Scaling Mobile Private Contact Discovery to Billions of Users*

Laura Hetz[1], Thomas Schneider[1], and Christian Weinert[2]

[1] ENCRYPTO, Technical University of Darmstadt, Germany
{laura.hetz, schneider}@encrypto.cs.tu-darmstadt.de
[2] Royal Holloway, University of London, United Kingdom
christian.weinert@rhul.ac.uk

Abstract. Mobile contact discovery is a convenience feature of messengers such as WhatsApp or Telegram that helps users to identify which of their existing contacts are registered with the service. Unfortunately, the contact discovery implementation of many popular messengers massively violates the users' privacy as demonstrated by Hagen et al. (NDSS '21, ACM TOPS '23). Unbalanced private set intersection (PSI) protocols are a promising cryptographic solution to realize mobile *private* contact discovery, however, state-of-the-art protocols do not scale to real-world database sizes with billions of registered users in terms of communication and/or computation overhead.

In our work, we make significant steps towards truly practical large-scale mobile private contact discovery. For this, we combine and substantially optimize the unbalanced PSI protocol of Kales et al. (USENIX Security '19) and the private information retrieval (PIR) protocol of Kogan and Corrigan-Gibbs (USENIX Security '21). Our resulting protocol has a total communication overhead that is sublinear in the size of the server's user database and also has sublinear online runtimes. We optimize our protocol by introducing database partitioning and efficient scheduling of user queries. To handle realistic change rates of databases and contact lists, we propose and evaluate different possibilities for efficient updates. We implement our protocol on smartphones and measure online runtimes of less than 2 s to query up to 1 024 contacts from a database with more than two billion entries. Furthermore, we achieve a reduction in setup communication up to factor 32× compared to state-of-the-art mobile private contact discovery protocols.

Keywords: mobile contact discovery · PSI · PIR.

1 Introduction

The number of users of mobile messengers such as WhatsApp, Telegram, and Signal has been rising for over a decade. In 2020, WhatsApp reached two billion monthly active users [25]. Messengers connect these users by presenting them a selection of their existing address book contacts who are registered with the same

* Please cite the conference version of this paper published at ESORICS'23 [38].

service. This convenient feature is called *mobile contact discovery* and requires matching users' contact lists with the service's database. The address book of users is also checked regularly to ensure an up-to-date list of possible contacts.

However, a recent survey [33] showed that five out of eleven studied messengers, including WhatsApp and Telegram, implement contact discovery by obtaining their users' contact lists in plaintext. Thus, service providers not only learn about mutual contacts, but also information of unregistered contacts. Based on this, the entire social graph of users, possibly containing sensitive information, can be inferred. Even if a user has never signed up with a messenger or social media platform, contact discovery services might have already stored their personal data. Meta, WhatsApp's and Facebook's parent company, which acquired WhatsApp in 2014 for 16 billion USD [59], has acknowledged this with a tool that lets non-users check, delete, and block their data from several of their services' contact discovery databases, however, excluding WhatsApp [29]. It is currently unclear how these block lists are implemented and which privacy implications they entail. Due to the availability of information, access to it might be enforced legally (by governments) or illegally (by hackers).

A naive approach used by some messengers to protect privacy is to apply a cryptographic hash function before uploading phone numbers. However, due to the clearly defined structure and low entropy of phone numbers, the reversal of a single hash is possible in less than 0.1 ms on commodity hardware [33]. The privacy-preserving messenger Signal thus uses hardware enclaves, specifically Intel SGX, to securely realize mobile contact discovery. However, the security of enclaves is not trivial as even code without vulnerabilities can be subject to various types of attacks [73, 21, 65, 9].

The cryptographic approach for mobile *private* contact discovery is to apply protocols for unbalanced *private set intersection (PSI)*. In our setting, the server's user database DB and the client's phone contacts X each represent one set ($|X| \ll |DB|$) while only the client learns about the mutual elements.

Recent works [15, 28, 58, 66] show promising results for fast and communication-efficient PSI in different use cases, but are still impractical for mobile private contact discovery at large scale due to the required online computation performed by the server. With over two billion WhatsApp users [25], the unbalanced PSI protocol by Cong et al. [15] requires less than 80 MiB of total communication, but more than 35 seconds online time with multi-threading (T=24 threads) to query $|X| = 2^{10}$ client contacts. Hence, the protocol by Kales et al. [44] based on oblivious pseudorandom functions (OPRFs) is still state of the art for private contact discovery due to its fast online runtimes (linear in $|X|$ and less than 3 seconds for $|DB| = 2^{28}$, $|X| = 2^{10}$ [44]) and optimization for mobile devices. However, this protocol has setup communication and client storage costs linear in the database size – 8 GiB for $|DB| = 2^{31}$ – which also makes it impractical for large-scale messengers. To make such protocols viable, communication sublinear in the database size is necessary. The authors of [44, 23] thus recommend using a protocol for multi-server private information retrieval (PIR) in PSI to achieve sublinear communication.

Our Contributions. In this work, we make big steps towards truly practical mobile private contact discovery by reducing the setup communication to be sublinear in the size of the server's database. The authors of [23] already achieved this, however, their protocol requires online computation linear in the database size. We achieve both, total communication and online computation sublinear in the database size. For this, we survey the current literature and select the offline-online PIR (OO-PIR) protocol by Kogan and Corrigan-Gibbs [47] as a building block for its sublinear complexities. By combining the state-of-the-art protocol for unbalanced PSI on mobile devices [44] with OO-PIR [47], we obtain an asymptotically and concretely efficient mobile private contact discovery protocol.

We further extend our protocol to handle large sets, i.e., databases with up to $|DB| = 2^{31}$ items, to meet the requirements of real-world messengers. To our knowledge, we are the first to consider a database with more than a billion records in unbalanced PSI (8× more than related works [46, 23, 67, 44, 71]). For this setting, we reduce the setup communication by up to factor 32× over the state-of-the-art protocol of [44]. To prevent the inefficient processing of a large database as a whole, we let multiple instances of the PIR protocol operate on smaller database partitions. Queries to these partitions should not reveal to the server which database partitions are of interest to the client. Therefore, we schedule these queries based on a balls-to-bin analysis similar to [61, 63, 23]. This reduces communication by a factor up to 24× compared to the naive approach of sending the maximum possible number of queries to all partitions to hide the information which partitions are of interest.

We also study ways to efficiently handle updates to client contact lists and server databases. For this, we evaluate solutions for dynamic databases proposed by recent literature [23, 44, 47, 52] and improve on their ideas for our protocol design. With less than 3 MiB/day for processing a realistic number of 2^{21} daily updates [32, 33], our resulting protocol has the lowest communication cost.

Finally, we implement our protocol on smartphones to demonstrate feasibility and obtain concrete runtime measurements in realistic WiFi and LTE network settings. Over WiFi, we achieve an online runtime of less than 2 seconds for $|DB| = 2^{31}$ database records and $|X| = 2^{10}$ phone contacts. Further highlights of our implementation include containerized builds for improved reproducibility, multi-threading for additional runtime improvements, and significant optimizations of the original PIR implementation of [47]. Our implementation "DISCO" (short for "DIScover COntacts") is available at https://encrypto.de/code/disco.

To summarize, our main contributions are as follows:

- New mobile private contact discovery protocol based on unbalanced PSI [44] and private information retrieval (PIR) [47] with sublinear total communication and online runtime.
- Reproducible, multi-threading-capable implementation on mobile clients.
- Large-scale evaluation for databases with more than two billion records and online runtime of less than 2 seconds over WiFi.
- Efficient update strategy with less than 3 MiB/day of additional communication costs.

2 Preliminaries

In this section, we describe the basic concepts used in our work, specifically protocols for oblivious pseudorandom function (OPRF), private set intersection (PSI), and private information retrieval (PIR). We also explain Cuckoo filters (CFs), a probabilistic data structure used in our protocol.

Oblivious Pseudorandom Function. An oblivious pseudorandom function (OPRF) is a secure two-party computation (STPC) protocol where the computed public function f is a keyed pseudorandom function (PRF). Party P_2 inputs key k and P_1 inputs a value x for which P_1 obtains the PRF output $f_k(x)$. Both parties stay oblivious about the other party's input and only P_1 obtains the OPRF output. OPRF constructions can be used to realize PSI protocols, as shown in a variety of works, including [27, 34, 62, 46, 44]. We focus on the Naor-Reingold PRF (NR-PRF) [57] and PRFs that evaluate block ciphers such as AES and the STPC-friendly cipher LowMC [1] using Yao's garbled circuit (GC) [74], a generic protocol for STPC. These OPRFs offer malicious client security [62, 46] and their implementations were already optimized for mobile devices [44]. While recent works [68, 66, 12] improve over our selected OPRFs, we leave their evaluation as future work and focus on reducing the setup communication and client storage of the state-of-the-art protocol for mobile private contact discovery.

Cuckoo Filter. A Cuckoo filter (CF) is a probabilistic data structure for fast membership testing. A CF stores tags (i.e., short representations of items), where each tag is located in one of h possible buckets and each bucket contains up to b tags. The tag of x with length v is computed using hash function H_t: $t_x = H_t(x) \in \{0,1\}^v$ and its possible positions are determined by h hash functions [26]. CFs are similar to Bloom filters (BFs) [7], but have better performance, reduced storage, and allow item deletion. Hash collisions for tags can result in false positives. We follow the parameter recommendations in [44] with bucket size $b = 3$ and tag size $v = 32$ for a false positive probability (FPP) of $\epsilon \leq 2b/2^v \approx 2^{-29}$.

Private Set Intersection. In protocols for private set intersection (PSI), two parties P_1 and P_2 hold sets X_1 and X_2, respectively. They want to know their mutual items (i.e., $X_1 \cap X_2$) without revealing anything else about their sets. State-of-the-art PSI protocols for large sets build on the oblivious key-value store (OKVS) data structure [28, 58, 66]. However, they require online communication linear in the size of the larger set. Another line of work on *unbalanced* PSI based on fully homomorphic encryption (FHE) [14, 13, 15] has a small communication footprint, but is not well suited for large-scale contact discovery as the server online computation is linear in the database size for each client.

In this work, we thus focus on unbalanced OPRF-based PSI protocols [34, 62, 20, 46, 44] for mobile private contact discovery. The high-level idea requires server S, holding the larger set DB, to sample a secret key k and to encrypt its input using a PRF and k to obtain $PRF_k(DB[i])$ for $i \in \{1, \ldots, |DB|\}$. This encrypted set is sent to the client C who stores it. Both parties then run the corresponding OPRF protocol on C's input $X[i]$ for $i \in \{1, \ldots, |X|\}$ and S's key k such that C obtains the encrypted values $PRF_k(X[i])$ and locally checks which of them are contained in the server's encrypted set. The performance of

such PSI protocols is great in the online phase (independent of $|DB|$), but suffers from high setup communication and client storage requirements (linear in $|DB|$), which prohibits applicability for mobile private contact discovery at large scale. In this work, we make significant steps towards practicality by replacing the download in the setup phase with a protocol for PIR for reduced communication and storage requirements.

Private Information Retrieval. Protocols for PIR enable a client C to privately obtain a record from a public database with N_{PIR} records while the server stays oblivious about the requested item. The server's computational cost must be inherently linear in the database size, as the server would otherwise learn which elements the client is not interested in [6]. PIR with preprocessing is thus critical to achieve online complexities sublinear in the database size N_{PIR} by shifting the linear costs to an offline phase. We comprehensively surveyed single- and multi-server PIR protocols with preprocessing for our use case (cf. § A). The state-of-the-art single-server PIR protocols [54, 56, 37, 22] are based on FHE: The client uses FHE to hide their query from the server while also enabling the server to answer their query under encryption. In a large-scale deployment scenario, the client-independent preprocessing in [22, 37] offers a significant advantage as server costs otherwise depend on the high number of clients. While these protocols are most promising in the single-server setting, the parties still perform online computation linear in N_{PIR}. Also, online communication costs with query batching are impractically high at large scale. Moreover, FHE-based protocols have yet to be implemented and evaluated for this use case on mobile devices.

In the setting with multiple non-colluding servers (see § 4.2 for a detailed discussion), different strategies have been proposed [10, 18, 47, 69, 52, 31]. We select the two-server OO-PIR protocol in [47] for its sublinear online complexities (communication in $O(\log N_{PIR})$ and computation in $O(\sqrt{N_{PIR}})$), existing mobile implementation, and database update strategies [47, 52]. We refer to the required servers as offline server S_{off} and online server S_{on}, and give an informal protocol description of the protocol in [47]: In the *offline phase*, S_{off} randomly samples N_{Sets} sets, each containing $\sqrt{N_{PIR}}$ database indices, calculates the parity of each set, and sends sets and parities as hints to client C. The parameter $N_{Sets} = \lambda\sqrt{N_{PIR}}\log 2$ is chosen to ensure that any database index appears in at least one set with overwhelming probability [47] based on the statistical security parameter λ. In the *online phase*, the client finds a set that contains the index *idx* they want to query, and they remove it from the set in a process called *puncturing*, i.e., $Set'_i = Set_i \setminus \{idx\}$. The client sends the punctured set Set'_i to the online server S_{on}, which returns the parity of the received set. The requested database record $DB[idx]$ is reconstructed from the punctured and the unpunctured sets' parities, i.e., $y_{idx} = p_{Set_i} \oplus p_{Set'_i}$. Reusing the set Set_i leaks information about the queries to the server. Thus, the client generates a new set containing the requested index to ensure that the set remains random while at least one set still contains index *idx*. The client obtains the parity for the new set by puncturing it, requesting the punctured set's parity from the offline server, and adding the database record they just retrieved for *idx* to the parity.

The sublinear communication cost of [47] is achieved by transmitting the sets in compressed form as set keys, which are puncturable PRF keys.

3 Related Work

We focus our discussion of related works on *unbalanced* PSI for mobile private contact discovery. Nevertheless, we acknowledge the existence of further unbalanced PSI protocols based on FHE [14, 13, 15], which are not suitable for large-scale contact discovery because the server performs computation linear in the large database for each client in the online phase (cf. § 2).

Our protocol is based on the mobile private contact discovery protocols in [46, 23, 44]. In [46], the authors improve PSI for the unbalanced setting and mobile clients by shifting the required setup computation and communication costs that depend linearly on the database size $|DB|$ to a novel precomputation phase. They further reduce the communication and storage costs by storing the larger set in a Bloom filter, a probabilistic data structure similar to Cuckoo filters (CFs). The authors of the state-of-the-art unbalanced PSI protocol for mobile private contact discovery [44] build on the promising results of [46] and optimize the performance as well as communication cost of two OPRF-based PSI protocols with malicious client security. By integrating and optimizing a two-server PIR protocol [47] in the protocol design of [44], we achieve a reduction in setup communication by 32× at only marginally higher online costs (cf. § 5).

A combination of two-server PIR and PSI for private contact discovery was first proposed in [23] with PIR-PSI. Their protocol also achieves sublinear communication complexity in the database size. However, due to a lack of PIR-preprocessing, the servers in PIR-PSI perform online computation linear in the database size for each query, which prohibits large-scale deployments. Furthermore, the constructions and base protocols differ: The authors of [23] improve the performance of the balanced PSI protocol of [48] by running PIR based on distributed point functions (DPFs) [10, 11] to reduce the input set sizes. Instead, we use OO-PIR by [47] to reduce the communication of unbalanced OPRF-based PSI [44] for mobile devices. PIR-PSI, similar to our work, models query scheduling as a ball-to-bins problem (cf. § 4.3). In contrast to our protocol, PIR-PSI requires inter-server online communication (32 kiB for $|\mathrm{X}| = 2^{10}$ for each client), which incurs 8× higher financial costs compared to computation [42].

In addition to mobile contact discovery, contact tracing and compromised credential checking (C3) are two other use cases for our protocol. Epione [71] combines public key (PK)-based PSI with keyword-PIR for efficient privacy-preserving contact tracing. Epione also achieves sublinear online communication but requires online computation linear in $|DB|$ and has a high online inter-server communication cost. Protocols for C3 are deployed in web browsers to check if *one* credential is in a database of leaked credentials ($|DB| \approx 12.5$ billion [47, 31, 72]). For this, PK-based PSI protocols are used in practice; however, to reduce communication overhead, a hash prefix is leaked to the server to indicate which partition of the encrypted database must be downloaded [50, 70]. PIR protocols such as [47, 31], as well as our work, could be used to mitigate attacks that leverage this leakage.

4 Our Protocol

Our protocol provides computational security and assumes a semi-honest setting with two non-colluding servers, S_{off} and S_{on} (we discuss malicious client security in § 4.2). Client C inputs their set of phone contacts X of size $|X|$, and the messaging service inputs their user database DB of size $|DB|$, which is encrypted and encoded in a Cuckoo filter CF. We divide the database of the PIR protocol into N_{Part} partitions, where $CF_p \in \{CF_1, \ldots, CF_{N_{Part}}\}$, to allow for large databases. This requires a scheduling of queries to reduce communication while preventing leakage (cf. § 4.1).

Our protocol is divided into *base*, *setup*, and *online* phase, as introduced by [46]. The client-input-independent parts of the protocol, i.e., base and setup phase, are considered to be *offline*. The base phase of our protocol is input-independent and contains the OPRF precomputation between C and S_{off} as well as the server's generation of the secret key k. This phase is identical to the base phase in [44]; it has a communication complexity of $O(|X|^{pre})$ and allows the client to check up to $|X|^{pre}$ contacts in the online phase. We split the server-input-dependent setup phase into client-independent setup and per-client setup. The server setup is run only once and includes the encoding of the database and CF creation by S_{off}. S_{on} receives no cleartext data, only the CF containing the encrypted and hashed values. The per-client setup has to be executed once for each client and consists of the offline phase of our extended PIR protocol [47].

Our protocol's online phase combines those of [44] and [47]. C and S_{off} run the OPRF protocol on their respective inputs $x_i \in X$ and k, and C obliviously obtains $e_i = PRF_k(x_i)$ for $i \in \{1, \ldots, |X|\}$. C simulates the offline server's data placement in the CF for their encrypted inputs to learn which CF buckets to retrieve via PIR. The encrypted value e_i is in one of two possible CF buckets if $x_i \in DB$, and C retrieves both to locally check if $e_i \in CF$, i.e., $x_i \in DB$. PIR queries to S_{off} and S_{on} are generated for each index based on the stored hints for CF_{Part}. Sending only those actual queries reveals to the server which partitions interest the client. We show in § 4.1 how to avoid this by sending dummy queries in a communication-efficient manner.

4.1 Database Partitioning and Querying

We assume messenger services with up to $|DB| = 2^{31}$ users and CFs with up to $N_{CF} = 2^{\lceil \log_2(|DB|/b) \rceil} = 2^{30}$ buckets. To our knowledge, this work is the first to consider a database of this size in the context of mobile private contact discovery. Our selected PIR protocol [47] requires offline computational cost linear in the database size and parties have to process sets with $\sqrt{N_{PIR}}$ items. Using the CF as PIR database (i.e., $N_{PIR} = N_{CF}$) thus leads to poor performance and high memory requirements. Additionally, sets of this size are not supported by the existing OO-PIR implementation [47].

We avert these limitations by partitioning the database and running the protocol on smaller database partitions at a time. The PIR database size now depends on the number of partitions N_{Part} where $N_{PIR} = N_{CF}/N_{Part}$. A smaller number of partitions generally requires less communication since less PIR execu-

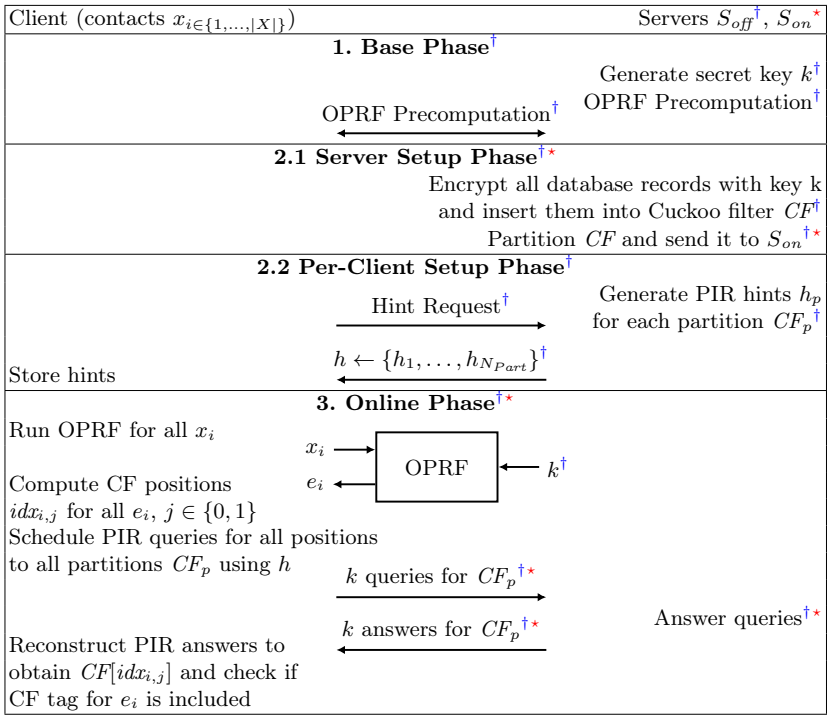


Fig. 1: Protocol phases for communication-efficient OPRF-based unbalanced PSI with two-server PIR. Offline server S_{off} marked with †, online server S_{on} with *.

tions are needed, but higher computational cost due to the increased database size N_{PIR}. We consider this trade-off in the parameter selection for our partitioning. Database partitioning further allows us to distribute the workload between multiple servers to improve the performance and scalability of our protocol.

With database partitioning, if the client would only query the desired indices, the server would learn which partitions are of interest to the client. This leakage could easily be prevented with dummy queries to all other partitions to conceal the actual queries. However, this naive approach requires in the worst case $2|X|N_{Part}$ queries – more than 73 MiB of online communication for $|DB| = 2^{31}$, $|X| = 2^{10}$.

The literature presents various approaches for scheduling queries [43, 36, 3], also called batching, to reduce communication or computational cost. In [75], the feasibility of using batching techniques in OO-PIR protocols is studied, and a lower bound for communication and time in the preprocessing phase of $t \cdot r = \Omega(N_{PIR}k)$ is proven for batch size k, hint size r, and online time t. The authors show that server performance improves at the cost of higher client runtime, and communication. They conclude that the benefits of PIR protocols in the offline-online model and batching are not compatible. Probabilistic batch codes [3] in OO-PIR achieve this lower bound, but due to the high storage requirements and client costs of this technique, we conclude that (probabilistic) batch codes are not practical for our use case.

Instead of optimizing the query scheduling with batch codes, we focus on leveraging our protocol's underlying data structure: Cuckoo filters. Items in a CF are distributed uniformly under the assumption of uniformly random hash functions, and that the items are chosen independently from the hash functions and from each other (note that our items are encrypted set elements) [24]. This allows us to represent the query scheduling as a balls-to-bin problem, where we ask for the maximum number of balls in any bin when placing n balls independently into β bins chosen uniformly at random. We assume $\beta = N_{Part}$ bins (i.e., DB partitions) and $n = 2|X|$ balls (i.e., queries), and use Eq. (1) based on [64, 63] to calculate the probability p of any bin containing more than k balls after inserting n balls into β bins.

$$p = 1 - \left(\sum_{i=0}^{k-1} \binom{n}{i} \cdot \left(\frac{1}{\beta}\right)^{i} \cdot \left(1 - \frac{1}{\beta}\right)^{n-i} \right)^{\beta}. \tag{1}$$

We require this probability to be negligible, i.e., $p < 2^{-40}$. Based on this formula, we determine k via a Mathematica script as the maximum number of queries made to each of N_{Part} partitions for $2|X|$ actual queries, and achieve a reduction in communication in the worst case by up to factor 24×, and only require 3 MiB instead of 73 MiB for $|DB| = 2^{31}$, $|X| = 2^{10}$. We note that [64] provide a closed-form solution for the balls-to-bin problem (but with an unspecified constant γ), which we leverage for our asymptotic analysis in § 4.2.

4.2 Complexity & Security Analysis

We now discuss our protocol's communication complexity and analyze its security.

Complexity. Our protocol consists of OPRF and PIR invocations. The considered parameters are the client contact list with size $|X|$ and at most $|X|^{pre}$ precomputed entries; the server database has $|DB|$ entries, which are processed in our protocol in N_{Part} database partitions of size N_{PIR}. The asymptotic communication complexity for OPRF is the same as in [44], namely $O(|X|^{pre})$ in the base and $O(|X|)$ in the online phase. In our PIR protocol, each CF bucket with $b = 3$ tags of size $v = 32$ bit is one record of length $\ell = v \cdot b = 96$ bit. The concrete communication cost for running PIR on N_{Part} partitions of size $N_{PIR} = |DB|/N_{Part}$, $|X|$ client inputs, record length ℓ, and a constant factor γ is as follows:

- offline communication: $N_{Part} \cdot \lambda(\ell\sqrt{N_{PIR}} + 1)$ bits.
- online communication:

$$N_{Part} \cdot \underbrace{\left(\frac{2|X|}{N_{Part}} + \gamma \sqrt{\frac{2|X|}{N_{Part}} \cdot \log_2 N_{Part}} \right)}_{\text{num. queries to each partition [64, 61, 63]}} \cdot \underbrace{(2(\lambda+1)\log_2 N_{PIR} + 4\ell)}_{\text{bits per PIR query [47]}} \text{ bits.}$$

Based on this, we can compare the asymptotic communication complexities of our full protocol with the state-of-the-art protocol in [44] in Tab. 1. Our protocol achieves sublinear communication cost in the setup phase, improving significantly over the linear costs in [44]. The online phase of our protocol includes the communication cost of [44] in addition to the PIR protocol being executed for $|X|$

Table 1: Comparison of asymptotic communication complexities considering database size $|DB|$, client set size $|X|$ with at most $|X|^{pre}$ elements, and the number of database partitions $N_{Part} = N_{CF}/N_{PIR}$ for partition size N_{PIR}.

Phase	[44]	Ours
Base	$O(\|X\|^{pre})$	$O(\|X\|^{pre})$
Setup	$O(\|DB\|)$	$O(N_{Part}\sqrt{N_{PIR}})$
Online	$O(\|X\|)$	$O((\|X\| + \sqrt{\|X\|N_{Part}\log N_{Part}})\log N_{PIR})$

client items on N_{Part} partitions. The amortized total communication cost per client item is still significantly smaller in our protocol compared to [44] (cf. § 5.3).

Security. We now discuss the security of our protocol provided by the underlying OPRF and PIR building blocks. We first discuss malicious client behavior and then assumptions required for the server side.

The OPRF protocols used in this work, NR-ECC-OPRF [57, 27, 34, 44] and GC-LowMC-OPRF [62, 20, 1, 44], guarantee malicious client security when using maliciously secure oblivious transfer (OT) [60] and OT extension [5, 45] protocols. PIR protocols generally assume a public database with possible leakage to the client, hence there are no concerns regarding privacy leakage caused by malicious behavior of clients. Furthermore, our protocol's underlying structure prevents clients (and additional servers) from obtaining cleartext database records as they only ever receive encrypted and hashed values as part of the CF. However, in [52], the authors describe an attack on updated databases that enables a malicious client to obtain deleted database records. Therefore, no formal malicious client security is possible for our protocol with updates via in-place edits (cf. § 4.3). We note that clients can generally monitor the database to learn about added and deleted items, so we consider the attack by [52] as irrelevant in our setting and leave the task of formally establishing malicious client security for OO-PIR without updates as future work. Malicious clients can also easily test if the database includes a certain number by running the PSI protocol. Due to the limited entropy of phone numbers, rate limiting of client queries is recommended to restrict the possibility of misuse via large-scale crawling attacks [44, 32, 33].

A malicious server could sabotage the OPRF and PIR sub-protocols by sending incorrect information or by using another input set. As only the client obtains the intersection, this only affects correctness. However, the authors of [44] observe that messengers will afterwards most likely receive the outcome of the intersection and could thus learn about non-registered users in the client's contact list in case they include additional entries in their database. Therefore, service providers must be semi-honest, which is reasonable to assume as they are bound by legal requirements and would face significant financial and reputational risk when detected cheating. As we operate in a multi-server PIR setting, we furthermore have to assume two non-colluding servers. This is a prominent assumption in multi-server protocols for reducing computation and communication costs. We see several successful real-world deployments of protocols utilizing this assumption, e.g., the Internet Security Research Group (ISRG) is providing a non-colluding server for data aggregation and analysis with their "Divvi Up" system [41] based

on "Prio" [16] and "Poplar" [8]. The ISRG further runs non-colluding servers for privacy-preserving COVID-19 analysis in North America [4, 40]. The use of financial incentives [30] and the execution of secure cryptographic protocols inside of trusted execution environments (TEEs) that provide remote attestation (e.g., Intel SGX) could further strengthen the non-collusion assumption between servers.

4.3 Updates

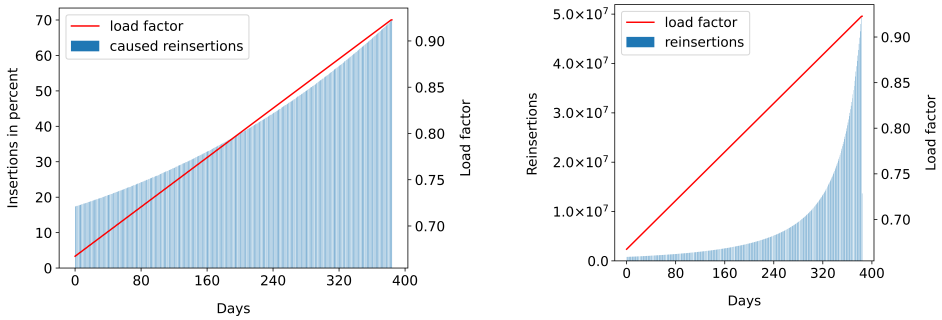
To design our protocol for real-world messaging applications, considering the ever-changing user base and client contacts is essential. The authors of [32, 33] based on publicly available data assume daily change rates of $CR \approx 0.1\,\%$ for Signal, $0.5\,\%$ for Telegram, and only $0.05\,\%$ for WhatsApp. We therefore assume a slowly growing messenger user base with daily updates of at most $1\,\%$, which is already very high given the real-world data of messengers [32, 33].

Updates to the client's phone contacts can include adding or deleting a phone number, and updating a contact's details. Since the client's input is only relevant in the online phase, the handling of updates is trivial: The client can simply run the online phase of the contact discovery protocol for newly added or updated phone numbers to obtain the information if these numbers are registered with a service. Deleted phone numbers are no longer included in the client's set.

Database updates in our protocol could be handled by rerunning the PIR setup and online phase. While this strategy would be simple, the costs would significantly increase with realistic database growth rates, thus making this approach impractical. We therefore propose and evaluate different update strategies for offline-online PIR [47, 52] and PSI for mobile contact discovery [23, 32].

Waterfall Updates. The authors of our selected PIR protocol [47] propose waterfall updates, an update strategy with tiered sub-databases (called buckets) of increasing size. The database is initially stored in one bucket for which the client obtains hints. Updates are inserted into the smallest bucket until this bucket reaches its maximum capacity and overflows into the next larger bucket. The client obtains new hints for all buckets that changed. With this strategy, hints for smaller buckets must be computed and communicated frequently, while larger buckets change less often. With frequent updates, the performance decreases and the client-dependent computational and communication costs increase significantly, which makes this strategy impractical for large-scale messengers and is thus excluded from further evaluations.

Updates via In-place Edits. The authors of [52] propose a different update strategy for OO-PIR [47, 69] that avoids additional databases by updating the client hints to include the updated records. Within our protocol, PIR takes the static-sized CF as a database such that each bucket is a database record in PIR. Updates to the CF do not increase the number of buckets N_{CF}, only their contents and the CF's load factor, which indicates the occupancy level of the filter. Thanks to our protocol's underlying data structure, CFs, we can simplify the approach in [52] by only considering bucket changes, i.e., *in-place edits*. With this strategy, the server applies updates to the CF and sends the corresponding bucket index idx and content change Δ to the client. The client updates all set



(a) Daily percentage of CF insertions that caused reinsertions displayed in blue.

(b) Daily number of CF reinsertions for the given updates.

Fig. 2: Simulation of updates to a CF for database size $|DB| = 2^{31}$, $N_{CF} = 2^{30}$ CF buckets, and a change rate of $CR = 0.1\%/\text{day}$.

parities that contain *idx* by adding the received change, i.e., $p \leftarrow p \oplus \Delta$. While this approach seems straightforward, there is one caveat with the use of CFs: an insertion to the CF can cause a chain of reinsertions where every affected bucket changes and is thus another in-place edit, which potentially increases this strategy's communication cost a lot.

To better understand the impact of reinsertions to the CF, we simulate the growing user base of messengers by inserting a certain percentage CR of the initial database size $|DB|$ to the CF over multiple days. Our simulation shows that more than 80 % of CF insertions are immediately successful during the first days. This number decreases with an increasing load factor α, and at $\alpha \approx 0.92$ insertions start to fail – independent of the change rate CR. Thus, in the following, we focus on the finer-grained change rate of 0.1 % for a detailed analysis of how many positions in the CF must be changed over time.

In Fig. 2a, we give the daily percentage of insertions that initially failed and thus caused reinsertions. We see polynomial growth in the number of reinsertions with increasing load factor in Fig. 2b. With a decreasing number of empty slots, more reinsertions are necessary to insert an item, decreasing the filter's performance. Our simulation shows that most insertions require only few reinsertions to be successful, even when the CF is almost full, however, the number of long reinsertion chains is significantly increasing. We calculate the communication cost of updates based on our simulation (cf. Tab. 2). Transmitting the in-place edits of a single bucket requires *bucket size* + *index length* bit, here 128 bit, with an average of 13.52 MiB/day for $|DB| = 2^{31}$, $CR = 0.1$, and 30 days.

Updates via in-place edits allow the client to update their already stored hints and to run the PIR protocol on the original CF. The daily download cost of CF updates is thus the only additional cost to our PSI protocol. The server-side computation of CF updates is client-independent and requires only *XOR* operations. In comparison, the client-side hint updates require higher computational costs as all set keys must be evaluated to identify sets with updated indices. Thus,

Table 2: Comparison of average update communication costs per day considering $|DB| = 2^{31}$, $|X| = 2^{10}$. Best results marked in bold.

Update Strategies	∅ Comm. / day [MiB]					
	0.1 %/day		0.5 %/day		1.0 %/day	
	1 day	30 days	1 day	30 days	1 day	30 days
In-place edits (Ours, § 4.3) [52]	12.24	13.52	62.05	107.77	126.23	486.37
Additional database (Ours, § 4.3) [23, 32]	**2.90**	**2.90**	**5.63**	**5.63**	**7.65**	**7.65**
Additional database (PK-based PSI) [53, 19]	65.60	65.54	327.74	327.68	655.42	655.36

the update procedure can either be applied at once or during the regular online phase (which requires more client storage).

Additional Update Database. Next to updates to the PIR database, we evaluate the strategy of incremental contact discovery [23, 32], where updates are stored in an additional smaller database on which another PSI instance is run. The client then has to query each of their contacts on the original and the update database. With our PSI protocol, communication and client storage of updates require less than 3 MiB per day for $|X| = 2^{10}$ contacts with a change rate of $CR = 0.1\,\%$/day for $|DB| = 2^{31}$.

We also evaluate the cost of running a simple public key (PK)-based PSI protocol [53, 19] due to its trivial implementation and reasonable communication cost as well as computational efficiency for smaller set sizes [35]. However, this turns out to be significantly less efficient for the considered growth rates (cf. Tab. 2). With less updates and smaller set sizes, PK-based PSI and the state-of-the-art balanced PSI protocols could be more efficient though.

Comparison & Privacy Considerations. We compare the proposed update methods in Tab. 2. Clearly, combining our PIR-based PSI protocol with the incremental contact discovery strategy of [23, 32] is the most efficient solution. We note that updates via in-place edits leak some information to the client about the server's change rate. Likewise, the size of additional update databases clearly indicates this value. Also, when the client repeats the online phase of the protocol for new contacts, this leaks information to the server about the number of changes experienced by the client. Such information leakage can be prevented using dummy insertions and dummy queries.

5 Evaluation

We implemented our protocol in C++ and Go (based on the implementations of [44] and [47]) and describe our evaluation for large-scale set parameters next. Our implementation supports multi-threading on partition level for clients and servers, and introduces optimizations that reduce the client setup time by factor 2.8× over [47] (cf. § 5.2). As described in § 4.1, database partitioning is implemented to circumvent hardware and computational limitations of the underlying PIR protocol for large database sizes. For enhanced reproducability, the server-side implementation is containerized. Our implementation called "DISCO" (short for "DIScover COntacts") is available at https://encrypto.de/code/disco.

5.1 Experimental Setup

To meet the requirements of large-scale messengers, we evaluate server database sizes $|DB| \in \{2^{28}, 2^{31}\}$ and client contact list sizes $|X| \in \{1, 2^{10}\}$. The client is a OnePlus 8T smartphone with Snapdragon 865 octa-core CPU (1x2.84 GHz Cortex-A77, 3x2.42 GHz Cortex-A77, 4x1.80 GHz Cortex-A55) and 12 GiB RAM. Our protocol requires two servers that we set up as Linux VMs on a KVM host with two Intel Xeon Gold 6144 CPUs @ 3.50 GHz. Each VM has 8 logical cores (mapped to 4 physical ones) and 128 GiB RAM. In the multi-threaded benchmarks, denoted with T=4/8, the server uses 4 and the mobile client 8 threads. The number of threads is based on the number of available physical cores as we did not see sufficient performance increase on the server side when putting all logical cores under maximum load.

We consider two network settings: *WiFi* with 566 MBit/s down-/upload speed and 12.4 ms RTT, and *LTE* with 30 MBit/s down-/upload speed and 49.3 ms RTT. The settings are simulated in a real WiFi network by limiting bandwidth and introducing delay using tcconfig [39]. We evaluate the performance of our PSI protocol for the *NR-ECC-* and *GC-LowMC-*OPRF. The OPRF performance was measured on a single thread, the PIR costs on a single and multiple threads.

We benchmarked the impact of different partition sizes and select the best-performing size for each database size considering the trade-off between offline communication and online time.

5.2 Profiling and Optimizations

Via profiling we observed that a bottleneck in the online phase is the client's search for a hint/set that contains the desired index, which requires them to expand each set key until the index is found. The implementation of [47] therefore adds a precomputation step that accelerates this search significantly by generating a mapping between database indices and sets. We optimize the runtime of this client setup by covering not all but only a certain percentage of indices. This significantly reduces offline costs while the online computational costs increase only marginally in the rare case that an index is not found in the mapping table. Considering this trade-off and the requirement of a fast online phase, we use a threshold of 99.99 % for the client preprocessing, reducing the one-time client setup time by 2.8× compared to [47] (286.78 s for $|DB| = 2^{31}$).

Another bottleneck in the protocol is the required one-time computation in the setup phase, including the server's CF creation and client-dependent preprocessing. With parallelization, we reduce the client-dependent setup costs significantly by up to 3.8× with T=4/8 over our protocol's single-threaded setting.

Overall, we achieve a PIR online runtime of less than 1 s for $|X| \leq 2^{10}$ client contacts in the WiFi setting and an improvement of up to factor 8.3× with multiple threads compared to the original single-threaded implementation.

5.3 Comparison to Related Work

We compare our protocol with the state-of-the-art mobile private contact discovery protocol in [44] and PIR-PSI [23] (cf. § 3).

Table 3: Comparison of runtime and communication costs. Runtimes for [44] based on our Go implementation's CF setup and OPRF results. We set $|X|^{pre} = |X|$. Best results marked in bold.

Parameters				Base			Setup					Online		
		Protocols		Time [s]		Comm.		Time			Comm.	Time [s]		Comm.
$\|DB\|$	$\|X\|$	PRF	PSI Parameters	WiFi	LTE	[MiB]	Server [min]	Server [s] (Per-Client)	Client [s] WiFi	LTE	[MiB]	WiFi	LTE	[kiB]
2^{28}	1	NR-ECC	[44] $T = 1$	**0.07**	**0.29**	**0.04**	590.46	**-**	**15.17**	285.95	1 072.14	0.06	0.12	4.05
			Ours $N_{Part} = 32$, $T = 1$	**0.07**	**0.29**	**0.04**	590.46	216.47	109.63	129.57	**66.00**	0.83	5.23	38.79
			Ours $N_{Part} = 32$, T=4/8	**0.07**	**0.29**	**0.04**	590.46	63.71	35.26	**57.76**	**66.00**	0.39	0.76	38.79
		GC-LowMC	[44] $T = 1$	0.09	0.36	0.06	**33.26**	**-**	**15.17**	285.95	1 072.14	**0.04**	**0.07**	**2.02**
			Ours $N_{Part} = 32$, $T = 1$	0.09	0.36	0.06	**33.26**	216.47	109.63	129.57	**66.00**	0.81	5.18	36.76
			Ours $N_{Part} = 32$, T=4/8	0.09	0.36	0.06	**33.26**	63.71	35.26	**57.76**	**66.00**	0.37	0.71	36.76
	2^{10}	NR-ECC	[44] $T = 1$	**0.15**	**0.52**	**2.04**	590.46	**-**	**15.17**	285.95	1 072.14	2.20	2.29	4 145.00
			Ours $N_{Part} = 32$, $T = 1$	**0.15**	**0.52**	**2.04**	590.46	216.47	109.63	129.57	**66.00**	5.59	12.17	6 097.25
			Ours $N_{Part} = 32$, T=4/8	**0.15**	**0.52**	**2.04**	590.46	63.71	35.26	**57.76**	**66.00**	2.65	3.47	6 097.25
		GC-LowMC	[44] $T = 1$	1.26	5.39	21.56	**33.26**	**-**	**15.17**	285.95	1 072.14	**0.63**	**1.22**	**2 064.00**
			Ours $N_{Part} = 32$, $T = 1$	1.26	5.39	21.56	**33.26**	216.47	109.63	129.57	**66.00**	4.02	11.10	4 016.25
			Ours $N_{Part} = 32$, T=4/8	1.26	5.39	21.56	**33.26**	63.71	35.26	**57.76**	**66.00**	1.08	2.40	4 016.25
2^{31}	1	NR-ECC	[44] $T = 1$	**0.07**	**0.29**	**0.04**	4 752.34	**- 121.23**		2286.98	8 576.00	0.06	0.12	4.05
			Ours $N_{Part} = 64$, $T = 1$	**0.07**	**0.29**	**0.04**	4 752.34	1 988.81	961.48	1 035.53	**264.00**	1.93	10.78	77.54
			Ours $N_{Part} = 64$, T=4/8	**0.07**	**0.29**	**0.04**	4 752.34	525.09	286.78	**392.35**	**264.00**	0.49	1.43	77.54
		GC-LowMC	[44] $T = 1$	0.09	0.36	0.06	**269.82**	**- 121.23**		2286.98	8 576.00	**0.04**	**0.07**	**2.02**
			Ours $N_{Part} = 64$, $T = 1$	0.09	0.36	0.06	**269.82**	1 988.81	961.48	1 035.53	**264.00**	1.91	10.73	75.51
			Ours $N_{Part} = 64$, T=4/8	0.09	0.36	0.06	**269.82**	525.09	286.78	**392.35**	**264.00**	0.47	1.38	75.51
	2^{10}	NR-ECC	[44] $T = 1$	**0.15**	**0.52**	**2.04**	4 752.34	**- 121.23**		2286.98	8 576.00	2.20	2.29	4 145.00
			Ours $N_{Part} = 64$, $T = 1$	**0.15**	**0.52**	**2.04**	4 752.34	1 988.81	961.48	1 035.53	**264.00**	8.31	21.50	6 801.49
			Ours $N_{Part} = 64$, T=4/8	**0.15**	**0.52**	**2.04**	4 752.34	525.09	286.78	**392.35**	**264.00**	2.94	4.68	6 801.49
		GC-LowMC	[44] $T = 1$	1.26	5.39	21.56	**269.82**	**- 121.23**		2286.98	8 576.00	**0.63**	**1.22**	**2 064.00**
			Ours $N_{Part} = 64$, $T = 1$	1.26	5.39	21.56	**269.82**	1 988.81	961.48	1 035.53	**264.00**	6.75	20.43	4 720.49
			Ours $N_{Part} = 64$, T=4/8	1.26	5.39	21.56	**269.82**	525.09	286.78	**392.35**	**264.00**	1.37	3.61	4 720.49

Mobile Private Contact Discovery [44] (Tab. 3). Our protocol replaces the costly CF download in [44] – including its communication cost linear in the database size – with a more communication-efficient PIR protocol. We give the benchmark results of these protocols for NR-ECC-OPRF and GC-LowMC-OPRF in Tab. 3. Since both protocols have the same OPRF and CF setup costs, we report these based on our Go implementation and calculate the CF transmission time in the setup phase of [44] based on our connection speeds. With our PIR-based protocol, we achieve total communication costs of 272.68 MiB for $|DB| = 2^{31}$, $N_{Part} = 64$, and $|X| = 2^{10}$ (cf. Tab. 3). This is an improvement by factor 32× compared to ≈8 GiB in [44] at only marginally higher runtimes.

PIR-PSI [23] (Tab. 4). We further compare our protocol implementation with PIR-PSI based on the results in [23, Tab. 2]. The authors of [23] evaluate their performance on a single server with two 18-core Intel Xeon E5-2699 CPUs at 2.30 GHz, 156 GiB RAM, and simulated LAN setting with 10 GB/s bandwidth and 0.02 ms RTT. In comparison, our results are obtained in our WiFi setting using a mobile client and two virtual servers with fewer cores, i.e., 8 vs 18 per machine, and less RAM. Our comparison in Tab. 4 excludes the OPRF costs of our protocol as these would also have to be applied to [23] to strengthen their protocol's non-collusion assumption. As server setup costs are not reported in [23], we exclude them from this comparison.

We compare the runtimes for both protocols using a single (T=1) and multiple threads (T=4/8) in Tab. 4. While PIR-PSI does not require an offline phase,

Table 4: Comparison to PIR-PSI [23]. Results for PIR-PSI are from [23, Tab. 2] with parameters block size b and $\beta = c \cdot |DB|/\log_2(|DB|)$ bins, where c is a scaling factor. The protocols are compared in a single- (T=1) and multi-threading (T=4/8) setting. Best results in the online phase are marked in bold.

Parameters				Offline			Online		
$\|DB\|$	$\|X\|$	Protocols		Time [s]		Comm.	Time [s]		Comm.
		PSI	Param.	T=1	T=4/8	[MiB]	T=1	T=4/8	[kiB]
2^{28}	1	[23]	$c = 1, b = 32$	-	-	-	1.21	-	**30.72**
		Ours	$N_{Part} = 32$	326.10	98.97	66.00	**0.77**	**0.33**	34.74
	2^{10}	[23]	$c = 0.25, b = 1$	-	-	-	33.02	13.22	5 048.32
		[23]	$c = 4, b = 16$	-	-	-	4.07	1.60	28 979.20
		Ours	$N_{Part} = 32$	326.10	98.97	66.00	**3.39**	**0.45**	**1 952.25**

marked with "–", our protocol has client-dependent one-time costs, which can be amortized over all online queries. The DPF-PIR protocol [10, 11] used in PIR-PSI requires online computation linear in the database size, whereas OO-PIR [47] in our protocol has sublinear complexity. For $|DB| = 2^{28}$, our implementation's online runtime is significantly faster for single- and multi-threading, especially considering the hardware and network limitations in our setting. We expect the benefit of our protocol's low online costs to become even more visible for larger database sizes ($|DB| = 2^{31}$), for which PIR-PSI does not report results.

FHE-based PSI [15]. The authors of [15] consider their protocol for the use case of mobile private contact discovery and acknowledge increasing hardware requirements for large-scale database sizes. Based on their recommendation to partition the database, as done in our work, their protocol has 76.2 MiB online communication for $|DB| = 2^{31}$, $|X| = 2^{10}$. Our protocol requires 16.5× less online communication – only 4.61 MiB per online phase – but has additional one-time offline costs, which amortize over many queries. Based on [15, Tab. 2], the runtimes for a single partition of size 2^{28} with T=24 threads are 2 487 s offline and 4.54 s online, which is significantly higher than those of our work. These additional costs, and the lack of a mobile implementation, currently hinder the use of FHE-based protocols for mobile private contact discovery.

6 Conclusion

In this work, we proposed a new communication-efficient unbalanced PSI protocol by combining and further optimizing OPRF-based unbalanced PSI [44] with two-server PIR [47]. With this, we take big steps towards practicality of large-scale mobile private contact discovery. While our protocol achieves a significant reduction in communication and thus outperforms the state-of-the-art protocol mobile private contact discovery [44] in this regard, the client-dependent setup and update costs are still limiting factors for real-world practicality with large-scale messengers. Continuing research on PIR protocols with client-independent preprocessing is thus a crucial area of future work.

Acknowledgements

This project received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 850990 PSOTI). It was co-funded by the Deutsche Forschungsgemeinschaft (DFG) within SFB 1119 CROSSING/236615297 and GRK 2050 Privacy & Trust/251805230.

References

1. Albrecht, M.R., Rechberger, C., Schneider, T., Tiessen, T., Zohner, M.: Ciphers for MPC and FHE. In: EUROCRYPT (2015)
2. Ali, A., Lepoint, T., Patel, S., Raykova, M., Schoppmann, P., Seth, K., Yeo, K.: Communication-computation trade-offs in PIR. In: USENIX Security (2021)
3. Angel, S., Chen, H., Laine, K., Setty, S.T.V.: PIR with compressed queries and amortized query processing. In: S&P (2018)
4. Apple, Google: Exposure Notification Privacy-preserving Analytics (ENPA) White Paper. https://covid19-static.cdn-apple.com/applications/covid19/current/static/contact-tracing/pdf/ENPA_White_Paper.pdf (2021)
5. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer extensions with security for malicious adversaries. In: EUROCRYPT (2015)
6. Beimel, A., Ishai, Y., Malkin, T.: Reducing the servers computation in private information retrieval: PIR with preprocessing. In: CRYPTO (2000)
7. Bloom, B.H.: Space/Time trade-offs in hash coding with allowable errors. Commun. ACM **13**(7) (1970)
8. Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., Ishai, Y.: Lightweight techniques for private heavy hitters. In: S&P (2021)
9. Borrello, P., Kogler, A., Schwarzl, M., Lipp, M., Gruss, D., Schwarz, M.: ÆPIC leak: Architecturally leaking uninitialized data from the microarchitecture. In: USENIX Security (2022)
10. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: EUROCRYPT (2015)
11. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing: Improvements and extensions. In: CCS (2016)
12. Bui, D., Couteau, G.: Improved private set intersection for sets with small entries. In: PKC (2023)
13. Chen, H., Huang, Z., Laine, K., Rindal, P.: Labeled PSI from fully homomorphic encryption with malicious security. In: CCS (2018)
14. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: CCS (2017)
15. Cong, K., Moreno, R.C., da Gama, M.B., Dai, W., Iliashenko, I., Laine, K., Rosenberg, M.: Labeled PSI from homomorphic encryption with reduced computation and communication. In: CCS (2021)
16. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: NSDI (2017)
17. Corrigan-Gibbs, H., Henzinger, A., Kogan, D.: Single-server private information retrieval with sublinear amortized time. In: EUROCRYPT (2022)
18. Corrigan-Gibbs, H., Kogan, D.: Private information retrieval with sublinear online time. In: EUROCRYPT (2020)
19. Cristofaro, E.D., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: CANS (2012)
20. Cristofaro, E.D., Tsudik, G.: Practical private set intersection protocols with linear complexity. In: FC (2010)

21. Cui, J., Yu, J.Z., Shinde, S., Saxena, P., Cai, Z.: SmashEx: Smashing SGX enclaves using exceptions. In: CCS (2021)
22. Davidson, A., Pestana, G., Celi, S.: FrodoPIR: Simple, scalable, single-server private information retrieval. PETS (2023)
23. Demmler, D., Rindal, P., Rosulek, M., Trieu, N.: PIR-PSI: Scaling private contact discovery. PETS (2018)
24. Eppstein, D.: Cuckoo filter: Simplification and analysis. In: SWAT (2016)
25. Facebook, Inc. (FB): First Quarter 2020 Results Conference Call. https://s21.q4cdn.com/399680738/files/doc_financials/2020/q1/Q1'20-FB-Earnings-Call-Transcript.pdf (2020)
26. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.: Cuckoo filter: Practically better than bloom. In: CoNEXT (2014)
27. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: TCC (2005)
28. Garimella, G., Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Oblivious key-value stores and amplification for private set intersection. In: CRYPTO (2021)
29. Ghosh, S.: Facebook probably has your phone number, even if you never shared it. Now it has a secret tool to let you delete it. https://www.businessinsider.com/facebook-has-hidden-tool-to-delete-your-phone-number-email-2022-10 (2022)
30. Gong, T., Henry, R., Psomas, A., Kate, A.: More is merrier in collusion mitigation. CoRR arXiv:2305.08846 (2022)
31. Günther, D., Heymann, M., Pinkas, B., Schneider, T.: GPU-accelerated PIR with client-independent preprocessing for large-scale applications. In: USENIX Security (2022)
32. Hagen, C., Weinert, C., Sendner, C., Dmitrienko, A., Schneider, T.: All the numbers are US: Large-scale abuse of contact discovery in mobile messengers. In: NDSS (2021)
33. Hagen, C., Weinert, C., Sendner, C., Dmitrienko, A., Schneider, T.: Contact discovery in mobile messengers: Low-cost attacks, quantitative analyses, and efficient mitigations. TOPS (2023)
34. Hazay, C., Lindell, Y.: Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. Journal of Cryptology (2010)
35. Heinrich, A., Hollick, M., Schneider, T., Stute, M., Weinert, C.: PrivateDrop: Practical privacy-preserving authentication for Apple AirDrop. In: USENIX Security (2021)
36. Henry, R.: Polynomial batch codes for efficient IT-PIR. PETS (2016)
37. Henzinger, A., Hong, M.M., Corrigan-Gibbs, H., Meiklejohn, S., Vaikuntanathan, V.: One server for the price of two: Simple and fast single-server private information retrieval. In: USENIX Security (2023)
38. Hetz, L., Schneider, T., Weinert, C.: Scaling mobile private contact discovery to billions of users. In: ESORICS (2023)
39. Hombashi, T.: Tcconfig. https://github.com/thombashi/tcconfig (2022)
40. Internet Security Research Group: ISRG Prio Services for Preserving Privacy in COVID-19 EN Apps. https://divviup.org/blog/prio-services-for-covid-en/ (2021)
41. Internet Security Research Group: Divvi Up. https://divviup.org/ (2023)
42. Ion, M., Kreuter, B., Nergiz, A.E., Patel, S., Saxena, S., Seth, K., Raykova, M., Shanahan, D., Yung, M.: On deploying secure computing: Private intersection-sum-with-cardinality. In: EuroS&P (2020)
43. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Sahai, A.: Batch codes and their applications. In: STOC (2004)

44. Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: USENIX Security (2019)
45. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: CRYPTO (2015)
46. Kiss, Á., Liu, J., Schneider, T., Asokan, N., Pinkas, B.: Private set intersection for unequal set sizes with mobile applications. PETS (2017)
47. Kogan, D., Corrigan-Gibbs, H.: Private blocklist lookups with checklist. In: USENIX Security (2021)
48. Kolesnikov, V., Kumaresan, R., Rosulek, M., Trieu, N.: Efficient batched oblivious PRF with applications to private set intersection. In: CCS (2016)
49. Lazzaretti, A., Papamanthou, C.: Single server PIR with sublinear amortized time and polylogarithmic bandwidth. ePrint 2022/081 (2022)
50. Li, L., Pal, B., Ali, J., Sullivan, N., Chatterjee, R., Ristenpart, T.: Protocols for checking compromised credentials. In: SIGSAC (2019)
51. Liu, J., Li, J., Wu, D., Ren, K.: PIRANA: Faster multi-query PIR via constant-weight codes. ePrint 2022/1401 (2022)
52. Ma, Y., Zhong, K., Rabin, T., Angel, S.: Incremental Offline/Online PIR. In: USENIX Security (2022)
53. Meadows, C.A.: A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In: S&P (1986)
54. Menon, S.J., Wu, D.J.: SPIRAL: Fast, high-rate single-server PIR via FHE composition. In: S&P (2022)
55. Mughees, M.H., Chen, H., Ren, L.: OnionPIR: Response efficient single-server PIR. In: CCS (2021)
56. Mughees, M.H., Ren, L.: Vectorized batch private information retrieval. S&P (2023)
57. Naor, M., Reingold, O.: Number-theoretic constructions of efficient pseudo-random functions. Journal of ACM **51**(2) (2004)
58. Nevo, O., Trieu, N., Yanai, A.: Simple, fast malicious multiparty private set intersection. In: CCS (2021)
59. Olson, P.: Facebook Closes $19 Billion WhatsApp Deal. https://www.forbes.com/sites/parmyolson/2014/10/06/facebook-closes-19-billion-whatsapp-deal/ (2014)
60. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: CRYPTO (2008)
61. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. In: USENIX Security (2015)
62. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: AC (2009)
63. Pinkas, B., Schneider, T., Zohner, M.: Scalable private set intersection based on OT extension. TOPS (2018)
64. Raab, M., Steger, A.: "Balls into Bins" - A simple and tight analysis. In: RANDOM (1998)
65. Ragab, H., Milburn, A., Razavi, K., Bos, H., Giuffrida, C.: CrossTalk: Speculative data leaks across cores are real. In: S&P (2021)
66. Raghuraman, S., Rindal, P.: Blazing fast PSI from improved OKVS and subfield VOLE. In: CCS (2022)
67. Resende, A.C.D., Aranha, D.F.: Faster unbalanced private set intersection. In: FC (2018)
68. Rindal, P., Schoppmann, P.: VOLE-PSI: Fast OPRF and circuit-PSI from vector-OLE. In: EUROCRYPT (2021)

69. Shi, E., Aqeel, W., Chandrasekaran, B., Maggs, B.M.: Puncturable pseudorandom sets and private information retrieval with near-optimal online bandwidth and time. In: CRYPTO (2021)
70. Thomas, K., Pullman, J., Yeo, K., Raghunathan, A., Kelley, P.G., Invernizzi, L., Benko, B., Pietraszek, T., Patel, S., Boneh, D., Bursztein, E.: Protecting accounts from credential stuffing with password breach alerting. In: USENIX Security (2019)
71. Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D.: Epione: Lightweight contact tracing with strong privacy. IEEE Data Eng. Bull. **43**(2) (2020)
72. Troy Hunt: Have I Been Pwned: Check if your email has been compromised in a data breach. https://haveibeenpwned.com/ (2023)
73. van Schaik, S., Minkin, M., Kwong, A., Genkin, D., Yarom, Y.: CacheOut: Leaking data on intel CPUs via cache evictions. In: S&P (2021)
74. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: FOCS (1986)
75. Yeo, K.: Lower bounds for (batch) PIR with private preprocessing. In: EUROCRYPT (2023)
76. Zhou, M., Lin, W.K., Tselekounis, Y., Shi, E.: Optimal single-server private information retrieval. In: EUROCRYPT (2023)

Appendix

A PIR Survey

In Tab. 5, we summarize our survey of recent PIR protocols for their use in OPRF-based PSI based on which we selected the OO-PIR by Kogan and Corrigan-Gibbs [47].

Table 5: Surveyed PIR protocols for OPRF-based PSI. Complexities are simplified. We distinguish between client C's and server(s) S's computational costs where possible. Table entries are left empty when complexities are not clear from the original paper or related work.

n	Protocol	Assumption	Preprocessing	Updatablity	Batching	Implementation	Offline Comp. C	Offline Comp. S	Offline Comm.	Online Comp. C	Online Comp. S	Online Comm.
1	SealPIR [3]	RLWE	✓	✗	✓	✓	-	N	-	N		$dN^{1/d}$
	MulPIR [2]	RLWE	✓	✗	✓	✓	-	N	-			$dN^{1/d}$
	[56]	RLWE	✓‡	✗	✓	✓	-	N	-	$B/pN_B^{2/d}$		$BN_B^{1/d}/p$
	Spiral (family) [54]	RLWE	✓	✗	✓	✓	-	N				$\log N$
	PIRANA [51]	RLWE	✓	✗	✓	✓	-	N		N/M	N/M	N/M
	[18]	LWE	✓†	✗	✗	✗	$\sqrt{N}$	N	$\sqrt{N}$	$\sqrt{N}$		$\sqrt{N}$
	OnionPIR [55]	RLWE	✓†	✗	✓	✓	N	N	N	N		N
	[49]	LWE	✓†	✗	✗	✗	N	N	N	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$
	[76]	LWE	✓†	✗	✓	✗		N	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	1
	[17]	RLWE	✓†	✗	✓	✗	N	N	N	N	N	N
	SimplePIR [37]	LWE	✓†‡	✓	✓	✓	N/M	N	$\sqrt{N}$		N	$\sqrt{N}$
	DoublePIR [37]	LWE	✓†‡	✓	✓	✓		N	d_l^2			$\sqrt{N}$
	FrodoPIR [22]	LWE	✓†‡	✓	✓	✓	N	N	1	N	1	N
2+	DPF-PIR [10]	OWF	✗	-	✓	✓	-	-	-	$\log N$	N	$n \log N$
	CIP-PIR [31]	OWF	✓‡	✓‡	✓	✓	-	N	-	$\sqrt{N/n}$	N/n	$n\sqrt{N/n}$
	[18]	OWF	✓†	✗	✗	✗	$\sqrt{N}$	N	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$n \log N$
	[47]	OWF	✓†	✓¶	✓	✓⋆	$\sqrt{N}$	N	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$n \log N$
	[69]	LWE	✓†	✗	✓	✗	$\sqrt{N}$	N	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$n \log N$
	iCK [52]	OWF	✓†	✓‖	✓	✓	$\sqrt{N}$	N	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$n\sqrt{N}$
	iSACM [52]	LWE	✓†	✓‖	✓	✓	$\sqrt{N}$	N	$\sqrt{N}$	$\sqrt{N}$	$\sqrt{N}$	$n\sqrt{N}$

Database size N, number of servers n, plaintext size p, lattice dimension d_l, database hypercube dimension d, encryption parameter M, number of buckets B and bucket size N_B, † Stateful / offline-online, ‡ client-independent, ¶ waterfall updates, ‖ in-place edits, ⋆ includes mobile implementation.