPIR client while each committee member individually mentally plays the role of the RPIR server. (The database is the list of n public keys, which is known to everyone; so, the state of the RPIR server is public.)

The result of the RPIR protocol is the previous committee holding a set of k random keys, but since we have honest-majority in the committee then the adversary does not know whose keys were chosen. The committee then runs a secure-MPC protocol to re-randomize the chosen keys and output the result. This time, the circuit size depends on the complexity of the RPIR client. For some RPIR constructions, this depends only on k, not on the total number n of keys. Putting all these ideas together we get:

Theorem (informal): In the model of [2], there exists a scalable ECPSS scheme tolerating any fraction $f < 1/2$ of corrupted parties.

Of course, once we have the committees we can again let them compute on secrets rather than just pass them along, hence we have:

Theorem (informal): In the model of [2], there exists a scalable secure MPC scheme tolerating any fraction $f < 1/2$ of corrupted parties.

PIR with Preprocessing In many applications it is interesting to consider offline preprocessing before the inputs are known, which can help improve the efficiency of the on-line computation once all the inputs are available. This approach is very popular in contemporary secure-MPC work, and was also used for PIR (e.g., [1,8]).

As it turns out, our PIR-to-RPIR reductions from Section 2.4 can be used for that purpose. These reductions have the following format: They first run the underlying RPIR protocol on the original database DB, letting the client learn a few random bits from it. The client then sends a single message to the server, from which the server computes a new database DB' of size $n' < n$. The parties then run a PIR protocol on the new (smaller) database, and the client uses what it learns to compute the bit that it needs from the original DB.

This format makes it possible to run the RPIR protocol in a pre-processing phase, before the client knowns what index it wants, and only execute the last part during the online phase. Using a standard PIR to implement the RPIR in the pre-processing step, we obtain a black-box method of shifting work from the online to the offline phase of a PIR protocol. If $CC(n, \kappa)$ is the server communication complexity of an underlying PIR protocol (as a function of the database size n and the security parameter κ), the online server communication complexity of the resulting protocol with preprocessing will be only $CC(n', \kappa)$. Specifically:

- Using the SimplePIR protocol from Section 2.4, we obtain a PIR-with-Preprocessing protocol with offline communication $CC(n, \kappa)$, online communication $CC(n/2, \kappa)$, and the client sending one more message of $\log n$ bits.

– Using the PartitionPIR protocol from Section 2.4, we get for any $t < n$ a PIR-with-Preprocessing protocol with offline communication $t \cdot CC(n, \kappa)$, online communication $CC(O(n/t), \kappa)$, and the client sending one more long message (of more than n bits).

1.3 Batch RPIR

In our first motivating application in Section 1.2, the client needs to fetch not one but k random entries from the database, so we would like to amortize the work and implement it in complexity less than that of k independent RPIR executions. Building such batch PIR protocols from PIR was studied by Ishai, Kushilevitz, Ostrovsky, and Sahai (IKOS) [13]. However, their solutions require the underlying protocol to be a full-blown PIR protocol (rather than RPIR). It is not clear how to build batch-RPIR protocols from an underlying RPIR protocol any better than either running k independent instances of RPIR, or converting to full-blown PIR and using the IKOS solutions.

But it turns out that our motivating application can make do with a weaker security notion than what RPIR provides. What we care about in this application is not quite that the indexes look random to the server, but rather that a server with limited "corruption budget" in the entire population cannot corrupt too many of the selected indexes (whp). Roughly, we can replace the pseudorandomness of the indexes from the server's perspective by unpredictability. Defining this property takes some care. In Section 5.1 we provide a definition in the real/ideal style.

Having lowered the security bar, we take another look at the constructions from [13] and note that we can use better parameters than are possible for batch-PIR (or batch-RPIR with strong security). Moreover, we describe in Section 5.2 an even simpler construction that cannot possibly work for batch PIR or strong-RPIR, but we prove that it meets our weaker security notion of batch RPIR. The simplicity and efficiency of this construction may be attractive for practical implementations.

1.4 Multi-Server RPIR

It is known that nontrivial single-server PIR cannot offer information-theoretic privacy; nontrivial single-server RPIR has the same limitation. It is interesting to ask whether by involving multiple non-colluding servers (each with the same database as input) we can build RPIR that is (a) nontrivial, (b) information-theoretic and (c) noninteractive (meaning that only a single round of communication — from each server to the client — is required). We answer this question in the affirmative; we show a two-server nontrivial, information-theoretic non-interactive RPIR with communication complexity equal to half the size of the database.

While it seems that multi-server RPIR cannot be used directly in the application of secure computation on public blockchains, it can be used for PIR pre-processing (either for a multi-server PIR execution with the same servers that participated in the pre-processing, or perhaps even for a single-server PIR execution with only one of those servers).

1.5 Organization

In Section 2 we formally define (single-server and multi-server) RPIR, and study the relationship between RPIR and PIR. In Section 3 we describe some constructions of RPIR in the noninteractive setting. In Section 4 we describe the application of batch RPIR with weak security to the architecture of Benhamouda *et al.* [2] for large-scale MPC. Motivated by this application, we study in Section 5 more efficient constructions of batch-RPIR.

In Appendix A we describe the notion of a *random-index oblivious-RAM* (RORAM), which relates to ORAM in the same way that RPIR relates to PIR. In particular we show that RORAM can replace RPIR in the same context of large-scale MPC, offering a somewhat different performance profile. For completeness, in Appendix B we discuss a third approach for the large-scale MPC context that uses mix-nets.

2 Random-Index Private Information Retrieval

2.1 Background: Private Information Retrieval

A single-server Private Information Retrieval (PIR) scheme is a two-party protocol $\mathit{\Pi}$ between a server holding a n-bit string $DB \in \{0,1\}^n$ and a client holding an index $i \in [n]$. In addition, both parties know the security parameter κ.

We assume for simplicity that the server communication complexity, i.e. the number of bits sent by the server, depends only on n and κ, but not on the specific values of DB and i (or the protocol randomness), and denote it by $CC_{\mathit{\Pi}}(n,\kappa)$. The two properties of interest for a PIR protocol $\mathit{\Pi}$ are its client-privacy (i.e. the index i is hidden from the server) and its communication complexity.

Definition 1 (Single-server PIR [14]). *A two-party protocol $\mathit{\Pi}$ is a (semi-honest) single-server PIR if it satisfies:*

Correctness. *The client's output is $DB[i]$, except with probability negligible in κ.*

Client privacy. *For every n, database $DB \in \{0,1\}^n$, and indexes $i, i' \in [n]$, the ensembles $\mathsf{serverView}(\mathit{\Pi}(\kappa, n;\ i, DB))_\kappa$ and $\mathsf{serverView}(\mathit{\Pi}(\kappa, n;\ i', DB))_\kappa$ are indistinguishable.*

Nontriviality. *For any κ and large enough n, it holds that $CC_{\mathit{\Pi}}(n,\kappa) < n$.*

Batch PIR. In this work we are also interested in amortized protocols in which the client queries more than a single entry of the database at a time, but rather k indexes at a time. The definition of batch PIR is identical to the above, except that the single index $i \in [n]$ is replaced with a vector $\vec{i} \in [n]^k$. Everything else remains the same.

Multi-Server PIR. We additionally explore protocols involving multiple non-colluding servers. The definition of multi-server PIR is similar to the above, except that client privacy is defined with respect to each of the servers (individually).

Ideal functionality. A different approach for defining PIR is via an ideal functionality that gives no output to the server and outputs $DB[i]$ to an honest client.[4] We will use that style of definition for random-PIR below, as it seems easier to work with than the one above, especially for the weaker-security variant from Section 5.1.

Nevertheless, defining security of PIR using an ideal functionality can sometimes be problematic. If the server is maliciously corrupted, then the simulator needs an extra input, the database, from the malicious server. However, the server communication is too small to even *define* the database, so this extraction cannot be performed. In this work we only consider semi-honest corruptions of the server, thus we do not run into this problem.

2.2 Defining RPIR

A random-index PIR (RPIR) protocol is different from PIR in that the index i is an output of the client, rather than an input. Namely, RPIR is a two-party protocol between a server holding a n-entry database $DB \in \{0,1\}^n$ and a client with no input. At the conclusion of the protocol, the client is supposed to get a pair $(i, DB[i])$, with i random in $[n]$.

Just like standard PIR, an RPIR protocol is parametrized by the security parameter κ and the database size n, both known to the two parties. As above, we assume that the server communication complexity depends only on n and κ but not on the specific value of DB or the randomness, and we denote it by $CC_{\Pi}(n, \kappa)$.

It will later (when we define batch RPIR) be convenient to define client-privacy by means of an "ideal RPIR functionality", i.e., via simulation-based security. We give both simulation-based and game-based definitions of RPIR below, and show that the two definitions are equivalent. Later we only use the simulation-based style.

The RPIR functionality. The functionality $\mathcal{F}_{\mathsf{RPIR}}$ accepts from the server an input $DB \in \{0,1\}^*$. It leaks DB to the adversary, and waits for the client to ask for an output. When the client does, $\mathcal{F}_{\mathsf{RPIR}}$ sets $n = |DB|$, chooses $i \leftarrow [n]$ uniformly at random, and returns $(i, DB[i])$ to the client (when the adversary says it's time to give output).

Definition 2 (Single-server RPIR (simulation-based)). *A two-party protocol Π is a single-server RPIR if it realizes the functionality $\mathcal{F}_{\mathsf{RPIR}}$ above with*

[4] Note that standard PIR does not provide any privacy to the server, hence the functionality lets a corrupted client get the entire database.

semi-honest server and honest client in the UC framework [4]. It is nontrivial if for any κ and large enough n if it holds that $CC_{\Pi}(n, \kappa) < n$.

Remarks on the formalization. When defining RPIR we leak the entire database to the adversary. This models that the database is not required to be kept secret from the client or anyone in the network. (We only *require* that the client learns $DB[i]$; we do not give the database as output to the client as this would turn it into an implementation requirement.) Since we give the database to the adversary, the protocol is allowed to leak information on DB to the adversary even if the server and client are both honest. So, the protocol can run over an authenticated but unencrypted channel. One could have chosen a more complicated formulation where nothing is leaked to the adversary when both server and client are honest; this complicates the formulation and is not needed in our setting, so we opted for the simpler formulation.

It is important to consider a corrupt (semi-honest) server in order to ensure that such a server cannot learn i. However, we choose not to consider a corrupt (semi-honest) client, since our main application (where the client is run within an MPC, and thus honest) does not require this. Considering definitions for corrupt clients, as well as considering malicious corruptions, is left for future work.

Next, for clarity, we look at the relationship of the simulation-based definition given above to a game-based definition. We can adopt the definition of [14] to the case of a random index as follows.

Definition 3 (Single-server RPIR (game-based)). *A two-party protocol Π is a (semi-honest) single-server RPIR if it satisfies:*

Correctness. *The client's output is $(i, DB[i])$, where i is statistically close to uniform on $[n]$.*

Client privacy. *For every n, database $DB \in \{0,1\}^n$ and a run of the protocol, let i_{REAL} be the index output to the client and let i_{RAN} be a fresh uniform value from $[n]$. We require that the ensembles $(\mathsf{serverView}(\Pi(\kappa, n; DB))_{\kappa}, i_{\mathrm{REAL}})$ and $(\mathsf{serverView}(\Pi(\kappa, n; DB))_{\kappa}, i_{\mathrm{RAN}})$ are indistinguishable.*

Nontriviality. *For any κ and large enough n, it holds that $CC_{\Pi}(n, \kappa) < n$.*

In the statement of correctness, we ask that i is statistically close to uniform. We could have asked for only computational indistinguishability, but these are equivalent when i is on a domain of polynomial size.

Lemma 1. *Let Π be a two-party protocol. If it is an RPIR according to Definition 3 then it is an RPIR according to Definition 2.*

Proof (sketch). Assume the protocol Π fulfills the game-based definition. We construct a simulator for the simulation-based definition. Assume without loss of generality an environment $\mathcal{E}$ and a dummy adversary. This means the simulator is *de facto* interacting with $\mathcal{E}$.

Our simulator engages with $\mathcal{E}$ in an honest execution $\Pi(\kappa, n; DB)$, where the simulator plays the role of client. As a result of the interaction, $\mathcal{E}$ gets

$\mathsf{serverView}(\mathit{\Pi}(\kappa, n; DB))$; the simulator gets i_{REAL}. When that happens, the simulator will instruct $\mathcal{F}_{\mathsf{RPIR}}$ to deliver output to the client. The effect of this is that $\mathcal{E}$ learns a uniformly random i_{RAN} from $\mathcal{F}_{\mathsf{RPIR}}$ as client output. (It also learns $DB[i_{\text{RAN}}]$, but it can compute this value from i_{RAN} anyways, since it knows DB.) So, the view of $\mathcal{E}$ is $(\mathsf{serverView}(\mathit{\Pi}(\kappa, n; DB)), i_{\text{RAN}})$.

Consider now a hybrid world with a hacked $\mathcal{F}_{\mathsf{RPIR}}$, where we allow the simulator to give i_{REAL} to $\mathcal{F}_{\mathsf{RPIR}}$ when it instructs delivery, and then $\mathcal{F}_{\mathsf{RPIR}}$ gives i_{REAL} as output to the client instead of a fresh random value. This means that the view of $\mathcal{E}$ becomes $(\mathsf{serverView}(\mathit{\Pi}(\kappa, n; DB)), i_{\text{REAL}})$. We can use a reduction to client privacy to prove that the simulation and the hybrid are indistinguishable. Notice that the view of $\mathcal{E}$ in the hybrid has the same distribution as the the view of $\mathcal{E}$ in a real protocol execution.

Lemma 2. *Let $\mathit{\Pi}$ be a two-party protocol. If it is an RPIR according to Definition 2 then it is an RPIR according to Definition 3.*

This is clear, since any distinguisher that can break client privacy can be used to build an environment that can distinguish between the client output in a real execution and the ideal client output that it sees when interacting with the simulator.

2.3 Defining Multi-Server RPIR

We also consider a multi-server version of RPIR. An ℓ-server RPIR protocol involves ℓ servers $\mathsf{Server}_1, \ldots, \mathsf{Server}_\ell$ each holding the same database $DB \in \{0,1\}^n$, and a client who wants to retrieve a random index i of the database. Multi-server RPIR is interesting since, while nontrivial single-server RPIR cannot provide information-theoretic privacy, nontrivial multi-server RPIR can. We therefore require *perfect* correctness and client-privacy for multi-server RPIR. Since we do not extend multi-server RPIR to the batch setting, we use the simple definitions of multi-server RPIR that are analogous to those for PIR (Section 2.1).

Definition 4 (Multi-server RPIR). *An $(\ell + 1)$-party protocol $\mathit{\Pi}$ is a (semi-honest) ℓ-server RPIR if it satisfies:*

Correctness. *For every n, every database $DB \in \{0,1\}^n$, and every index $i \in [n]$, the client's output in $\mathit{\Pi}(n;\ \perp, DB, \ldots, DB)$ is $(i, DB[i])$ with probability $\frac{1}{n}$.*

Client privacy. *For every n, every database $DB \in \{0,1\}^n$, and every server index $j \in [\ell]$, the view $\mathsf{serverView}_j(\mathit{\Pi}(n;\ \perp, DB, \ldots, DB))_\kappa$ is independent of the index i that the client outputs.*

Nontriviality. *For any κ and large enough n, it holds that $CC_{\mathit{\Pi}}(n, \kappa) < n$ (where the $CC_{\mathit{\Pi}}(n, \kappa)$ is communication complexity of* all *the servers).*

2.4 RPIR is equivalent to PIR

In terms of existence, it is obvious that PIR implies RPIR: the client chooses a random index $i \in [n]$ and the parties then run a PIR protocol in which the client learns $DB[i]$. The opposite direction is less clear: how can the client get a specific index in the database using the RPIR tool that only provides random indexes? Below we show, however, that RPIR *does imply* PIR with very small overhead. We begin with a simple PIR protocol that works when n is a power of two, makes a single RPIR call, and has the server send $n/2$ additional bits. This protocol is described in Figure 1.

```
SimplePIR[Client($i \in [n]$), Server($DB \in \{0,1\}^n$)]   ($n$ is a power of two)

1. Server and client run RPIR[Client, Server($DB$)], client gets $(j, DB[j])$;
2. Client sends to server $\delta = i \oplus j$ ($i, j$ are viewed as $\log(n)$-bit strings)
3. Server partitions the index-set $[n]$ into $n/2$ pairs $p = \{k, k \oplus \delta\}$, computes for
   each pair $\sigma_p = DB[k] \oplus DB[k \oplus \delta]$, and sends these $n/2$ bits to the client;
4. Client computes $DB[i] = DB[j] \oplus \sigma_{\{i,j\}}$.
```

Fig. 1. A simple PIR protocol with one RPIR call and $n/2$ bits of communication

Lemma 3. *For n a power of two, the SimplePIR protocol from Figure 1 is a nontrivial PIR protocol in the hybrid-RPIR model in which the client sends $\log n$ bits and the server sends $n/2$ bits.*

Proof. Correctness and complexity are obvious. For client privacy, note that in the hybrid-RPIR model the client gets a uniformly random index $j \in [n]$, and since n is a power of two then j is also a uniformly random $\log(n)$-bit string. Hence from the server's perspective, the message $\delta = i \oplus j$ from the client is also a uniformly random $\log(n)$-bit string, and in particular it carries no information about the client's input i.

Next, we note that Steps 3-4 in the SimplePIR protocol actually implement the trivial PIR protocol for a database of size $n/2$: The server sends all the $n/2$ bits and the client looks up the one that it needs. We can do better by replacing these steps with a recursive call for the same PIR protocol on this smaller database, as described in Figure 2.

Theorem 1. *An r-round RPIR with server-communication $m = m(n, \kappa)$ and client-communication $k = k(n, \kappa)$ can be transformed into a PIR protocol with $(r+1)\lceil \log n \rceil$ rounds, server communication $1 + \sum_{i=1}^{\lceil \log n \rceil} m(2^i, \kappa) \le 1 + m(n, \kappa) \cdot \lceil \log n \rceil$, and client communication $\sum_{i=1}^{\lceil \log n \rceil} i + k(2^i, \kappa) \le \binom{\lceil \log(n) \rceil}{2} + k(n, \kappa) \cdot \lceil \log n \rceil$.*

Proof (sketch). On a size-n database, the server pads it to size the nearest power of two and then the parties run the RecursivePIR protocol from Figure 2. The complexity is obvious, and correctness and privacy are argued by induction, following the same proof as for Lemma 3.

```
$\mathsf{RecursivePIR}\big[\text{Client}(i \in [n]),\ \text{Server}(DB \in \{0,1\}^n)\big]$ ($n$ is a power of two)

0. If $n = 1$ the server sends $DB$ to the client. Else continue to Step 1.
1. The server and client run $\text{RPIR}\big[\text{Client, Server}(DB)\big]$, client gets $(j, DB[j])$
2. Client sends to server $\delta = i \oplus j$ ($i, j$ are viewed as $\log(n)$-bit strings)
3. Server partitions the index-set $[n]$ into $n/2$ pairs $p = \{k, k \oplus \delta\}$ and computes
   for each pair the bit $\sigma_p = DB[k] \oplus DB[k \oplus \delta]$.
4. Let $DB' = (\sigma_p)_p$ be the resulting database of size $n/2$, and let $i' \in [n/2]$ be
   the index corresponding to the pair $\{i, j\}$ in this database.
   The parties run $\mathsf{RecursivePIR}\big[\text{Client}(i'),\ \text{Server}(DB')\big]$, client gets $\sigma_{i'}$.
5. Client outputs $DB[i] = DB[j] \oplus \sigma_{i'}$.
```

Fig. 2. A recursive PIR protocol with $\log n$ calls to RPIR and one bit of communication

PIR from RPIR with Fewer Rounds While the protocol in Figure 2 has a low communication complexity, it has a large number of rounds. Below we describe instead a protocol that has the same number of rounds as the SimplePIR protocol from Figure 1, but lower server communication complexity. The basic idea is for the client to learn more random indexes $DB[j]$, then partition the bits in DB into larger sets instead of the pairs $\{i, i \oplus \delta\}$ from SimplePIR. Specifically, we have a parameter t that tells us how large should these groups be.

```
$\mathsf{PartitionPIR}\big[\text{Client}(i \in [n]),\ \text{Server}(DB \in \{0,1\}^n)\big]$ ($n$ is divisible by $t$)

1. Server and client run in parallel $t'$ executions $\text{RPIR}\big[\text{Client, Server}(DB)\big]$,
   where $t'$ is large enough to ensure that the client gets whp at least $t - 1$
   distinct entries $(j_1, DB[j_1]), \ldots, (j_{t-1}, DB[j_{t-1}])$, all different from $i$.
2. Client chooses a random partition $\mathcal{P}$ of $[n]$ into sets of size $t$, with one of them
   being $I = \{i, j_1, j_2, \ldots, j_{t-1}\}$, and sends $\mathcal{P}$ to server.
3. For each $t$-subset $J \in \mathcal{P}$, the server computes the bit $\sigma_J = \oplus_{j \in J} DB[j]$, and
   sends these $n/t$ bits to the client.
4. Client computes $DB[i] = DB[j_1] \oplus \cdots \oplus DB[j_{t-1}] \oplus \sigma_I$.
```

Fig. 3. A partition-based PIR protocol

Exactly the same proof as Lemma 3 shows that this is a secure PIR protocol in the RPIR-hybrid model, with t' executions of the RPIR protocol all on the same database DB, and additional server communication of n/t bits. If we have a r-round RPIR protocol with server communication $m = m(n, \kappa) < n/2$, we can set $t \approx \sqrt{n/m}$ and $t' = t(1 + o(1))$, and then we would get a $(r + 2)$-round PIR protocol with server communication $t'm + n/t = (1 + o(1))\sqrt{nm} + \sqrt{mn} \approx 2\sqrt{nm}$.

Theorem 2. *Given a r-round RPIR protocol with server-communication m, there is a PIR protocol with $r + 2$ rounds and server communication $O(\sqrt{mn})$.*

□

We note that the client communication in the protocol is large, since describing a random partition of $[n]$ into t-subsets takes more than n bits. Finding a protocol with few rounds and small client communication is an open problem.

3 RPIR Protocols

3.1 Noninteractive RPIR

While equivalent in terms of existence, RPIR can still be cheaper to implement than PIR by some measures. In particular, the fact that the client has no input in RPIR means that it can be (almost) *noninteractive*, something that is obviously impossible for PIR. Many interactive PIR protocols can be converted to noninteractive RPIR protocols; below we sketch two such protocols. One is based on FHE, and the other on trapdoor permutations (similar to Kushilevitz-Ostrovsky [15]). We suspect that many other PIR protocols can be similarly modified to obtain noninteractive RPIR; we leave it to future work to explore whether those other protocols can give better parameters.

In both of the protocols described in this section, the client sends a short "pre-processing message" to the server, and then the server can succinctly send to the client arbitrarily many random entries from the database, without learning what they are and without any more messages from the client. (These protocols can be upgraded to handle a malicious server by adding succinct proofs of correct behavior; however, we only need semi-honest security for our primary application, described in Section 4.)

Noninteractive RPIR from FHE. It is fairly easy to implement noninteractive RPIR from FHE. For example, the client sends to the server "once and for all" an encryption of a seed s for a PRF $f_s(\cdot)$ with range $[n]$. Then the server can run many instances of a protocol, where it chooses a random x, and homomorphically computes $i = f_s(x)$ and $y = DB[i]$. The server sends the ciphertexts encrypting (i, y) to the client, who can decrypt them.

Noninteractive RPIR from One-way Trapdoor Permutations. This construction is based on the Kushilevitz-Ostrovsky PIR protocol from [15]. In this protocol the client sends the description of a permutation to the server, and then the server can send as many random indexes to the client as we want. As in the original Kushilevitz-Ostrovsky protocol, each random index costs just a little less than n bits of communication for an n-bit database.

Background: UOWHFs from one-way permutations. Recall that Naor and Yung described in [16] a construction for 2-to-1 universal one-way hash functions (UOWHF) based on one-way permutations. Namely, given a one-way permutation π over $\{0,1\}^k$ (and some other public randomness that we ignore here) they define a 2-to-1 function $h_\pi : \{0,1\}^k \to \{0,1\}^{k-1}$, such that given π and a random $x \in \{0,1\}^k$, it is hard to find the second pre-image $x' \neq x$ such that

$h_\pi(x') = h_\pi(x)$. However given a trapdoor π^{-1}, it is easy to compute the two pre-images of any $y \in \{0,1\}^{k-1}$. Finally, applying the Goldreich-Levin hard-core predicate [12], we also know that given the permutation π and random $x, r \in \{0,1\}^k$, the inner product $\langle r, x' \rangle \bmod 2$ is pseudorandom, where x' is the second pre-image of $h_\pi(x)$.

A noninteractive variant of the Kushilevitz-Ostrovsky construction. In a preprocessing phase, the client chooses a one-way permutation π over $\{0,1\}^k$ together with its trapdoor π^{-1}, and sends π to the server. Let $h_\pi(x)$ be a Naor-Yung UOWHF based on π, that has input length k and output length $k-1$.

The server partitions the database into pairs of k-bit blocks (x_i^0, x_i^1), $i = 1, 2, \ldots$. For simplicity, we assume below that $x_i^0 \neq x_i^1$ for all i (we mention at the end how to change the protocol when this is not the case). The server also chooses a random $r \in \{0,1\}^k$ that defines a Goldreich-Levin hard-core predicate [12] $\rho_r(x) = \langle x, r \rangle \bmod 2$. The server sends to the client the k-bit string r, and also for each pair (x_i^0, x_i^1) it sends a tuple

$$\big(h_\pi(x_i^0), h_\pi(x_i^1), \rho_r(x_i^0) \oplus \rho_r(x_i^1)\big).$$

Note that each tuple is only $(2k-1)$-bits long, whereas the pair itself has $2k$ bits, so this is a nontrivial protocol (as long as there are more than k pairs).

For each received tuple (y_i^0, y_i^1, σ_i), the client uses its trapdoor to invert the hash function, computing the two possible pre-images $u_i^0, v_i^0 \in h_\pi^{-1}(y_i^0)$ and $u_i^1, v_i^1 \in h_\pi^{-1}(y_i^1)$. By construction, $x_i^0 = u_i^0$ or $x_i^0 = v_i^0$ and similarly $x_i^1 = u_i^1$ or $x_i^1 = v_i^1$. Next, the client finds an index i such that,

(a) either $\rho_r(u_i^0) = \rho_r(v_i^0)$ and $\rho_r(u_i^1) \neq \rho_r(v_i^1)$, or
(b) $\rho_r(u_i^0) \neq \rho_r(v_i^0)$ and $\rho_r(u_i^1) = \rho_r(v_i^1)$.

As r was chosen at random and $x_i^0 \neq x_i^1$ for all i, there is at least one such index whp. If there are more than one then the client chooses one of them at random. Moreover it can be shown that the index used by the client is uniform in $[n]$.

In case (a) the client knows that $\rho_r(x_i^0) = \rho_r(u_i^0) = \rho_r(v_i^0)$, and so it can use $\sigma = \rho_r(x_i^0) \oplus \rho_r(x_i^1)$ to determine the value of $\rho_r(x_i^1)$, and therefore decide whether $x_i^1 = u_i^1$ or $x_i^1 = v_i^1$. Similarly in case (b) the client knows that $\rho_r(x_i^1) = \rho_r(u_i^1) = \rho_r(v_i^1)$, so it can use σ to decide if $x_i^0 = u_i^0$ or $x_i^0 = v_i^0$. In either case, the client learns a single k-bit block of the database.

The security of this protocol follows from the OWUHF property and the Goldreich-Levin hard-core predicate, in exactly the same way as in [15].

Theorem 3. *If trapdoor one-way permutations exist, then there exists a nontrivial noninteractive random-PIR protocol.* □

Remark: To deal with generic databases where we could have $x_i^0 = x_i^1$ for some i, the server can choose another k-bit string $w \in \{0,1\}^n$ which is also sent to the client, and use ${x'}_i^1 = x_i^1 \oplus w$ instead of x_i^1 for all i. This ensures that $x_i^0 \neq {x'}_i^1$ except with exponentially small probability, and the client can mask-out w at the end of the protocol if needed.

3.2 Multi-Server RPIR Protocols

It is well known that nontrivial single-server PIR cannot offer information-theoretic security, and RPIR is no different. To get nontrivial information-theoretic security we need to look at multi-server solutions, where two or more non-colluding servers are used. In this section, we describe two non-interactive multi-server solutions. The first one (Section 3.2) is an information-theoretic two-server RPIR; the second one (Section 3.2) uses some symmetric cryptography, and is based on Reed-Muller PIR.

Note that converting existing multi-server PIR schemes to noninteractive multi-server RPIR schemes is only possible under very specific conditions. Multi-server PIR schemes always have a message from the client to the servers, to encode the query; to convert multi-server PIR to noninteractive multi-server RPIR, we need the servers to generate these messages locally on their own. However, in many PIR schemes (such as the Reed-Muller and matching-vector PIR schemes), these messages need to be highly correlated, making local generation by the servers (instead of coordinated generation by the client) difficult. In Section 3.2, we describe a construction where the servers don't need any correlated messages; in Section 3.2, we adapt the Reed-Muller construction — which does require correlated messages — by giving the servers some correlated information as setup, which they are able to expand to support an arbitrary number of RPIR executions. (Note that not all forms of correlated messages are amenable to this kind of expansion from efficient cryptographic primitives. The randomness required for the matching vectors PIR construction does not appear to be, given the state of the art in non-interactive randomness expansion.)

Non-Interactive Information-Theoretic Two-Server RPIR In Figure 4 below we describe a nontrivial two-server solution that offers information-theoretic security and in addition is completely noninteractive. Differently than the protocols from Section 3.1, this protocol does not even have a pre-processing phase. All it has are two messages, one from each server, from which the client can deduce $DB[i]$ for a random index i, with i independent of the view of each server (separately). In this protocol, one server sends a single database record, while the other sends $n/2$ values each of which correspond to the XOR of two database records. The client is able to use the record sent by the first server to recover another record from one of the values sent by the second server. (Reducing the communication complexity in this noninteractive multi-server setting below $n/2$ for a a size-n database remains an interesting open problem.)

Lemma 4. *For even n, the SimpleMSPIR protocol from Figure 4 is a noninteractive, nontrivial two-server RPIR protocol with information theoretic security in which the servers send $n/2 + \log(n) + 1$ bits.*

Proof. Correctness and complexity are obvious. For client privacy, we separately consider privacy against Server$_1$ and Server$_2$. Server$_1$, who chooses j, learns nothing about i since the random and uniform δ is unknown to Server$_1$, and

```
SimpleMSPIR[Client, $\text{Server}_1(DB), \text{Server}_2(DB)$ where $DB \in \{0,1\}^n$)]
  1. $\text{Server}_1$ chooses a random index $j \in [n]$ and sends $DB[j]$ to Client.
  2. $\text{Server}_2$ chooses a random mask $\delta \in [n]$ (viewed as a $\log(n)$-bit string).
     (a) If $\delta = 0$, $\text{Server}_2$ sets $DB' = \bot$.
     (b) Otherwise, let $p_1, \ldots, p_{n/2}$ be the list of pairs of indices $p_k = (j_{k,1}, j_{k,2})$
         such that $j_{k,1} \oplus j_{k,2} = \delta$ (ordered e.g. by increasing smallest value in
         the pair). These pairs are publicly computable given $\delta$. $\text{Server}_2$ obtains
         the database $DB'$ as $DB'[k] = DB[j_{k,1}] \oplus DB[j_{k,2}]$. ($DB'$ contains $n/2$
         records.)
  3. $\text{Server}_2$ sends $(\delta, DB')$ to Client.
  4. If $\delta = 0$, Client returns $DB[j]$.
  5. Otherwise, Client finds the pair $p_k$ such that $j \in p_k$. Let $i$ be the other index
     in $p_k$. Client returns $(i, DB'[k] \oplus DB[j])$.
```

Fig. 4. A simple multi-server RPIR protocol with $n/2$ bits of communication

each choice of δ leads to a different choice of i. Similarly, Server_2, who chooses δ, also learns nothing about i since the random and uniform index j is unknown to Server_2, and each choice of j leads to a different choice of i.

Non-Interactive Computational Multi-Server RPIR with a Better Rate We can use Reed-Muller codes and ideas from pseudo-random secret sharing to get a non-interactive multi-server scheme based on the existence of pseudorandom generators. (Note that, since this scheme relies on computational assumptions, it meets only a weaker — computational — version of Definition 4.) The construction follows the usual roadmap to get PIR from Reed-Muller codes, and then uses pseudo-random secret sharing (PRSS) to generate the line of points usually sent to the servers by the client.

We will encode DB as a multivariate polynomial. Let v be the number of formal variables. Let d be the maximal degree of the polynomial. Let $q > d$ be a prime. We consider multivariate polynomials $f(\mathtt{x}) \in \mathbb{Z}_q[\mathtt{x}_1, \ldots, \mathtt{x}_v]$ of degree at most d. It is easy to see that there are $K = \binom{v+d}{v}$ unique monomials of degree at most d,[5] so we can use $f(\mathtt{x})$ to encode an element from $\mathbb{Z}_q^K$. This will allow us to encode at least $K(\log_2(q) - 1)$ bits of the database by encoding bits into positions in the binary representation of the field elements in $\mathbb{Z}_q$. Note that this is a locally-decodable encoding: To decode a bit we only need the field element it sits in. The codewode will be $f(\mathbb{Z}_q^v)$, i.e., we evaluate f on all points in $\mathbb{Z}_q^v$. There are $N = q^v$ such points. We can encode by placing the K elements $\mathsf{DB}[i]$ in K evaluation points $f(a)$ and then use interpolation to compute $f(\mathbb{Z}_q^v)$. This

[5] Consider an array of length $v + d$. Consider placing a 0 in v positions and a 1 in the remaining d positions. Let the degree of $\mathtt{x}_i$ be the number of 1's between the i'th occurrence of a 0 and the $(i + 1)$'th occurrence of a 0 (or the end of the array when $i = v$). Clearly this gives total degree at most d and there is a one-to-one correspondence between such assignments and monomials of degree at most d. There are $k = \binom{v+d}{v}$ ways to place the v entries which are 0.

gives a linear code

$$\mathsf{Enc} : \mathbb{Z}_q^K \rightarrow \mathbb{Z}_q^N \ .$$

Let $f_{\mathsf{DB}}(\mathtt{x})$ be the polynomial used to encode DB. Below we call a point a encoding entries $\mathsf{DB}[j]$ a *database point*. We assume that each database point encodes the same number of bits.

We let $d = q - 2$. This means that the rate is

$$\frac{K}{N} = \frac{\binom{v+q-2}{v}}{q^v} .$$

For a constant v we have that

$$\binom{v+q-2}{v} = \Theta_q(q^v) \ ,$$

which gives us a constant rate.

Interactive Information-Theoretic Multi-Server RPIR. We can use the local decodability of Reed-Muller to get a multi-server RPIR for $c = q - 1$ servers $\mathsf{S}_1, \ldots, \mathsf{S}_c$ as follows.

1. Each server S_i forms the polynomial $f_{\mathsf{DB}}(\mathtt{x})$.
2. The client picks a uniformly random $a \in \mathbb{Z}_q^v$ and $b \in \mathbb{Z}_q^v$ and for $\lambda = 1, \ldots, q-1$ it lets $c_\lambda = a + \lambda b$. It queries S_i for $f_{\mathsf{DB}}(c_i)$.[6]
3. Let $\mathtt{y}$ be a formal variable over $\mathbb{Z}_q$ and consider the univariate polynomial $g(\mathtt{y}) = f_{\mathsf{DB}}(a + \mathtt{y}b)$. Since $f(\mathtt{x})$ has degree at most d, so does $g(\mathtt{y})$. The client knows $q - 1$ points on $g(\mathtt{y})$ as $g(i) = f_{\mathsf{DB}}(c_i)$. Since $d + 1 = q - 1$ the client can use interpolation to learn $g(0) = f_{\mathsf{DB}}(a)$.
4. If a happens to be a database point, then let j be uniform among the encoded entries j and output $(j, \mathsf{DB}[j])$. Otherwise, output $\perp$.

Privacy follows from $a + ib$ perfectly hiding a when $i \neq 0$. So a single server gets no information on a. Therefore, if a hits a database point j, then it hits a uniformly random database point in the view of the all servers. And each database point contains the same number of bits, so the position i will be uniform. The schemes has constant correctness. Namely, since the rate is constant it happens with constant probability that a hits a database point. This correctness can be amplified to any constant by a constant number of parallel repetitions and taking $(j, \mathsf{DB}[j])$ from the first correct instance. It can be amplified to negligible probability of error by repeating a linear number of times in the security parameter. For a batch scheme one can run $O(m)$ instances in parallel to get m correct instances except with negligible probability.

[6] For the reader familiar with Reed-Muller based PIR it looks odd to pick a at random. However, this leads up to the non-interactive versions, as detailed below.

Achieving Noninteractivity. In the above scheme the client can of course choose a to be a database point, yielding the well known Reed-Muller based multi-server PIR. Before showing how to derive a non-interactive version using pseudo-random secret sharing, we review the notion of pseudo-random secret sharing [9].

Consider servers $\mathsf{S}_1, \ldots, \mathsf{S}_c$ for $c = q - 1$. For each i we can pick a seed s_i for a pseudo-random generator and give s_i to all servers except S_i. By stretching the seed this will allow the servers to create any number of instances of pseudo-random $\alpha_1, \ldots, \alpha_c \in \mathbb{Z}_q$ where α_i is known to all servers $\mathsf{S}_j \neq \mathsf{S}_i$ and where α_i is indistinguishable from uniform in the view of S_i. Below we assume for simplicity that the elements are truly uniform.

Let $g_i(\mathtt{y}) \in \mathbb{Z}_q[\mathtt{y}]$ be a polynomial of degree 1 such that $g_i(0) = 1$ and $g_i(i) = 0$. Let $g_\alpha(\mathtt{y}) = \sum_{i=1}^{c} \alpha_i g_i(\mathtt{y})$. Note that $g_\alpha(0) = \sum_{i=1}^{c} \alpha_i$. This is an element uniformly random in the view of all servers. We can therefore take this to be one coordinate in our evaluation point $a \in \mathbb{Z}_q^v$. We can repeat v times in parallel to get all of a. Note also that S_i can compute $g_\alpha(i) = \sum_{j=1}^{c} \alpha_j g_j(i)$ as it knows α_j for $j \neq i$ and $g_j(i) = 0$ for $j = i$. This gives us the following non-interactive version.

1. The setup consists of seeds $s_1, \ldots, s_c$ for a PRG where s_i is given to all servers but S_i.
2. Each server S_i forms the polynomial $f_{\mathsf{DB}}(\mathtt{x})$.
3. The servers use v parallel instances of PRSS of lines (with $t = 1$) to implicitly generate uniformly random $a \in \mathbb{Z}^v$ and $b \in \mathbb{Z}^v$ such that for $\lambda = 1, \ldots, q-1$ server S_λ knows $c_\lambda = a + \lambda b$. Then S_i sends $(c_i, f_{\mathsf{DB}}(c_i))$ to C.
4. Let $\mathtt{y} \in \mathbb{Z}_q$ and consider the univariate polynomial $g(\mathtt{y}) = f_{\mathsf{DB}}(a + \mathtt{y}b)$. The client uses interpolation to learn $g(0) = f_{\mathsf{DB}}(a)$.
5. If a happens to be a point where f_{DB} encodes a database entries, then let j be uniform among the encoded entries and output $(j, \mathsf{DB}[j])$. Otherwise, output $\perp$.

Again we can use parallel repetition to amplify correctness.

We now consider the communication complexity of the protocol. We can make the optimization that only S_1 and S_2 send c_1 and c_2, as $c_3, \ldots, c_{q-1}$ can be computed by interpolation: the evaluation points are on a line. This is $2v$ elements from $\mathbb{Z}_q$. All $q - 1$ servers have to send $f_{\mathsf{DB}}(c_i)$, which is an element from $\mathbb{Z}_q$. This is, all in all, less than $q + 2v$ elements from $\mathbb{Z}_q$. For constant v the communication is therefore $\Theta_q(q)$ elements from $\mathbb{Z}_q$. We have that $K = \Theta_q(q^v)$ so for constant v and growing K we have that the communication is $\Theta_K(K^{1/v} \log(K))$ bits. The database has size $K \log_2(K)$. The constant rate of the Reed-Muller code will deteriorate with growing constant v. Therefore the number of times to iterate the RPIR in parallel to get a given correctness level will grow with v. The communication for each iteration drops with growing v. This means that in practice for a fixed K there is a tradeoff to be found for v.

4 Applications to Large-Scale DoS-Resistant Computation

As described in the introduction, a strong motivation for RPIR is setting up communication channels to random parties who should remain anonymous. Below we call these *target-anonymous communication channels.* Imagine a very large number of parties (perhaps millions), that want to securely perform some computation in the presence of a powerful denial of service (DoS) adversary. While distributed computation requires sending and receiving messages, in this setting the parties run the risk of being knocked offline by a targeted DoS attack as soon as the adversary learns that they play an important role in the computation.

If the adversary is limited to attacking at most some fraction f of the parties, then one solution is to run a secure MPC protocol among all the parties. If the MPC protocol is resilient to f fraction of misbehaving participants, the DoS adversary will not be able to disable sufficiently many participants to thwart the computation. But this resilience comes at a steep price, as MPC protocols typically requires communication between all pairs of parties, which is completely infeasible at the scales that we consider.

Another approach entails assigning special roles to a small number of parties, and relying on them to carry out the computation. This could be much more efficient, but security is a challenge: as soon as the adversary discovers what parties are playing the special roles, it can target those parties and knock them offline. Hence, realizing these potential efficiency gains requires that the parties playing special roles *remain anonymous up until they speak*, and moreover they *can only speak once* before their special role is concluded, else the adversary can identify and target them. The parties playing special roles can be thought of in terms of a sequence of *committees*, where parties in committee i speak simultaneously in the i'th round.

Secure-MPC protocols where parties only need to speak once were described in several recent works [2,3,7,11]. But using these protocols in the presence of that powerful DoS adversary requires solving a delicate problem: How can you send messages to these parties, in order to provide them with the state that they need to carry out their task? This is where we want to use target-anonymous channels. We need to continuously establish communication channels to random parties, while preventing the adversary from learning who are the recipients, so that it cannot target them for attacks.

Benhamouda *et al.* (BGG+) proposed in [2] one approach using a “nomination” process. First, a nominating committee is established using standard tools (such as VRFs, or by solving moderately hard puzzles). Then, every (honest) nominator p chooses another random party q, looks up its public key, and broadcasts a re-randomized version of that key. This lets everyone send messages to q, without the adversary knowing who the recipient is. As pointed out in the introduction, a side-effect of this nomination technique is that the adversary knows the identity of the nominee if *either* the nominator *or* the nominee is corrupt. So, if overall only some fraction f of the parties are corrupt, the adversary will know the identities of around $f + (1 - f)f$ of the committee members. This doubling is unfortunate; it implies that honest majority among the nominees (which is crucial for secure computation with guaranteed output delivery), requires that the overall fraction is bounded by some $f < 0.29$. In the following, we outline an approach that does not have this adversarial doubling effect.

4.1 Target Anonymous Communication Channels from RPIR

Rather than let individual parties establish target anonymous channels to future committee members, our solution leverage past committees to do this job.

That is, past committees will run a secure-MPC protocol to choose a random small subset of the public keys, re-randomize them, and then broadcast the result. Since past committees are ensured (by induction) to have honest majority, we no longer allow corrupt nominators to choose corrupt nominees. We are ensured that all future committee members are chosen at random, and the adversary does not know who they are (unless it happened to corrupt them independently).

The only issue with this solution, is that the circuit describing the nominator's function is large: The input consists of everyone's keys (which could number in the millions), hence a naive MPC protocol will be too expensive. This is where we use RPIR, we let past committees simulate the RPIR client, while the state of the RPIR server remains completely public (and so can be simulated locally by each committee member). Specifically, the server state in our protocol consists of the list of public keys belonging to all the parties, as well as some public randomness (e.g., derived from a beacon). Since the client's work and communication is much smaller than the database size, we obtain a secure-MPC protocol that scales well with the total number of parties.

To simplify the presentation we describe this solution in terms of a noninteractive RPIR protocol, but of course it can be adapted to handle arbitrary RPIR protocols. Let $\Pi = (\mathsf{Setup}, \mathsf{Client}, \mathsf{Server})$ be a noninteractive RPIR protocol, where:

- $\mathsf{Setup}(1^\kappa) \to (\mathsf{sk}, \mathsf{pk})$ is the client's setup function;
- $\mathsf{Server}(\mathsf{pk}, DB, \rho) \to m$ is the server's processing function (where ρ is randomness); and
- $\mathsf{Client}(\mathsf{sk}, m) \to (i, DB[i])$ is the client's output function.

For simplicity, assume that we have a one-time trusted setup, which is used to run the Setup procedure, makes pk publicly known by anyone, and shares sk among the members of an initial committee. Let d be the number of rounds required to run Client *together* with a re-randomization of the obtained key. Assume we are given a public source of randomness, and target anonymous communication channels to d committees, each guaranteed to have an honest majority, and the first of which has secret shares of the RPIR secret key sk. Then, we can generate communication channels to an arbitrary additional number of committees by using our existing committees to run the RPIR protocol (followed by key randomization).

Server: All committee members *locally* obtain the randomness ρ (from a public source of randomness), and evaluate $\mathsf{Server}(\mathsf{pk}, DB, \rho) \to m$. Note that, because the client secret state is secret shared, this message is not enough to reveal the output to any individual committee member. Note also that, since this computation was entirely local, no committee member needs to speak during this computation.

Output: The members of the d committees run $\mathsf{Client}(\mathsf{sk}, m) \to (i, DB[i])$, followed by a re-randomization of the retrieved public key, using techniques from [2,3,7,11] so that each committee only needs to speak once. Then they publicly reveal the output, thus establishing as many target-anonymous channels as needed to keep the process going.

This process consumes d committees, but can be used to make any desired number of key-selections and rerandomizations. In particular we can use it to establish d more committees that would handle the next selection, in addition to however many are needed to an external application. We can even let the same committee handle different steps of different RPIR instances: The last step in the protocol for the next committee, the second-to-last step in the protocol for the committee after that, *et cetera.* To conclude, we state the following informal theorem.

Theorem 4. *(informal) In the model of Benhamouda et al. [2] with a broadcast channel and mobile adversary, given anonymous PKE (for the target-anonymous channels) and a nontrivial weak RPIR protocol satisfying Definition 6, there exists a scalable evolving-committee proactive secret sharing scheme (ECPSS) as per [2, Def 2.3], tolerating any fraction $f < 1/2$ of corrupt parties.*

We note that the construction from [2] required other components (such as NIZK), but in our honest-majority setting those can be replaced by information-theoretic counterparts. We also comment that while the description above used public randomness, this can be replaced by the client generating the required randomness via a secure-MPC protocol. Also, we can use the same committees and the same techniques to get scalable secure-MPC for realizing arbitrary functions.

Theorem 5. *(informal) In the model of Benhamouda et al. [2] with a broadcast channel and mobile adversary, given anonymous PKE (for the target-anonymous channels) and a nontrivial weak RPIR protocol satisfying Definition 6, there exists scalable secure-MPC protocols for realizing any poly-time function, tolerating any fraction $f < 1/2$ of corrupt parties.*

5 Batch RPIR

We consider the application to large-scale secure-MPC as a “stress test” for RPIR efficiency. Not only do we need to run the RPIR client inside a secure-MPC protocol, but this protocol must use the only-speak-once pattern [11] which makes things hard, and we need to run very many copies of it to generate enough target-anonymous channels for it to sustain itself. It is therefore crucial to get the basic RPIR construction as efficient as can be for this application, which is what we do in this section. In particular, we consider a batch protocol that can choose multiple random indexes cheaper than choosing them one at a time, and also observe that the application can use a weaker security property than Definition 2, making it possible to do even better.

5.1 Definitions

Definition 2 can be easily adapted to amortized protocols in which the client gets more than a single entry of the database — say k entries at a time. The functionality for this case, denoted $\mathcal{F}^k_{\mathsf{RPIR}}$, is almost identical to the one from Section 2.2, except that the random single index $i \in [n]$ is replaced with a vector $\vec{i} \in [n]^k$. Everything else remains the same.

As we mentioned, it turns out that Definition 2 can sometimes be an overkill for applications of batch RPIR. In particular our motivating application uses RPIR to choose a random subset of indexes, where some subsets are "bad" (since they include too many corrupted parties), but they are very rare. In such an application, we may not really care about the chosen subset being random. Rather all we care about is that the odds of hitting a "bad subset" remains small. We thus weaken the security condition to only say that every collection of subsets that has negligible probability-mass by the uniform distribution, remains with a negligible probability-mass also in the RPIR output.

Formalizing this requirement using a game-based approach seems rather awkward, since the distribution of indexes that we care about is the a-posteriori distribution as seen by a computationally-bounded server. Fortunately it is easy to formulate it using the real/ideal approach of Definition 2. All we need to do is change the $\mathcal{F}^k_{\mathsf{RPIR}}$ functionality, so that instead of the uniform distribution, it chooses the indexes from some other distribution $\mathcal{D}$ which is "not too different" than uniform. Let us first define the statistical property of being not too different.

Definition 5 ((f, α)-domination). *Let D_1, D_2 be two distributions with X being the union of their support sets, and let $f, \alpha \in \mathbb{R}^+$ be positive numbers. We say that D_1 is (f, α)-dominated by D_2 if for any subset $S \subseteq X$ it holds that $D_1(S) \leq f \cdot D_2(S) + \alpha$.*

An ensemble $\mathcal{D}_1 = \{D_{1,k}\}_k$ is polynomially dominated *by another ensemble $\mathcal{D}_2 = \{D_{2,k}\}_k$ if each $D_{1,i}$ is (f_i, α_i)-dominated by $D_{2,i}$, where $\{f_k\}_k$ is polynomially bounded and $\{\alpha_k\}_k$ is negligible.*

It is clear that if $\mathcal{D}_1$ is polynomially dominated by $\mathcal{D}_2$, and some collection S has negligible probability in $\mathcal{D}_2$, then it also has negligible probability in $\mathcal{D}_1$.

The parametrized RPIR functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{RPIR}}$. The functionality is similar to the standard batch functionality $\mathcal{F}^k_{\mathsf{RPIR}}$, except that it is also parametrized by a distribution ensemble $\mathcal{D} = \{D_n\}_n$ (with D_n being a distribution over $[n]^k$).

When the client is honest and the server input is some $DB \in \{0,1\}^n$, the functionality draws an index set $\vec{i} \leftarrow D_n$ (rather than uniform in $[n]^k$) and returns to the client $(\vec{i}, DB[\vec{i}])$.

Definition 6 (Single-server batch weak RPIR). *A two-party protocol Π is a (semi-honest) single-server batch weak RPIR if it UC realizes the functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{RPIR}}$ with semi-honest server and honest client for some $\mathcal{D}$ which is polynomially dominated by the uniform distribution over $[n]^\kappa$ (with κ the security parameter). It is nontrivial if the server sends less than n bits.*

5.2 Constructions

Ishai, Kushilevitz, Ostrovsky, and Sahai (IKOS) described in [13] several constructions for batch PIR from standard PIR protocols. Unfortunately, even if we wanted to use those constructions to fetch *random* indexes (rather than specific ones), the underlying protocol must still be full-blown PIR (rather than RPIR). Luckily, it turns out that we can use similar approaches with an underlying RPIR protocol if we are willing to settle for the weaker security from Definition 6, and we can even get must better parameters than what the IKOS constructions give.

Specifically, below we describe how to modify the IKOS "expander-based" construction from [13]. The original construction, used to fetch k entries out of an n-entry database, is parameterized by two more integers $m > d \geq 2$. Using public randomness which is shared by the server and client, the construction uses m bins and puts every database entry into d random bins. This created a degree-d bipartite expander, with the n database entries on one side and the m bins on the other. Then for every k-subset of entries that the client wants to fetch, it finds a perfect matching in that expander graph, with the k requested entries on one side and a k-subset of the bins on the other. The client then uses standard PIR to fetch these items from their bins (and dummy items from the other bins).

As we mentioned above, even if we wanted to use that construction to fetch k random items, we would still need to fetch specific items from selected bins, so the underlying protocol must be a PIR protocol, rather than RPIR. In terms of parameters, that construction has "rate" of $\rho = 1/d \leq 1/2$ (meaning the total space taken by all the bins is d times larger than the database size), and it requires $m = \Omega(k(nk)^{1/(d-1)})$, which is optimal for replication-based constructions. We can apply this construction with much better parameters, however, if we are willing to settle for the weak security notion (but the underlying protocol must still be PIR rather than RPIR).

Lemma 5. *There exists a weak-RPIR scheme as per Definition 6 based on the IKOS expander-based construction [13], with parameters (k, d, m) such that $m = (1 + O(e^{-d}))k$.*

Proof (sketch). When running the expander-based scheme above with a much smaller m, there will necessarily be some k-subsets of indexes that cannot be retrieved. The RPIR protocol will therefore have the client resample its indexes until it arrives at a subset that can be retrieved one per bin.

It is easy to see that the fraction of k-subsets that cannot be retrieved with some parameters d, m, corresponds exactly to the failure probability of inserting k random elements into a Cuckoo hash table [17] with d hash functions and table-size m. It is known that for $d = 2$ it is enough to use $m = (2 + \epsilon)k$ to get failure probability $o(1)$, and for larger d we get the same guarantee with $m = (1 + O(e^{-d}))k$ (see e.g., Fountoulakis-Panagiotou-Steger [10]). The probability mass of each of the achievable subsets is therefore increased only by a $1 + o(1)$ factor, which means that any negligible-probability collection of subsets remain negligible. □

A Practically Appealing Weak Batch-RPIR While the construction above has good parameters, the work that the client has to perform is far from simple, as it needs to resample indexes until some perfect matching can be found in the construction graph. In our motivating application this would have to be done via secure MPC, requiring a complex and costly protocol. One could attempt to simplify this construction by having the client simply choose k random bins and retrieve a random item from each bin, but analyzing this variant is very challenging. Instead, we describe and analyze below an even simpler and more efficient construction.

The construction. In addition to n (the number of entries) and k (the number of indexes to fetch), the construction is also parametrized by m (the number of bins). We assume that both n and k are divisible by m, and note that k/m is playing a somewhat similar role to d in the expander-based construction. We deterministically partition the indexes in $[n]$ into m bins of size n/m each, for example $\{0, \ldots, \frac{n}{m} - 1\}, \{\frac{n}{m}, \ldots, \frac{2n}{m} - 1\}, \ldots$. Then we just fetch k/m random indexes from each bin using an underlying RPIR protocol. See Figure 5.

Simple Batch-RPIR (parameters $m < k < n$, m divides k, n)

1. Partition DB into m "bins", $B_i = \{DB[\frac{i \cdot n}{m}], \ldots, DB[\frac{(i+1)n}{m} - 1]\}$
2. Client, Server run k copies of RPIR to retrieve k/m entries from each B_i.

Fig. 5. A simple batch-RPIR protocol.

Note that by replicating each bin k/m times and fetching one item from each replica, we can view this construction as a very specific instance of the IKOS construction from [13] with exactly k bins, where instead of putting each item in $d = k/m$ random bins we put the first n/m items in bins $0, \ldots \frac{k}{m} - 1$, then the next n/m items in bins $\frac{k}{m}, \ldots \frac{2k}{m} - 1$, and so on. Note that we may end up fetching the same item more than once in this protocol, but this is quite acceptable for our application for large-scale MPC.

Analysis of the Simple Batch-RPIR Protocol. Clearly, if the underlying RPIR protocol has work $w(\kappa, n)$ and communication $c(\kappa, n)$ on databases of size n, then this protocol has work $k \cdot w(\kappa, n/m)$ and communication $k \cdot c(\kappa, n/m)$. In particular if the work is $w(\kappa, n) = p(\kappa) \cdot n$ then the work in this protocol is $p(\kappa) \cdot kn/m$, which is m times better than the naive solution of just running k RPIR instances against the entire database.

Theorem 6. *The simple batch-RPIR protocol from Figure 5 is a weak-RPIR protocol as per Definition 6, provided that the underlying RPIR protocol satisfies Definition 2 and that $m = O(\log \kappa / \log \log \kappa)$ (and $k = \mathrm{poly}(\kappa)$).*

We show that when drawing k elements at random from a universe of size n which is split evenly between m bins, the probability drawing exactly k/m elements from each bin is only exponentially small in m, regardless of n. Since $m = O(\log \kappa / \log \log \kappa)$, it means a noticeable probability in κ. We state the following lemma.

Lemma 6. $\binom{n}{k} / \binom{n/m}{k/m}^m = \Theta(\frac{1}{\sqrt{k}}(C \cdot k/m)^{m/2})$ *for some constant* C.

Proof. We use Stirling's approximation (cf. [19]) – namely, there are constants $C_1 = \sqrt{2\pi}$, and $C_2 = e$, such that for all positive t

$$C_1\sqrt{t} \cdot (t/e)^t \ < \ t! \ < \ C_2\sqrt{t} \cdot (t/e)^t.$$

Using these bounds we have:

$$\begin{aligned}
\binom{n}{k} \Big/ \binom{n/m}{k/m}^m &= \frac{n!(k/m)!^m (n/m - k/m)!^m}{k!(n-k)!(n/m)!^m} \\
&< \frac{C_2^{(1+2m)} \cdot n^{n+\frac{1}{2}} \cdot (k/m)^{k+\frac{m}{2}} \cdot ((n-k)/m)^{n-k+\frac{m}{2}}}{C_1^{(2+m)} \cdot k^{k+\frac{1}{2}} \cdot (n-k)^{n-k+\frac{1}{2}} \cdot (n/m)^{n+\frac{m}{2}}} \\
&= \frac{C_2^{(1+2m)} \cdot k^{(m-1)/2} \cdot (n-k)^{(m-1)/2}}{C_1^{(2+m)} \cdot n^{(m-1)/2} \cdot m^{m/2}} \\
&< \frac{C_2}{C_1^2 \cdot \sqrt{k}} \cdot \left(\frac{C_2^4}{C_1^2} \cdot \frac{k}{m}\right)^{m/2} \ < \ \frac{1}{2\sqrt{k}} \cdot \big(9k/m\big)^{m/2}. \qquad (1)
\end{aligned}$$

Lemma 6 implies that drawing k/m elements from each of the m bins (rather than drawing k elements uniformly from the entire universe) increases the probability of each k-subset by at most a factor of $\Theta(\frac{1}{\sqrt{k}}(C \cdot k/m)^{m/2})$ for some $C < 9$. For $k = \mathrm{poly}(\kappa)$ and $m = O(\log \kappa / \log \log \kappa)$, this factor is polynomial in the security parameter. Finally, the underlying RPIR protocol satisfying Definition 2 implies that the server cannot distinguish the output of the protocol from drawing exactly k/m random elements from each bin. This concludes the proof of Theorem 6. □

f	m	k	f	m	k
0.2	10	440	0.3	10	1080
0.2	40	640	0.3	40	1560
0.25	10	680	0.35	10	1850
0.25	40	1000	0.40	10	3500

Table 1. Some parameters for batch-RPIR with $n = 10000$ and security level=128.

Setting the Parameters. While the general Theorem 6 only holds for very small $m = O(\log \kappa / \log \log \kappa)$, in the context of our motivating application we can choose much large values, linear in κ. The reason is that the probability mass of the "bad subsets" in this case is exponentially small, not just negligible. As we show below we can choose the committee-size k as a small multiple of the security parameter. Hence, we not only get much better resilience than Benhamouda *et al.* [2], but also much smaller committees, and the secure-MPC cost can be kept small by increasing the number of bins m.

In the application from Section 4 we have an adversary $\mathcal{A}$ that watches an execution of the batch-RPIR protocols (for choosing k parties from a universe of size n in m bins). Then $\mathcal{A}$ adaptively corrupts up to $f \cdot n$ parties (for some $f < 1/2$). For each corrupted party, $\mathcal{A}$ learns if that party was chosen or not, and its goal is to corrupt $k/2$ (or more) of the parties that were chosen by the protocol.

To get concrete parameters, we can start by analyzing the naive RPIR protocol with one bin, and then view Lemma 6 as quantifying the security loss by going to the more efficient protocol with m bins. By that lemma, the min-entropy of $\mathcal{D}$ (and hence the security level) decreases by roughly $\frac{m}{2} \log(9k/m)$ bits when switching from one to m bins. Analyzing the naive protocol is rather straightforward. For example, we can use the Chernoff bound, which says that for any $f \lnsim 1/2$ we can set $k = c \cdot \kappa$ for some $c = \Theta(f(\frac{1}{2} - f)^2)$ to get security level of (say) 2κ. We can then set $m = \kappa/\Theta(\log c) = k/\theta(c \log c)$ and lose only κ bits, obtaining security κ while selecting only a constant $\Theta(c \log c)$ parties from each bin.

It turns out that for our parameter regime the Chernoff bound is rather loose, and we get much better concrete parameters using an exact calculation. Specifically, for the one-bin protocol we need to compute the probability that a random f-subset of $[n]$ contains more than 1/2 of the elements in $[k]$. The exact expression for this probability is

$$\sum_{i=k/2}^{k} \binom{fn}{i} \binom{(1-f)n}{k-i} \Big/ \binom{n}{k},$$

which is easy to compute for specific n, f, k values. Accounting for the "penalty" from Lemma 6 we therefore get:

Lemma 7. *For a specific setting of the parameters f, n, k, m, κ, if the underlying RPIR protocol satisfies Definition 2 then for any poly-time adversary $\mathcal{A}$ it holds that,*

$$\begin{aligned}&\Pr[\mathcal{A} \textit{ corrupts } k/2 \textit{ or more selected parties}]\\&\quad\le \frac{\sum_{i=k/2}^{k}\binom{fn}{i}\binom{(1-f)n}{k-i}}{\binom{n}{k}} \cdot \frac{1}{2\sqrt{k}} \cdot \left(\frac{9k}{m}\right)^{m/2} + \mathsf{negligible}(\kappa).\end{aligned}$$

□

In Table 1 we list a few example parameters for $n = 10000$ parties, corrupt fractions $f \in [0.2, 0.4]$, and various k, m values that achieve security level $\kappa = 128$.

References

1. A. Beimel, Y. Ishai, and T. Malkin. Reducing the servers computation in private information retrieval: PIR with preprocessing. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 55–73. Springer, Heidelberg, Aug. 2000.
2. F. Benhamouda, C. Gentry, S. Gorbunov, S. Halevi, H. Krawczyk, C. Lin, T. Rabin, and L. Reyzin. Can a public blockchain keep a secret? In R. Pass and K. Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 260–290. Springer, Heidelberg, Nov. 2020.
3. E. Blum, J. Katz, C. L. Zhang, and J. Loss. Asynchronous byzantine agreement with subquadratic communication. *IACR Cryptol. ePrint Arch.*, 2020:851, 2020.
4. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.
5. D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.
6. B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, Oct. 1995.
7. A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 94–123, Virtual Event, Aug. 2021. Springer, Heidelberg.
8. H. Corrigan-Gibbs and D. Kogan. Private information retrieval with sublinear online time. In A. Canteaut and Y. Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 44–75. Springer, Heidelberg, May 2020.
9. R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In J. Kilian, editor, *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362. Springer, 2005.
10. N. Fountoulakis, K. Panagiotou, and A. Steger. On the insertion time of cuckoo hashing. *SIAM J. Comput.*, 42(6):2156–2181, 2013. See `https://arxiv.org/abs/1006.1231`.

11. C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In T. Malkin and C. Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 64–93, Virtual Event, Aug. 2021. Springer, Heidelberg.
12. O. Goldreich and L. A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32. ACM Press, May 1989.
13. Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch codes and their applications. In L. Babai, editor, *36th ACM STOC*, pages 262–271. ACM Press, June 2004.
14. E. Kushilevitz and R. Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, Oct. 1997.
15. E. Kushilevitz and R. Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 104–121. Springer, Heidelberg, May 2000.
16. M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *21st ACM STOC*, pages 33–43. ACM Press, May 1989.
17. R. Pagh and F. F. Rodler. Cuckoo hashing. In F. M. auf der Heide, editor, *Algorithms - ESA 2001, 9th Annual European Symposium, Aarhus, Denmark, August 28-31, 2001, Proceedings*, volume 2161 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2001.
18. E. Stefanov, M. van Dijk, E. Shi, C. W. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 2013*, pages 299–310. ACM Press, Nov. 2013.
19. Stirling's approximation. `https://en.wikipedia.org/wiki/Stirling%27s_approximation`, accessed Oct 2020.

A Random-Index Oblivious-RAM

In this section we note that a random-index ORAM (RORAM) can be used in our motivating application instead of RPIR, resulting is a somewhat different performance profile. We begin by defining RORAM.

A Random-Index ORAM (RORAM) is a two party protocol between a client and a server similar to Oblivious RAM (ORAM), except that the client does not choose the indexes to read from memory. Instead, these indexes are chosen at random (by the protocol), with the client getting (i, Mem_i) while hiding them from the server. Similarly to ORAM, we have procedures for Init, Read, and Write, except that the index to be read is not an input to Read but an output of it.

Definition 7 (RORAM Syntax). *A Random-Index ORAM protocol (RORAM) consists of the following components:*

- $\mathsf{Init}(1^\kappa, \mathsf{Mem}) \to (\mathsf{cst}; \mathsf{SST})$*: The initialization algorithm takes as input the security parameter and initial memory* $\mathsf{Mem} \in \{0,1\}^*$ *(that could be empty), and generates an initial secret client state* cst *and a public server state* SST*.*

- $\mathsf{Read}(\mathsf{cst}, \mathsf{SST}) \to (i, x, \mathsf{SST}')$*: The client fetches* (i, Mem_i) *(presumably for a random index* $i \in |\mathsf{Mem}|$*), and the server state is updated to* SST'*.*[7]
- $\mathsf{Write}(\mathsf{cst}, i, x, \mathsf{SST}) \to \mathsf{SST}'$*: The content of the memory is modified by setting* $\mathsf{Mem}[i] := x$ *and the server state is updated to* SST'*.*

A RORAM protocol is nontrivial if the communication in each of Read *and* Write *operations is* $o(|\mathsf{Mem}|)$*.*

Desired properties: The security notion for (computational) ORAM from [18] intuitively says that the server should not learn anything about which data and in what order it is being accessed. (We may also require that the server cannot learn if the operation is read or write.) As for RPIR, here too it is convenient to define security by means of an ideal functionality.

RORAM Functionality. The functionality $\mathcal{F}_{\mathsf{RORAM}}$ takes as input a (possibly empty) initial $\mathsf{Mem} \in \{0,1\}^*$ from the client. It stores Mem internally and gives the size of the memory $|\mathsf{Mem}|$ to the server.

Thereafter, on input Read from the client it sets $n := |\mathsf{Mem}|$, chooses at random an index $i \leftarrow [n]$, returns $(i, \mathsf{Mem}[i])$ to the client, and outputs n to the server. On input $\mathsf{Write}(i, x)$ from the client (i in unary) it modifies $\mathsf{Mem}[i] := x$ (extending the memory if needed), and outputs the new $|\mathsf{Mem}|$ to the server.

Definition 8 (RORAM). *A two-party protocol Π is a Random ORAM if it realizes the functionality* $\mathcal{F}_{\mathsf{RORAM}}$ *above.*

A.1 Target Anonymous Channels from RORAM

One can use (batch) RORAM as an almost "drop-in" replacement for (batch) RPIR to establish target-anonymous channels. Here too we have previous committees playing the part of the RORAM client, where the server state is publicly known so every committee member can simulate the server in its head. However, there are a few differences.

In the RPIR-based solution, the server state only changes when the database contents change; that is, when public keys are added or removed due to a party joining or leaving the pool of participants (or parties changing their keys). When this happens, no additional communication is needed to run the RPIR server, since all parties can update the server state locally. In contrast, the RORAM server state is evolving dynamically with each read/write operation, and the state depends on the client secret. This has several consequences. First, setting up the server state takes $O(n)$ communication (where n is the number of parties in the pool of participants), since communication with the client (played by the committees) is necessary for every write. Second, every party in the pool of participants must continuously update the server state and keep a local copy of it, so that it can simulate the server for itself if it gets selected to one of these committees. Namely, whenever a client-simulating committee broadcasts

[7] We can assume wlog that the client state does not change throughout the protocol.

an RORAM-client message, every party in the universe must update its local copy of the RORAM-server state accordingly.

The rest of the construction works just like the RPIR-based solution, with the committees implementing the RORAM client and any secrets that the client requires passed from committee to committee using the proactive secret sharing technique of Benhamouda *et al.* [2]. The result is summarized by the following informal theorem:

Theorem 7. *In the model of Benhamouda et al. [2] with a broadcast channel and mobile adversary, given anonymous PKE (for the target-anonymous channels) and a nontrivial RORAM protocol satisfying Definition 8, there exists a scalable ECPSS scheme as per [2, Def 2.3], tolerating any fraction $f < 1/2$ of corrupt parties.*

We remark that there is an interesting trade-off between the RPIR-based and the RORAM-based solutions: While both tools can provide a scalable solution (in that the amount of communication in each step is independent of the universe size n), they differ in how many parties need to perform local computation, and how much local computation each of them must do.

- When using RPIR, the only parties that need to perform local computations in each step are the current committee members (so only $O(\kappa)$ of them). However, each one of them must play the RPIR server, so it must do at least $\Omega(n)$ operations.
- When using RORAM, every party in the universe must keep up to date with the evolving server state, so every party must perform some computation in every step.[8] On the other hand, the computational complexity of one server-step is typically just polylog(n) (depending on the underlying RORAM protocol).

Hence we have a choice between $O(\kappa)$ parties performing $\Omega(n)$ operations each for RPIR, or all n parties performing only polylog(n) operations each for RORAM. It is an interesting open problem to find a solution where both the number of computing parties and the complexity of operations is sublinear in n (possibly using some combination of RPIR and RORAM).

B Target Anonymous Channels from Mix-Nets

A different approach to setting up target anonymous communication channels is using Mix-Nets [5], i.e., by repeatedly shuffling and re-randomizing all the keys. This solution can be implemented simply by having individual parties self-select to shuffle and re-randomize all parties' public keys, then proves in zero knowledge that they did so correctly. Since the shuffling parties do not need any secret state,

[8] Parties can perform these computations lazily, only when they are selected to a committee, but this does not change the total number of operations that they must perform.

they can self-select using VRFs or by solving moderately-hard puzzles. There is no need to establish target-anonymous channels with these parties as recipients.

Notice that this setting is slightly different than traditional use of Mix-Nets, in that the shuffled and re-randomized entities are themselves public keys, with the corresponding secret keys held by individual parties. This means in particular that the adversary can always recognize its own keys in the shuffled list; only the honest parties' keys are hidden. Therefore, even after all the shuffling is done, we still require fresh public randomness — unpredictable by the adversary — to select the rerandomized keys from the shuffled database. (Otherwise a malicious last shuffler can plant keys belonging to corrupt parties in the positions from which keys are to be selected.)

This solution uses κ (security parameter) shuffles, so that at least one of the shufflers will be honest with overwhelming probability. As usual with Mix-Nets, all we need is one honest shuffler, as biased shuffles do no harm as long as at least one shuffle along the way is uniform. Also, we assume a synchronous model, so if one or more shufflers do not show up to play their roles, we simply skip their turns.

The major drawback here is communication; each of the κ shufflers needs to broadcast n public keys, or $O(n\kappa)$ bits. This gives us a total communication complexity of $O(n\kappa^2)$. On the other hand, this solution is very simple and requires no evolving secret state to be passed among the parties, making it appealing in some practical settings where the number of parties is not so large.

The solution can be optimized further, along somewhat similar lines to the batch-RPIR construction from Section 5.2: We divide the database of public keys into m bins each containing $\frac{n}{m}$ public keys. We then run the Mix-Net solution above on each bin separately, using independently-chosen set of shufflers for each bin. Finally we use fresh public randomness to select k/m committee members from each bin. Note that we can now use only $s \ll \kappa$ shuffling steps, maybe as little as $s = \Theta(1)$. Each bin has 2^{-s} probability of having all corrupt shufflers, hence starting from an f-fraction of corrupt parties the expected fraction of corrupt committee members per bin is $f' = 2^{-s} + f(1 - 2^{-s})$, and setting m large enough we can ensure that the actual fraction is very close to f' whp.

The total communication complexity of this modified scheme becomes $O(n\kappa s)$. For comparison, the FHE-based batch RPIR approach (Section 3) in combination with YOSO MPC gives total communication complexity of $\tilde{O}(\kappa^3)$, where both the size of a YOSO MPC committee and the number of keys being selected (for communication channels to the next committee) is $O(\kappa)$, and the length of an FHE decryption share is $\tilde{O}(\kappa)$. While the dependence of the communication complexity on n in the Mix-Nets solution may appear crippling, in practice the term $\tilde{O}(\kappa^3)$ may dwarf the number of participants n.