Estimated Cost of Scaling Contact Discovery using Private Set Intersection

Kameron Shahabi Rolfe Schmidt

July 2023

Abstract

We evaluate the performance and operating cost of state of the art Private Information Retrieval (PIR) systems for contact discovery in messenger applications. After reviewing the PIR literature we implemented and evaluated the PSI/oo-PIR hybrid strategy of [8], which explicitly claims to scale mobile contact discovery to billions of users. We note that this system and other proposed PIR-based CDS solutions only provide Private Set Intersection (PSI), allowing a client to determine which phone numbers in its contacts are also in the server database but does not support retrieval of extra key data. Thus it provides less functionality than Signal's current CDS.

We implemented this system in C++ and Go and deployed it on a memory optimized 64-core virtual machine, similar to what might be chosen in production. The measured performance and resulting cost estimates are substantially better than those for earlier PIR systems, and we believe that with careful engineering and practical research we can bring them down further. Nevertheless they provide less functionality and have costs that are still orders of magnitude higher than the costs of operating hardware enclave based systems such as Signal Messenger's Contact Discovery Service [1].

1 Overview

Messaging apps are more useful when users have more connections on the app, so contact discovery - a feature that allows app users to find out who in their address book also uses the app - is a key usability feature. In principle this is a simple feature: we just need to join a table of all app users with a table of all of a user's contacts. For privacy-centric messaging apps, though, this is a challenge because both of these tables are highly sensitive. Revealing the list of all users is not only unwelcome from a business perspective, it can also harm users in environments where simply using a private messenger can be seen as suspicious. On the other hand, the set of contacts on a user's phone reveals a tremendous amount about the individual and must be protected. This is a hard problem that is not addressed in most messaging apps. Signal Messenger is one of the few apps that actively protects this data, and they do this using a side-channel hardened database inside a hardware enclave [1].

Private Information Retrieval (PIR) seems like a natural tool for this application. In Signal's first blog post about contact discovery [2] they rule out the use of PIR due to its high cost, but since that time research has advanced significantly [8, 5, 13] and recent systems claim to be able to handle billions of users. Thus a re-evaluation is in order.

This re-evaluation is the purpose of this document. We analyze a state of the art PIR system on realistic datasets and cloud-based servers, then use this analysis to estimate the performance and costs that an organization like Signal could expect from a PIR-based contact discovery solution.

2 Key Findings

We prototyped the state-of-the-art PSI/oo-PIR hybrid strategy detailed in [8] and ran it on a memory optimized 64-core R7g.16xlarge virtual machine. We confirm the findings of [8] that the system performs well for a single client. For a client with 1000 contacts to perform contact discovery on a database of 100 million records, an initial setup of 66 MiB requiring 153 seconds of server computation time is needed. Once the setup is complete, each CDS query for 1000 records requires 2.8 MiB of data transfer with 0.45 seconds latency.

These numbers are promising, but they only quantify retrieving contact phone numbers and this is not always sufficient. For example, Signal must retrieve account identifier and key data for each contact as well. To retrieve both phone numbers and this additional data, we estimate an initial setup requiring 990 MiB of data transfer with 7 minutes of server computation time. Each CDS query for 1000 records would then require 7.4 MiB of data transfer with 1.04 seconds latency.

Tuning certain parameters can offer a trade-off between communication and latency, which can serve to optimize monetary costs. Overall, for a pure contact discovery system, we estimate we can put together a low budget system for close to \$1 million per year, and a high performing system approaching \$2 million per year. For a contact discovery system that includes retrieving account identifier and key data, we estimate a low budget system at \$5.6 million per year, and a high performing system near \$10 million per year.

3 Background

PIR, proposed in [3], allows a client to query a database from a server without revealing any information about their query to the server. This may seem impossible at first glance but it can be as simple as sending each client a copy of an entire database and allowing them to retrieve data locally. For a messenger app of even modest scale, this naive PIR would be far too expensive and would also have the unfortunate side effect of revealing the application's full user database to all clients.

A number of methods have been proposed to improve upon this. Typically, single-server schemes, such as those proposed in [14, 7, 5], utilize randomness to hide a client's query when asking the server for a database record. Multi-server schemes [10, 4, 11], which use multiple non-colluding servers with copies of a database, achieve better performance. Each server can send data to a client that is indistinguishable from random, which the client can then use to reconstruct their desired record. Note, however, in practice hosting even 2 non-colluding servers may prove challenging [12].

We chose to analyze the PSI/oo-PIR scheme of [8] due to its scalability claims and promising experimental results. The scheme uses a 2-server PIR protocol named Checklist [10] as a subroutine. While other PIR protocols could replace Checklist, we choose to continue using it due to its existing implementation and promising online performance and communication [13]. Although Checklist suffers from high client storage, it is competitive with state-of-the-art PIR schemes and helps us bound online costs.

Checklist consists of an offline stage, during which an offline server generates random sets of database indices and parities of the data stored at those indices for each client. Clients store each set-parity pair as a hint. When a client wants to retrieve a record, they find a hint that contains the database index of that record. They then remove that index from the set, send it back to an online server, and receive its parity. Using the original set and the parity of that same set without the desired index, they can reconstruct the database record.

Overall, the system proposed in [8] works as follows. An offline server hashes each user's phone number and places it into a Cuckoo Filter (CF) [6] bucket based on the hashed value. Each bucket can hold multiple records. It also generates hints for each client, which include sets of bucket indices and parities for those buckets. The client can then use Checklist PIR to retrieve two possible buckets that one of their contacts may be in. After retrieving both of these buckets, they can determine whether their contact's phone number hash exists within one of them.

[8] also partitions the CF, allowing clients to use PIR only on relevant portions of the database. To prevent leaking which partitions are of interest to clients, the online server must receive a certain number of dummy queries based on a balls-to-bins analysis.

4 PSI versus PIR

The system we analyze from [8] is a private set intersection (PSI) scheme which uses PIR as a subroutine. PSI aims to find the intersection between two sets without leaking any information about members not in one of these two sets. In this section, we present the trade-offs between using PIR on a flat database versus conducting this PSI protocol.

A concern when considering direct PIR on a database is that clients must retrieve data using an array index. However, clients only know their contact phone numbers, not the physical location of their record in a server database. To allow for this, the online server would need some data structure on top of the database to perform look-ups without leaking information. The PSI scheme does not face this issue, as the CF acts as this data structure. The hash of a phone number determines the physical location of its bucket in the CF, therefore clients can use the hashes of their contact's phone numbers to make PIR queries to retrieve the needed CF buckets. In addition, the PSI scheme achieves superior online latency and throughput to PIR on a flat database. This comes with a trade-off in online and offline communication.

As noted above, users of a contact discovery service may need to retrieve additional account identifier and key data for each contact. The CF described in [8] only stores phone number hashes in each bucket. Data stored along with phone number hashes in a CF is susceptible to vulnerabilities. A client could pull a bucket where one of their contacts resides, then access data from other users in that same bucket. This, along with the problem that the query may require processing beyond simple retrieval, is an area for future research.

5 Prototyped System

We prototyped the system described in [8], and compared it to a baseline motivated by the discussion from Section 4. We consider the following methods:

DB-PIR We run Checklist PIR on a partitioned database of 2^{27} phone numbers. Each record is a 5-byte binary representation of a phone number. We use the balls-to-bins analysis from [8] to send the appropriate amount of dummy queries to all partitions, to avoid leaking which partitions are of interest to the clients.

PSI/oo-PIR The scheme from [8]. We run Checklist PIR on a partitioned CF with 2^{26} buckets. Each bucket holds three 4-byte phone number hashes. This allows us to store 2^{27} phone numbers in the CF without reaching full capacity. Each phone number can be placed into one of two buckets, so Checklist queries are for two entire buckets (24-bytes total). For the purpose of a cost analysis, we do not use an oblivious psuedorandom function (OPRF) in our implementation, and instead hash the raw phone numbers into the CF. While OPRFs provide more security [9], we do not expect their addition to significantly affect costs.

Both systems are implemented using Golang and C++. For each system, we evaluate a client querying for 1000 phone numbers. We ran the client and servers on the same memory optimized 64-core AWS R7g.16xlarge virtual machine, so these costs do not include networking overhead.

6 Performance

We analyze the performance of PSI/oo-PIR. First, we offer theoretical communication cost estimates, independent of any code optimizations. We then provide communication and performance results attained using our prototype implementation of the strategy. We also take a look at costs for retrieving additional data along with phone numbers. Finally, we analyze throughput and use our results to estimate monetary costs of using a PIR solution for Signal's CDS.

6.1 Theoretical Communication Costs

We start with an estimate of communication between client and server based on the complexity analysis from [8]. We present these numbers as hard lower bounds for PSI/oo-PIR. Implementation details will lead to slightly different costs.

Number of partitions	Offline Communication [MiB]	Online Communication [KiB]
2	16.97	1761.81
2^6	96.00	1986.16

Table 1: Communication costs to conduct PSI/oo-PIR from [8] between a database with 2^{27} users and a client with 2^{10} contacts. We assume a security parameter $\lambda = 128$.

Table 1 shows values for varying amounts of database partitions. Note that offline communication is a one time cost each time the server is booted up, while online communication is necessary every time a user needs to retrieve their contacts. Note that as the number of partitions increases the communication costs increase too. This is our first look at a performance trade-off which we explore in the following section.

6.2 Prototype Performance

We offer results on prototype implementations of two different methods, DB-PIR and PSI/oo-PIR. See Section 5 for details.

6.2.1 General PSI Results

We compare PSI/oo-PIR with the DB-PIR baseline discussed in Section 5. Table 2 shows estimates for conducting PSI for a single client with 1000 contacts on different partition sizes.

Method, Partitions	**Offline Time**	**Offline Communication [MiB]**	**Online Time**	**Online Communication [KiB]**	
				In-bound	Out-bound
DB-PIR, $N_{part} = 16$	5m43s	19.45	1.69s	1831.81	86.41
DB-PIR, $N_{part} = 32$	4m53s	27.50	1.01s	1987.56	101.41
DB-PIR, $N_{part} = 64$	5m10s	38.90	0.821s	2440.81	124.53
PSI/oo-PIR, $N_{part} = 16$	2m48s	33.00	0.896s	2075.23	254.11
PSI/oo-PIR, $N_{part} = 32$	2m49s	46.69	0.645s	2368.46	290.02
PSI/oo-PIR, $N_{part} = 64$	2m33s	66.00	0.446s	2591.02	345.47

Table 2: Communication and performance costs for conducting CDS on one client. Offline numbers measure the offline server's hint generation. Online metrics report the server's ability to handle 1000 PIR requests, including appropriate number of dummy queries based on a balls-to-bins analysis [8]

.

We see that PSI/oo-PIR achieves close to 2x better latency compared to DB-PIR, with a slight increase in online communication. Offline communication for both methods are significant, but may be reducible with new PIR schemes, as current literature introduces replacements to Checklist that aim to improve upon this [13].

6.2.2 Retrieving Contact Data

As discussed in Section 4, clients may need to retrieve additional account identifier and key data for each of a client's contacts. To simulate how this would affect PIR, we run the previous experiment with the following modifications. For DB-PIR, we use 61-byte records, which represent 5 phone number bytes and 56 additional data bytes. For PSI/oo-PIR, we store 56-bytes of data alongside each phone number hash. Note that entries are placed into buckets based on the phone number hash. The extra data does not affect this, so indexing can still be done using phone numbers. We assume that account identifier and key data can be stored in the CF securely, and leave confirming this for future work.

Table 3 shows a significant increase in offline communication when retrieving additional data. We expect this, as hint size scales linearly with record length. Similarly, out-bound communication is larger for the online server as more data must be sent to the client. In-bound communication sees little change, as clients still request indices in DB-PIR, and phone number hashes in PSI/oo-PIR.

6.3 Monetary Costs

6.3.1 Server Costs

We use a target of 100,000 queries per second for a database with 2^{27} records, as this is the order of magnitude of the throughput achievable by the SGX-based contact discovery system deployed by Signal Messenger on one 24-core Intel Icelake processor at a cost of \$20183 per year. To determine the server capacity necessary for PIR systems to match that rate, we use the online times presented in Tables 2 and 3 to estimate throughput. We scale that rate and estimate the number of 64-core AWS R7g.16xlarge virtual machines that are necessary to achieve 100,000 queries per second. We then calculate the annual server cost using AWS's rate of \$3.43/hour for each server. Results are presented in Table 4.

Method, Partitions	**Offline Time**	**Offline Communication [MiB]**	**Online Time**	**Online Communication [KiB]**	
				In-bound	Out-bound
DB-PIR, $N_{part} = 16$	9m20s	237.00	2.00s	1831.81	1054.16
DB-PIR, $N_{part} = 32$	8m47s	335.50	1.58s	1987.56	1237.16
DB-PIR, $N_{part} = 64$	8m47s	474.52	1.05s	2440.81	1519.28
PSI/oo-PIR, $N_{part} = 16$	7m37s	495.00	2.01s	2071.40	3804.61
PSI/oo-PIR, $N_{part} = 32$	7m37s	700.00	1.37s	2369.99	4353.05
PSI/oo-PIR, $N_{part} = 64$	7m13s	990.00	1.04s	2591.72	5183.44

Table 3: Communication and performance of conducting the CDS for one client, when we consider retrieving account identifier and key data for each contact.

Note that using fewer partitions allows us to run multiple instances of the protocol on one 64-core server. For example, PSI/oo-PIR w/ additional data on 16 partitions requires around 200 protocol instances to achieve a rate of 100,000 queries per second. However, because it uses 16 partitions, 4 instances can run on a single 64-core machine at a time, meaning only 50 of these machines would be needed to meet the throughput target.

Method, Partitions	**Annual Server Costs**	**Annual Server Costs w/ additional data**
DB-PIR, $N_{part} = 16$	\$1,310,158	\$1,502,340
DB-PIR, $N_{part} = 32$	\$1,553,908	\$2,437,504
DB-PIR, $N_{part} = 64$	\$2,528,910	\$3,229,692
PSI/oo-PIR, $N_{part} = 16$	\$700,782	\$1,553,908
PSI/oo-PIR, $N_{part} = 32$	\$1,005,470	\$2,102,347
PSI/oo-PIR, $N_{part} = 64$	\$1,371,096	\$3,168,755

Table 4: Annual AWS server costs necessary for handling 100,000 queries per second.

Our analysis above assumes that serving 100,000 queries is equivalent to serving 100 clients without concurrency. However, we may be able to improve on this rate and cut down on server costs. Table 5 shows online times for handling 10 clients concurrently. These preliminary numbers suggest that we can hope to save up to 50% on online costs through engineering solutions.

Method	Online Time for 10 Clients
DB-PIR	6.55s
PSI/oo-PIR	3.20s
DB-PIR w/ additional data	6.74s
PSI/oo-PIR w/ additional data	3.69s

Table 5: Time taken for one server to satisfy 10,000 total PIR requests coming from 10 client threads. Each method uses 64 partitions.

6.3.2 Communication Costs

Table 6 presents monetary costs for communication based on the results from Section 6.2. We estimate using a rate of \$0.05/GB out-bound, the bulk rate of the AWS machine we use. For offline costs, we present the one-time cost of sending hints to 2^{27} users. For online costs, we calculate the annual cost of retrieving 100,000 contacts every second.

Although we did not prototype an update system, in practice we would expect close to 10 database updates per second, invalidating previous hints. Current literature does include discussions on updates

Method, Partitions	**Offline Cost**	**Yearly Online Cost**	**Offline Cost w/ additional data**	**Yearly Online Cost w/ additional data**
DB-PIR, $N_{part} = 16$	\$136,867	\$13,625	\$1,667,742	\$166,219
DB-PIR, $N_{part} = 32$	\$193,514	\$15,990	\$2,360,871	\$195,075
DB-PIR, $N_{part} = 64$	\$273,734	\$19,635	\$3,339,137	\$239,560
PSI/oo-PIR, $N_{part} = 16$	\$232,216	\$40,068	\$3,483,252	\$599,910
PSI/oo-PIR, $N_{part} = 32$	\$328,551	\$45,730	\$4,925,810	\$686,388
PSI/oo-PIR, $N_{part} = 64$	\$464,433	\$54,473	\$6,966,504	\$817,324

Table 6: Monetary costs for PIR schemes. The first two columns are costs for retrieving phone numbers. Columns labeled "w/ additional data" include the costs of retrieving account identifier and key data. Yearly online costs are for serving 100 clients each with 1000 contacts every second.

[8]. From this, we expect offline costs to be substantial, but of a similar magnitude to the costs reported below.

7 Conclusions

Research in PIR, particularly for systems aimed at providing contact discovery services, has advanced significantly in recent years. By implementing one such state-of-the-art system and deploying it on realistic production hardware, we are able to confirm that latency of a PIR-based contact discovery service is now acceptable, even for systems at a scale of billions of users. There is more to performance than latency, though, and we must analyze the performance and cost of a system that is simultaneously serving millions or even billions of users. Here we see that the costs are still orders of magnitude higher than the cost of obtaining similar performance on a hardware enclave based system like the one developed at Signal Messenger. Furthermore, we note that since the PIR system we analyzed relies on multi-party computation, in practice it still may end up running inside attested enclaves as a defense in depth mechanism to ensure parties are not colluding. So we see that while PIR does seem viable as a contact discovery solution, it is unclear if organizations will find that the practical security benefits they provide over existing alternatives justify the substantial additional costs.

References

[1] Technology deep dive: Building a faster oram layer for enclaves.

[2] Technology preview: Private contact discovery for signal.

[3] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, nov 1998.

[4] Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 44–75, Cham, 2020. Springer International Publishing.

[5] Alex Davidson, Gonçalo Pestana, and Sofía Celi. Frodopir: Simple, scalable, single-server private information retrieval. Cryptology ePrint Archive, Paper 2022/981, 2022. https://eprint.iacr.org/2022/981.

[6] Bin Fan, Dave G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, CoNEXT '14, page 75–88, New York, NY, USA, 2014. Association for Computing Machinery.

[7] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. One server for the price of two: Simple and fast single-server private information retrieval. Cryptology ePrint Archive, Paper 2022/949, 2022. https://eprint.iacr.org/2022/949.

[8] Laura Hetz, Thomas Schneider, and Christian Weinert. Scaling mobile private contact discovery to billions of users. Cryptology ePrint Archive, Paper 2023/758, 2023. https://eprint.iacr.org/2023/758.

[9] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1447–1464, Santa Clara, CA, August 2019. USENIX Association.

[10] Dmitry Kogan and Henry Corrigan-Gibbs. Private blocklist lookups with checklist. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 875–892. USENIX Association, August 2021.

[11] Arthur Lazzaretti and Charalampos Papamanthou. Treepir: Sublinear-time and polylog-bandwidth private information retrieval from ddh. Cryptology ePrint Archive, Paper 2023/204, 2023. https://eprint.iacr.org/2023/204.

[12] Yehuda Lindell, David Cook, Tim Geoghegan, Sarah Gran, Rolfe Schmidt, Ehren Kret, Darya Kaviani, and Raluca Ada Popa. The deployment dilemma: Merits challenges of deploying mpc.

[13] Muhammad Haris Mughees, Sun I, and Ling Ren. Simple and practical amortized sublinear private information retrieval. Cryptology ePrint Archive, Paper 2023/1072, 2023. https://eprint.iacr.org/2023/1072.

[14] Mingxun Zhou, Andrew Park, Elaine Shi, and Wenting Zheng. Piano: Extremely simple, single-server pir with sublinear server computation. Cryptology ePrint Archive, Paper 2023/452, 2023. https://eprint.iacr.org/2023/452.