Random-index PIR with Applications to Large-Scale Secure MPC

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

[1]Algorand Foundation, USA, craig@algorand.foundation, shaih@alum.mit.edu
[2]Aarhus University, Denmark, {magri,jbn}@cs.au.dk,sophia.yakoubov@gmail.com

October 9, 2020

Abstract

Private information retrieval (PIR) lets a client retrieve an entry from a database held by a server, without the server learning which entry was retrieved. Here we study a weaker variant that we call *random-index PIR* (RPIR). It differs from standard PIR in that the retrieved index is an *output* rather than an input of the protocol, and it is chosen at random.

Our motivation for studying RPIR comes from a recent work of Benhamouda *et al.* (TCC'20) about maintaining secret values on public blockchains. Their solution involves choosing a small anonymous committee from among a large universe, and here we show that RPIR can be used for that purpose.

The RPIR client must be implemented via secure MPC for this use case, stressing the need to make it as efficient as can be. Combined with recent techniques for secure-MPC with stateless parties, our results yield a new secrets-on-blockchain construction (and more generally large-scale MPC). Our solution tolerates any fraction $f \lesssim 1/2$ of corrupted parties, solving an open problem left from the work of Benhamouda *et al.*

Considering RPIR as a primitive, we show that it 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, which is clearly impossible for PIR. We also study *batch* RPIR, where multiple indexes are retrieved at once. Specifically we consider a weaker security guarantee than full RPIR, which is still good enough for our motivating application. We show that this weaker variant can be realized more efficiently than standard PIR or RPIR, and 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.

Contents

1 Introduction

A single-server 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 it 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. [4] who described a solution with multiple non-colluding servers; a single-server solution was first described by Kushilevitz and Ostrovsky [11].

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.

To some extent, we show that this is *not* the case, since every RPIR protocol can be converted to a PIR protocol with only slightly more communication and rounds. Given an r-round RPIR protocol with server communication $m < n$, we show how to construct:

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

On the other hand, we show that 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. But we show that existing interactive PIR protocols have counterpart RPIR protocols in this model. In particular we show a Kushilevitz-Ostrovsky-like protocol [12] in this model, with communication $n - \Omega(n/\kappa)$ assuming trapdoor permutations, and an FHE-based protocol in this model with communication $\log n \cdot \mathsf{poly}(\kappa)$. Other examples of settings where RPIR is more efficient than PIR are discussed in Section 1.3 below.

1.2 Application to Large Scale MPC

Our motivation for studying RPIR came from a recent work of Benhamouda *et al.* [1] 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 ingredient in that work is a committee-selection procedure that establishes "target anonymous" secure channels to a fresh committee, allowing everyone to send them secret messages while keeping the adversary from learning their identities. Once established, an old committee with shares of the secret uses those channels to re-share the secret to the new committees. Importantly, the ECPSS scheme

from [1] is *scalable*: its communication complexity does not grow with the number of parties in the system.

Benhamouda et al. commented that once we have ECPSS, the committees can also compute on these secrets, not just re-share them. Indeed, very recent works of Choudhuri et al. [5], Blum et al. [2], and Gentry *et al.* [7] describe techniques for secure computation in this demanding model of (essentially) stateless parties. In particular, the paper Gentry *et al.* [7] describes protocols with guaranteed output delivery, provided that all the committees have honest majority.

A drawback of the scheme of Benhamouda *et al.* [1] is that it can only tolerate up to about 1/4 corruptions. The reason is that committee-selection is done by individual parties, who "nominate" members to the new committee by establishing target-anonymous channels to 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 sometime nominate corrupted parties by chance). Ensuring an honest majority among the new committee is therefore only possible when less than about 1/4 of the parties overall are corrupted.

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 [1] is

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

where the re-randomized keys implement the target-anonymous channels leading to the k members of the next committee. 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 may not be scalable: The circuit of the 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 plays the role of the RPIR server. (The database is the list of n public keys, which is known to everyone.) The result of the RPIR protocol is the committee (representing the client) fetching 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 only on k, not on the total number n of keys. Putting all these ideas together we get:

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

1.3 Batch RPIR

In our motivating application above, 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 protocols. Building such batch PIR protocols from PIR was studied by Ishai et al. [10]. 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 solutions from [10].

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 as a feature of the RPIR protocol itself takes some work to get it right. In Section 2.2 we describe a definition in the real/ideal style.

Having lowered the security bar, we take another look at the constructions from [10] 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 3.2 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 Organization

In Section 2 we formally define RPIR and batch-RPIR and study the relations to PIR. In Section 3 we describe some constructions of RPIR in the noninteractive setting, and efficient constructions of batch-RPIR with weak security. In Section 4 we describe the application of batch RPIR with weak security to the architecture of Benhamouda *et al.* [1] for large-scale MPC. We also sketch there an alternative mix-nets-based construction, which is not scalable but could be more concretely efficient in certain cases.

Finally, 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.

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 Π 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 $CC_\Pi(n,\kappa)$. The two properties of interest for a PIR protocol Π are its client-privacy (i.e. the index i is hidden from the server) and its communication complexity.

Definition 1 (Single-server PIR [11])**.** *A two-party protocol Π is a (semi-honest) single-server PIR if it satisfies:*

Correctness. *The client's output in $\Pi(\kappa, n;\ i, DB)$ 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* $\mathsf{serverView}(\Pi(\kappa, n;\ i, DB))_\kappa$ *and* $\mathsf{serverView}(\Pi(\kappa, n;\ i', DB))_\kappa$ *are computationally indistinguishable.*

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

A Symmetric PIR *(SPIR) protocol [8] satisfies all the above, and in addition also the following* Database privacy *condition: For every n, index i, and two databases $DB, DB' \in \{0,1\}^n$ s.t. $DB[i] = DB'[i]$, the ensembles* $\mathsf{clientView}(\Pi(\kappa, n;\ i, DB))_\kappa$ *and* $\mathsf{clientView}(\Pi(\kappa, n;\ i, DB'))_\kappa$ *are computationally indistinguishable.*

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.

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.[1] 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.

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 be convenient to define client-privacy by means of an "ideal RPIR functionality."

2.2.1 The RPIR functionality.

The functionality $\mathcal{F}_{\mathsf{RPIR}}$ accepts from the server an input $DB \in \{0,1\}^*$ and then waits for the client to ask for an output. If the client is honest then $\mathcal{F}_{\mathsf{RPIR}}$ sets $n = |DB|$, chooses $i \leftarrow [n]$ uniformly at random, and returns $(i, DB[i])$ to the client. If the client is corrupted then the functionality just gives it the entire database DB. (Alternatively, a random-SPIR functionality gives only $DB[i]$ to a corrupted client.)

Definition 2 (Single-server RPIR)**.** *A two-party protocol Π is a single-server RPIR if it realizes the functionality $\mathcal{F}_{\mathsf{RPIR}}$ above. It is nontrivial if for any κ and large enough n, it holds that $CC_\Pi(n, \kappa) < n$. (Similarly, the protocol is single-server RSPIR if it realizes the random-SPIR functionality.)*

We note that one can contemplate a security notion in between RPIR and RSPIR. For example the functionality can let a corrupted client choose the index, or maybe even apply an arbitrary predicate to the database.

[1] Note that standard PIR does not provide any privacy to the server, hence the functionality lets a corrupted client get the entire database. Alternatively a SPIR functionality gives only $DB[i]$ to a corrupted client.

2.3 Defining Batch RPIR

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

Batch RPIR with Weaker Security. It turns out that Definition 2 can sometimes be an overkill for applications of batch RPIR. Consider an application that uses RPIR to choose a random subset of indexes, where some subsets are "bad" 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" is 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 above. 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 3 ((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 4 (Single-server batch weak RPIR)**.** *A two-party protocol Π is a single-server batch weak RPIR if it realizes the functionality $\mathcal{F}^{\mathcal{D}}_{\mathsf{RPIR}}$ for some $\mathcal{D}$ which is polynomially dominated by the uniform distribution over $[n]^\kappa$ (with κ the security parameter). It is non-trivial if the server sends less than n bits.*

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, as stated in the introduction.

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 $DB[i] = DB[j] \oplus \sigma_{\{i,j\}}$.

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

Claim 1. *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.

RecursivePIR[Client($i \in [n]$), Server($DB \in \{0,1\}^n$)] (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 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 $p = \{k, k \oplus \delta\}$ 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 RecursivePIR[Client(i'), Server(DB')], client gets $\sigma_{i'}$.
5. Client outputs $DB[i] = DB[j] \oplus \sigma_{i'}$.

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

Theorem 1. *Given an r-round RPIR with client-communication k and server-communication m, there is a PIR protocol with $(r+1)\lceil \log n \rceil$ rounds, client communication $k\lceil \log n \rceil + \binom{\lceil \log(n) \rceil}{2}$ and server communication $m\lceil \log n \rceil + 1$.*

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 Claim 1. □

2.4.1 PIR from RPIR with Fewer Rounds

While the protocol in from 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.

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

1. Server and client run in parallel t' executions RPIR[Client, Server(DB)], where t' 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 $DB[i] = DB[j_1] \oplus \cdots \oplus DB[j_{t-1}] \oplus \sigma_I$.

Figure 3: A partition-based PIR protocol

Exactly the same proof as Claim 1 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 $m < n/2$, 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 roughly $2\sqrt{mn}$.

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 Protocols

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. Below we sketch two noninteractive RPIR protocols in the semi-honest model with pre-processing, one based on trapdoor permutations (similar to Kushilevitz-Ostrovsky [12]), and the other based on FHE. In these protocols 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.)

3.1.1 Noninteractive RPIR from One-way Trapdoor Permutations.

This construction is based on the Kushilevitz-Ostrovsky PIR protocol from [12]. 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. (Each random index costs just a little less than n bits of communication for an n-bit database.)

Let π be the permutation that the client sent to the server (and for which it has a trapdoor). Let $h_\pi(x)$ be a Naor-Yung universal one-way hash function (UOWHF) based on π [13], with 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 [9] $\rho_r(x) = \langle x, r \rangle \bmod 2$. 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 non-trivial 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 [12].

Theorem 3. *If trapdoor one-way permutations exist, then there exists a nontrivial non-interactive 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 $x'^1_i = x_i^1 \oplus w$ instead of x_i^1 for all i. This ensures that $x_i^0 \neq x'^1_i$ except with exponentially small probability, and the client can mask-out w at the end of the protocol if needed.

3.1.2 Noninteractive RPIR from FHE.

Of course we can get a construction with better parameters from FHE: For example, the client sends to the server "once and for all" an encryption of a seed s for a (weak) 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.

3.2 Batch RPIR Protocols

Ishai *et al.* described in [10] several constructions for batch PIR. One of them is an "expander-based" construction for fetching k entries out of an n-entry database: Parametrized by integers $m > d \geq 2$, the construction keeps m bins and puts every database item into d random bins, then for every k-subset of items it finds a k-subset of the bins that can be used to fetch these items, one from each bin, and uses standard PIR to fetch these items from their bins (and dummy items from the other bins).

One drawback of this construction in our context is that even if we wanted to fetch a *random* index (rather than a specific one), we still need to be able to specify exactly which index to get from each bin. Hence the underlying protocol must be a full-blown PIR rather than an RPIR protocol.

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 get much better parameters, however, if we are willing to settle for the weak security notion.

Lemma 1. *There exists a weak-RPIR scheme as per Definition 4 based on the expander-based construction of Ishai et al. [10], with parameters (k, d, m) such that $m = (1 + O(e^{-d}))k$.*

sketch. When running the PIR scheme above with a much smaller m, there will necessarily be some k-vectors of indexes that cannot be retrieved. The RPIR protocol will therefore have the client resample its indexes until it arrives at a vector that can be retrieved one per bin, and then use that k-vector of indexes.

It is easy to see that the fraction of k-vectors that cannot be retrieved with some parameters d, m, corresponds exactly to the failure probability of inserting k random elements into a Cuckoo hash [14] table 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 *et al.* [6]). The probability mass of each of the achievable vectors is therefore increased only by a $1 + o(1)$ factor, which means that any negligible-probability collection of vectors remain negligible. □ □

3.2.1 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 the analysis for this variant has proven to be very challenging. Instead, we describe and analyze below an even simpler and more efficient construction.

The construction. As above, the construction is parameterized by k (the number of indexes to fetch) and 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 construction above. We deterministically partition the indexes in $[n]$ into m bins of size m/n 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 4.

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 Ishai *et al.* construction from [10]

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

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

Figure 4: A simple batch-RPIR protocol.

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.

3.2.2 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 4. *The simple batch-RPIR protocol from Figure 4 is a weak-RPIR protocol as per Definition 4, provided that the underlying RPIR protocol satisfies Definition 2 and that* $m = O(\log \kappa / \log \log \kappa)$ *(and* $k = \text{poly}(\kappa)$*).*

Below we show that when drawing k elements at random from a universe of size n which is split evenly between m bins, there is a somewhat-large probability (only exponentially small in m) to draw exactly k/m elements from each bin. We state the following lemma.

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

Proof. We use Stirling's approximation (cf. [16]) – 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:

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

□ □

Lemma 2 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 $k = \text{poly}(\kappa)$ 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 4. □

3.2.3 Setting the Parameters.

While the general Theorem 4 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 we know which are the "bad subsets," and we can prove that their probability mass 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.* [1], but also much smaller committees, and the secure-MPC cost can be kept small by increasing the number of bins m.

Looking ahead to the relevant security game for the application to large-scale DoS-resilient secure-MPC described in Section 4: We consider an adversary $\mathcal{A}$ that watches an execution of the batch-RPIR protocols (that chooses k parties from a universe of size n in m bins). Then it 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 2 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 $f \lneq 1/2$ 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

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

which is easy to compute for specific n, f, k values. Accounting for the "penalty" from Lemma 2 we therefore get:

Claim 2. *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,*

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

□

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

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 (we call these *target-anonymous communication channels*). One setting where this is crucial is computation among a large

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

Table 1: Some parameters for the batch-RPIR protocol with $n = 10000$, with the security level set to 128 bits.

number of parties — perhaps millions — in the presence of a powerful denial of service (DoS) adversary, as in the work of Benhamouda *et al.* [1] and Gentry *et al.* [7]. Imagine that a large number of parties want to perform a secure computation, but as soon as a given party is known to play a crucial role in the computation, the adversary can instantly take that party offline with a targeted attack.

If the adversary is limited to attacking at most some fraction f of the parties, one solution is to run a secure multi-party computation (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 scale 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. (This was dubbed the *you-only-speak-once (YOSO) model* by Gentry *et al.* [7].) 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. Protocols following this YOSO approach were described in [1] and [7]. Roughly speaking, they consist of two types of roles:

Public-state roles, that do not require any secrets in order to compute their one outgoing message.

Secret-state roles, that must receive private communications from parties in previous committees before they can compute their outgoing message.

Public-state roles can be filled by having parties self-elect to them, using known techniques such as cryptographic sortition or solving moderately-hard puzzles. Assigning parties to secret-state roles and providing them with the secrets that they require is where we need target anonymous communication channels.

As advocated in [7], it makes sense to consider separately the questions of (1) setting up the target anonymous communication channels, and (2) leveraging those communication channels to perform the computation. The paper [7] considered the latter question, while we focus on the former. Target anonymous channels can be realized via public keys, with the corresponding secret keys known only to random anonymous parties. But choosing the recipients and providing them with the secret keys that they need — all while keeping their identities from the adversary — is challenging.

Benhamouda *et al.* proposed in [1] one approach using "nomination"; for each secret-state role, there is an associated public-state role whose job is to nominate a party to that secret-state role, and to publish a re-randomization of that party's public key. 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. Corrupt nominators will always nominate their fellow corrupt parties, while honest nominators occasionally nominate corrupt parties by chance. 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 two subsections, we outline two approaches to setting up target anonymous communication channels that do not have this adversarial doubling effect. The first (Section 4.1) uses RPIR; the second (Section 4.2) uses a Mix-Net [3] approach.

4.1 Target Anonymous Communication Channels from RPIR

This solution bootstraps past committees with secret-state roles to set up target anonymous communication channels to the next set of secret-state roles. Importantly, we rely on techniques from Gentry *et al.* [7], showing how to run MPC protocols that implement arbitrary functionalities YOSO-style, as long as all the committees have honest majority. In our case, we would let these past committee simulate the RPIR client, while the state of the RPIR server remains completely public (and so can be simulated locally by each committee member).

The server state in our setting consists of the list of public keys belonging to all the parties, as well as some public randomness (e.g., derived from a beacon). In the description below we consider specifically a noninteractive RPIR protocol as discussed in Section 3. However, this is done only for efficiency and ease of presentation, as this solution can clearly be adapted to handle arbitrary RPIR protocols.

The committees consisting of secret-state roles will play the role of the RPIR client. Any secrets that the client needs to use — such as a secret key — will be passed from committee to committee using the proactive secret sharing technique described by Benhamouda *et al.* [1]. Since the server state is public, each committee member can compute the server's messages in its head. They then use YOSO-style secure MPC protocol to compute the client's output (which is a randomly selected public key); before opening that output, however, they run YOSO MPC again to re-randomize the retrieved public key, publishing only the re-randomized key. More formally, let $\Pi = (\mathsf{Setup}, \mathsf{Client}, \mathsf{Server})$ be a non-interactive RPIR protocol, where:

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

We consider a setting with trusted setup, which in particular is running the Setup procedure, makes pk publicly known by anyone, and shares sk among an initial committee of honest

secret-state roles. 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 $\mathsf{Server}(pk, DB, \rho) \to m$. 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 committee run $\mathsf{Client}(sk, m) \to (i, DB[i])$ within YOSO-style MPC, followed by a re-randomization of the retrieved public key (still within MPC). They publicly reveal the output.

This process consumes d committees of secret-state roles, 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, etc. To conclude, we state the following informal theorem.

Theorem 5. *(informal) In the model of Benhamouda et al. [1] with a broadcast channel and mobile adversary, given anonymous PKE (for the target-anonymous channels) and a nontrivial weak RPIR protocol satisfying Definition 4, there exists a scalable evolving-committee proactive secret sharing scheme (ECPSS) as per [1, Def 2.3], tolerating any fraction $f < 1/2$ of corrupt parties.*

We note that the construction from [1] required other components (such as NIZK and sortition), but those can be replaced by the information-theoretic constructions from Gentry *et al.* [7]. We also comment that while the description above used public randomness, this can be replaced by the client generating the required randomness via a YOSO protocol. Also, we can use the same committees and the same techniques from [7] to get scalable secure-MPC for realizing arbitrary functions.

Theorem 6. *(informal) In the model of Benhamouda et al. [1] with a broadcast channel and mobile adversary, given anonymous PKE (for the target-anonymous channels) and a nontrivial weak RPIR protocol satisfying Definition 4, there exists scalable secure-MPC protocols for realizing any poly-time function, tolerating any fraction $f < 1/2$ of corrupt parties.*

4.2 Target Anonymous Channels from Mix-Nets

A different approach to setting up target anonymous communication channels is using Mix-Nets [3], i.e., by repeatedly shuffling and re-randomizing all the keys. This solution can be implemented simply as a sequence of public-state roles, each of which shuffles and re-randomizes all parties' public keys, then proves in zero knowledge that it did so correctly.

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 3.2.1: 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.

References

[1] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? IACR ePrint report 2020/464, 2020. URL: `https://eprint.iacr.org/2020/464`.

[2] Erica Blum, Jonathan Katz, Chen-Da Liu Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. *IACR Cryptol. ePrint Arch.*, 2020:851, 2020. URL: `https://eprint.iacr.org/2020/851`.

[3] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981. URL: `http://doi.acm.org/10.1145/358549.358563`, `doi:10.1145/358549.358563`.

[4] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th FOCS*, pages 41–50. IEEE Computer Society Press, October 1995. `doi:10.1109/SFCS.1995.492461`.

[5] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: secure multiparty computation with dynamic participants. *IACR Cryptol. ePrint Arch.*, 2020:754, 2020. URL: `https://eprint.iacr.org/2020/754`.

[6] Nikolaos Fountoulakis, Konstantinos Panagiotou, and Angelika Steger. On the insertion time of cuckoo hashing. *SIAM J. Comput.*, 42(6):2156–2181, 2013. See `https://arxiv.org/abs/1006.1231`. `doi:10.1137/100797503`.

[7] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. You only speak once: Secure MPC with stateless ephemeral roles. manuscript, 2020.

[8] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *30th ACM STOC*, pages 151–160. ACM Press, May 1998. `doi:10.1145/276698.276723`.

[9] Oded Goldreich and Leonid A. Levin. A hard-core predicate for all one-way functions. In *21st ACM STOC*, pages 25–32. ACM Press, May 1989. `doi:10.1145/73007.73010`.

[10] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Batch codes and their applications. In László Babai, editor, *36th ACM STOC*, pages 262–271. ACM Press, June 2004. `doi:10.1145/1007352.1007396`.

[11] Eyal Kushilevitz and Rafail Ostrovsky. Replication is NOT needed: SINGLE database, computationally-private information retrieval. In *38th FOCS*, pages 364–373. IEEE Computer Society Press, October 1997. `doi:10.1109/SFCS.1997.646125`.

[12] Eyal Kushilevitz and Rafail Ostrovsky. One-way trapdoor permutations are sufficient for non-trivial single-server private information retrieval. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 104–121. Springer, Heidelberg, May 2000. `doi:10.1007/3-540-45539-6_9`.

[13] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *21st ACM STOC*, pages 33–43. ACM Press, May 1989. `doi:10.1145/73007.73011`.

[14] Rasmus Pagh and Flemming Friche Rodler. Cuckoo hashing. In Friedhelm Meyer 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. `doi:10.1007/3-540-44676-1\_10`.

[15] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 299–310. ACM Press, November 2013. `doi:10.1145/2508859.2516660`.

[16] Stirling's approximation. `https://en.wikipedia.org/wiki/Stirling%27s_approximation`, accessed Oct 2020.

A Random-Index Oblivious-RAM

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 5 (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 generate 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'*.*[2]
- $\mathsf{Write}(\mathsf{cst}, i, x, \mathsf{SST}) \to \mathsf{SST}'$*: 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 [15] 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.

A.0.1 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 6 (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 committees of secret-state roles 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. 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 write *or read* 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 Secret-state committees) is necessary for every write. Second,

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

every party in the pool of participants must follow 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 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 secret-state-role 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.* [1]. The result is summarized by the following informal theorem:

Theorem 7. *In the model of Benhamouda et al. [1] with a broadcast channel and mobile adversary, given anonymous PKE (for the target-anonymous channels) and a nontrivial RORAM protocol satisfying Definition 6, there exists a scalable ECPSS scheme as per [1, 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.[3] 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).

[3]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.