Mobile Private Contact Discovery at Scale

Daniel Kales
Graz University of Technology

Christian Rechberger
Graz University of Technology

Thomas Schneider
TU Darmstadt

Matthias Senker
TU Darmstadt

Christian Weinert
TU Darmstadt

Abstract

Mobile messengers like WhatsApp perform contact discovery by uploading the user's entire address book to the service provider. This allows the service provider to determine which of the user's contacts are registered to the messaging service. However, such a procedure poses significant privacy risks and legal challenges. As we find, even messengers with privacy in mind currently do not deploy proper mechanisms to perform contact discovery privately.

The most promising approaches addressing this problem revolve around private set intersection (PSI) protocols. Unfortunately, even in a weak security model where clients are assumed to follow the protocol honestly, previous protocols and implementations turned out to be far from practical when used at scale. This is due to their high computation and/or communication complexity as well as lacking optimization for mobile devices. In our work, we remove most obstacles for large-scale global deployment by significantly improving two promising protocols by Kiss et al. (PoPETS'17) while also allowing for malicious clients.

Concretely, we present novel precomputation techniques for correlated oblivious transfers (reducing the online communication by factor 2x), Cuckoo filter compression (with a compression ratio of $\approx 70\,\%$), as well as 4.3x smaller Cuckoo filter updates. In a protocol performing oblivious PRF evaluations via garbled circuits, we replace AES as the evaluated PRF with a variant of LowMC (Albrecht et al., EUROCRYPT'15) for which we determine optimal parameters, thereby reducing the communication by factor 8.2x. Furthermore, we implement both protocols with security against malicious clients in C/C++ and utilize the ARM Cryptography Extensions available in most recent smartphones. Compared to previous smartphone implementations, this yields a performance improvement of factor 1,000x for circuit evaluations. The online phase of our fastest protocol takes only 2.92 s measured on a real WiFi connection (6.53 s on LTE) to check 1,024 client contacts against a large-scale database with 2^{28} entries. As a proof-of-concept, we integrate our protocols in the client application of the open-source messenger Signal.

1 Introduction

After installation, mobile messaging applications first perform a so-called *contact discovery*. This allows new users to automatically connect with all other users of the messaging service whose phone numbers are stored in their address book. There exist various ways to perform contact discovery. For example, WhatsApp simply uploads the user's entire address book on a regular basis to match contacts [1].

However, revealing all personal contacts to a service provider poses significant privacy risks: from the social graph of users a variety of personal information can be inferred and journalists, for example, may need to cover the identity of some of their informants to protect whistleblowers from potential consequences. When installing a mobile messaging application, users also jeopardize the privacy of people who are not even connected to the particular service by transmitting their contact information without consent. An illustrative example of a severe breach of privacy can be seen in the case of WhatsApp, which was acquired by Facebook in 2014 and shared its database with the parent company: Facebook users received friend recommendations of strangers who happened to see the same psychiatrists [33].

Unfortunately, applying simple protection mechanisms like hashing the phone numbers of contacts locally before the upload to the service provider is not helpful since these hashes are vulnerable to brute-force and dictionary attacks due to the relatively small range of possible pre-images. Furthermore, the service provider can still tell whether two users share a contact even a long time after running the discovery routine by storing the received hash values. Custom wrappers[1] for messaging applications can somewhat circumvent these problems by allowing users to manually select contacts to expose to the messaging application. However, this approach only protects the contacts of users actually using such custom wrappers. Furthermore, manually selecting the contacts to match is a usability problem.

[1] e.g., https://www.backes-srt.com/en/solutions-2/whatsbox

One possible solution to this dilemma is to apply a particular form of secure two-party computation. In general, secure two-party computation allows parties P_1 and P_2 to jointly compute a publicly known function f on their respective inputs X_1 and X_2 s.t. the parties learn no information from the protocol execution but the result. The research area of *private set intersection* (PSI) focuses on optimized protocols for the case where X_1 and X_2 are sets of elements, and f is the intersection function. PSI has been studied in great depth in the past years, yielding very efficient protocols (e.g., [41, 51]) based on oblivious transfer extensions (OTe, cf. [4, 36, 39]). However, while these protocols are very efficient in many scenarios, they turn out to be impractical for use-cases like private contact discovery on mobile devices, where the input set of the service provider is much larger (sometimes by a factor of a few million) than the input set of the user. This is because the online phase of these protocols (which depends on the actual inputs) has a computation and communication complexity that is linear in the size of the larger set.

Therefore, other PSI protocols for the case of *unbalanced* set sizes were developed (e.g., [19, 21, 40, 59]). However, only [40] actually provides an implementation on real mobile smartphone clients. The experiments performed by the authors of [40] show a rather large discrepancy between protocol execution on x86-based PC hardware and Android smartphones where performance-critical cryptographic operations are implemented in Java. In fact, their performance results do not encourage real-world deployment. For example, their fastest protocol that can easily be made secure against malicious clients requires more than 52 s on a smartphone with WiFi connection to check a single client contact against a database with only 2^{20} entries.

The developers of Signal, a mobile messaging service similar to WhatsApp but with focus on privacy, considered the use of PSI protocols for contact discovery. However, they refrained from actually implementing PSI since the academic research in PSI and the related private information retrieval (PIR) protocols "is quite a disappointment" [44]. Instead, they presented a technology preview that protects the contact discovery task on the server side with Intel Software Guard Extensions (SGX), a trusted execution environment that can be attested by remote users [45]. In theory, this yields a secure contact discovery service with negligible performance overhead compared to plain computation. However, Intel SGX is a proprietary engineering-driven solution with no cryptographic security guarantees and vulnerable to severe attacks, e.g., the recent Foreshadow attack [16] managed to reliably extract confidential data from enclaves. Moreover, some fixes for hardware security designs such as Intel SGX require hardware changes that can take years to enter the market and result in repeated acquisition costs. In contrast, fixes for flawed implementations of provably secure cryptographic protocols can be deployed quickly via software updates.

Thus, we revisit state-of-the-art unbalanced PSI protocols which provide cryptographic security and show that using new optimizations and native implementations they turn out to be practical on modern smartphones. Furthermore, we achieve security against malicious clients: since every user could run a manipulated version of the messaging application, deviations from the protocol may lead to revealing information about the server's database. On the other hand, we assume that the server behaves semi-honestly, i.e., it follows the protocol but tries to learn as much information as possible. This is a reasonable assumption since there are legal requirements and financial incentives to behave correctly: once misconduct gets known publicly, users will abandon the misbehaving service and switch to a more trustworthy alternative.

1.1 Our Contributions

As a motivation, we investigate how contact discovery is handled in widely used mobile messaging applications. For this, we conduct a survey where we analyze privacy policies, source code, and network traffic. Our results show that in practice none of these applications protect the users' privacy during contact discovery.

We optimize two protocols for unbalanced PSI that can easily be made secure against malicious clients and are suitable for private contact discovery: one that uses oblivious evaluations of the Naor-Reingold PRF (NR-PSI, cf. [31, 40, 47]) and one that uses Yao's garbled circuits (GC-PSI, cf. [40, 52, 56]) to run oblivious AES evaluations. For both protocols we apply new forms of correlated random OT precomputation (reducing the online communication by factor 2x, which is of independent interest) and introduce a method for Cuckoo filter compression (with a compression ratio of $\approx 70\,\%$ and negligible computational overhead) as well as 4.3x smaller Cuckoo filter updates to reduce the required network communication. Moreover, we improve the GC-PSI protocol by instantiating the PRF with LowMC [2], a cipher specifically designed for efficient evaluation in secure protocols, instead of the default choice AES. While this was already proposed in [40], we find optimal parameter sets for LowMC and provide implementations. Compared to AES, we thereby reduce the communication by factor 8.2x.

We provide C/C++ implementations for both protocols with security against malicious clients that make use of the Cryptography Extensions (CE) in the ARMv8 architecture available in most recent smartphones for hardware-accelerated execution. Thereby, we improve the runtime of the online phase of the GC-PSI protocol by more than a factor of 1,000x compared to the previous work of [40] that only implements security against semi-honest clients. We overcome further shortcomings of previous works w.r.t security and scalability by evaluating the implementations using recommended security parameters, reasonable false positive probabilities, and considering large-scale set sizes on the server side.

Our fastest protocol takes only 2.92 s measured on a real WiFi connection (6.53 s on LTE) and 6.07 MiB of communication in the online phase to check 1,024 client contacts against a database with 2^{28} entries (more than the number of monthly active users for popular messengers like Telegram [61]). For the setup phase it is required to transfer a compressed Cuckoo filter once whose size is linear in the number of the database entries (≈ 1 GiB for 2^{28} entries); since the filter is identical for all clients, service providers can handle the resulting traffic efficiently via CDNs. To remain practical for even larger set sizes (the market leader WhatsApp currently has more than 1.6 billion users [61]), we suggest multiple extensions, e.g., combining our protocols with multi-server PIR s.t. the overall client-server communication complexity becomes logarithmic in the size of the server database.

As a proof-of-concept, we integrate both of our protocols in the Signal Android client, thereby positioning our secure cryptographic approach as a practical alternative to vulnerable trusted execution environments like Intel SGX.

1.2 Motivating Survey

To determine how contact discovery is currently being done in practice, we conducted a survey on a comprehensive selection of mobile messengers that are "secure" in the sense that they offer end-to-end encryption. Each application was analyzed by evaluating the mandatory privacy policy, which is supposed to state exactly which data the application transmits to its server and how the server processes and stores that data. Unfortunately, these policies are not always precise enough to determine the employed contact discovery method. In these cases, we inspected the source code (if publicly available) or the network communication by means of the man-in-the-middle proxy *mitmproxy*[2]. We circumvented certificate pinning by using the *Xposed*[3] framework together with the *JustTrustMe*[4] plugin that can disable certificate checking routines in several commonly used security libraries.

Our results are summarized in Tab. 1. All surveyed messengers upload contact information (at least the contact's phone number) either in the clear or in hashed form. While this form of contact discovery is very efficient (requiring only a few bytes of communication per element), it threatens the privacy of users directly or indirectly via brute-force or dictionary attacks. Furthermore, even if the server cannot determine the actual contact data, it can still tell whether two users share a contact by comparing uploaded hash values.

This can be somewhat mitigated by using salted hashing s.t. the hashes received by the server are different whenever a client triggers contact discovery. However, only one of the surveyed messengers employs this approach as it requires to hash the entire server database for each fresh salt received by a client. Furthermore, brute-force attacks are still feasible.

Messenger	Hashed	Salted	Analysis Technique
Confide*	✓	✗	Privacy policy
Dust*	✗	✗	Network traffic
Eleet*	✗	✗	Privacy policy
G DATA Secure Chat	✓	✗	Network traffic
Signal (legacy)	✓	✗	Source code
SIMSme	✓	✓	Network traffic
Telegram	✗	✗	Privacy policy
Threema	✓	✗	Privacy policy
Viber	✗	✗	Privacy policy
WhatsApp	✗	✗	Privacy policy
Wickr Me	✓	✗	Privacy policy
Wire	✓	✗	Privacy policy

Table 1: Results of our contact discovery survey on secure mobile messengers. All applications upload contact information either in the clear or hashed (with salt). Messengers marked with * denote that contact discovery is optional.

2 Related Work

In this section, we discuss existing unbalanced PSI protocols and other works that focus on PSI in the smartphone setting.

Unbalanced PSI. Kiss et al. [40] discuss multiple unbalanced PSI protocols with precomputation (cf. §3.5) and security against semi-honest adversaries. Their NR-PSI and GC-PSI protocols (based on [31] and [52], respectively) are the foundation of our work. We augment these protocols with new OT precomputation techniques, efficient Cuckoo filters [27, 59], a specialized cipher [2] for the GC-PSI protocol, and security against malicious clients. The authors of [40] also evaluate their protocols on smartphones, but based on less efficient Java implementations. In our work, we present C/C++ implementations that make use of the hardware-accelerated cryptography available in most recent smartphones.

Resende and de Freitas Aranha [59] use techniques similar to [40], but replace Bloom filters [12] with the more efficient and versatile Cuckoo filters [27] to efficiently represent the encrypted server database (cf. §3.4) in a Diffie-Hellman style PSI protocol [7] with security against semi-honest adversaries. In our work, we optimize communication by proposing methods for Cuckoo filter compression and updates, and perform evaluations with reasonable parameters: while in [59] the authors settle with an error probability of $\approx 2^{-13}$, which results, on average, in one false positive when 10 clients match 2^{10} contacts each, we propose realistic Cuckoo filter parameters for error probabilities $\approx 2^{-29}$ and $\approx 2^{-39}$.

Demmler et al. [21] present a different approach assuming multiple non-colluding servers. Their idea is to first perform a variant of private information retrieval (PIR) to reduce the

[2] https://mitmproxy.org
[3] https://repo.xposed.info
[4] https://github.com/Fuzion24/JustTrustMe

server's input set and then perform a traditional PSI protocol on the reduced sets. While this approach is very performant, the requirement of non-colluding servers presents challenges for the data-owners: they not only need to guarantee that these servers do not collude, but also need to ensure that their client data is not leaked to other parties. This leads to the difficult situation where the server party needs to trust a second server but simultaneously is assumed to not collude with it. However, even if servers are malicious and/or collude, they cannot learn more about client inputs than in currently deployed naive hashing-based contact discovery methods.

Chen et al. [19] give a PSI protocol based on fully homomorphic encryption. The authors present multiple optimizations that make the protocol practically viable. Their work was improved and extended to the special use case of *labeled* PSI [18], where for intersecting items an associated label is transferred and security is not only guaranteed in case of malicious clients but also malicious servers (with some controlled leakage). The advantage of the protocols of [18, 19] is that their communication complexity is sublinear instead of linear in the size of the server set. However, this comes at the cost of repeated high computational overhead, whereas the online phase of our protocols is very efficient and requires no cryptographic operations on the server side.

Mobile PSI. Huang et al. [34] provided first performance results for secure computation on smartphones with security against semi-honest adversaries. They implemented a circuit-based PSI protocol on Android. Their implementation managed to evaluate ≈ 100 AND gates per second, taking about 10 min to intersect two sets of 256 items each.

Asokan et al. [6] implemented an RSA-based PSI protocol with security against semi-honest adversaries on smartphones for secure mobile resource sharing.

Carter et al. [17] presented a maliciously secure system for secure outsourced garbled circuit evaluation on mobile devices. Subsequently, Mood et al. [46] showed how to further optimize outsourced evaluation. They also point out how their framework can be used to implement a secure friend finder.

"PROUD" [49] is a decentralized approach for private contact discovery based on the DNS system. It enables users to privately discover the current network addresses of friends, which differs from the scenario of a centralized messaging service we consider. Moreover, friendship bootstrapping requires an out-of-band communication channel between users.

Compared to these works, we optimize protocols for unbalanced PSI with a central service provider and provide native implementations for maximum performance on smartphones.

3 Background

In the following, we introduce cryptographic building blocks that are required for the remainder of this work.

3.1 Oblivious Transfer (Extensions)

Oblivious transfer (OT) [57] is a cryptographic protocol that in its most basic form allows a sender P_1 to obliviously transfer one out of two messages (m_0, m_1) to a receiver P_2 based on a selection bit b chosen by P_2 s.t. P_1 learns nothing about b and P_2 learns only m_b but nothing about m_{1-b}.

It was shown in [35] that performing OTs always requires some form of public key cryptography. However, with OT extension (OTe) protocols [9, 36], a small number (e.g., 128) of "base OTs" can be extended to a large number of OTs using only efficient symmetric cryptographic operations.

There exist flavors of OTe with reduced communication complexity [5]: In random OT (R-OT), neither party inputs any values, but the inputs of sender and receiver are randomly chosen by the protocol. In correlated OT (C-OT), m_0 is chosen at random, whereas m_1 is computed as a function f of m_0: $m_1 = f(m_0)$, where f is privately known to P_1 only.

It is possible to precompute OTs s.t. all computationally expensive operations are performed via R-OTs in advance [8]. Later, the random values obtained via R-OTs are used to mask the actual inputs, requiring only cheap XOR operations in the style of one-time-pad encryption.

3.2 Garbled Circuits

Yao's garbled circuits (GC) [62] is one of the most prominent techniques for secure two-party computation. (In the following the two parties are called *garbler* and *evaluator*.) The idea is to represent the function that is evaluated as a Boolean circuit and to replace each logical two-input gate by a *garbled gate*. Each wire of the garbled gate is given two random wire labels, representing 0 and 1. To garble a gate, the garbler uses all four combinations of the gate's two input wire labels to encrypt the corresponding output wire label, based on the truth table of the original gate, and sends the resulting ciphertexts, the so-called *garbled table*, to the evaluator. The evaluator can then use the two input wire labels it possesses to decrypt one of the four ciphertexts and receive the output wire label, which is then used as input for subsequent gates.

We now describe how the evaluator obtains the wire labels corresponding to the inputs of the two parties: Since the garbler knows all wire labels, it can send the wire labels corresponding to its input bits to the evaluator. However, to ensure input privacy for the evaluator, the wire labels corresponding to the evalutor's input bits are retrieved via OTs. The garbler also sends information that allows the evaluator to decode the final output wire labels to 0 or 1.

Several optimizations for Yao's original scheme have been presented s.t. today it is most efficient to combine the following techniques: Point-and-Permute [10], Free-XOR [42], fixed-key AES garbling [11], and Half-Gates [63].

3.3 OPRF Evaluation

An oblivious pseudorandom function (OPRF) is a protocol between two parties: sender P_1 holding key k and receiver P_2 holding input x. After the invocation of the protocol, P_2 learns the output $f_k(x)$ of a keyed pseudorandom function (PRF) f. Additionally, it is guaranteed that P_1 does not learn anything about x and P_2 does not learn anything about k.

OPRF evaluations can be used to build PSI protocols as proposed in [28, 30, 40, 52]: The server samples a key k uniformly at random, evaluates the PRF $f_k(x_i)$ on each of its items $x_i \in X$, and sends the results to the client. Server and client now engage in the OPRF protocol, where the server inputs key k and the client inputs elements $y_j \in Y$. After this step, the client obtains $f_k(y_j)$ for each item $y_j \in Y$ and can perform a plain intersection between the items $f_k(x_i)$ and $f_k(y_j)$. The client then outputs the elements y_j corresponding to the values in the intersection.

In this work, we instantiate the PRF either using the Naor-Reingold PRF [47] (NR-PSI) or a garbled circuit-based evaluation of a block cipher (GC-PSI). In [37], the authors describe an alternative algebraic OPRF construction based on a PRF by Dodis-Yampolskiy [25]. However, due to the use of Paillier encryption, this construction is likely slower than the Naor-Reingold PRF and their follow-up work [38], the basis for [59] (cf. §6.2). Moreover, it requires a common reference string in the form of an RSA modulus with unknown factorization.

3.4 Cuckoo Filters

Cuckoo filters [27] are an alternative to the more popular Bloom filters [12]. Like Bloom filters, they are a data structure for compact set representation that allows for fast membership testing with controllable *false positive probability* (FPP). Cuckoo filters employ a hashing technique similar to Cuckoo hashing [48], which has been used in the past as a building block in PSI protocols (e.g., [41, 51, 53–56]).

Resende and de Freitas Aranha [59] first used Cuckoo filters in a PSI protocol. This is due to several advantages over Bloom filters when representing the server's database, namely they (i) support inserting and deleting items subsequently, whereas standard Bloom filters only support inserting items, and variants that do support deletion such as counting Bloom filters have much higher storage costs; (ii) have better lookup performance; and (iii) use less space in many scenarios while having the same false positive probability.

Cuckoo filters consist of a table of buckets with fixed bucket size b. Inside the buckets, so-called tags are stored. Tags are small bitstrings obtained by hashing items. More precisely, to represent an item x in a Cuckoo filter, we first calculate its tag $t_x = H_t(x)$, where H_t is a hash function with output bitlength v. This tag is stored in one out of two possible buckets. The position of the first possible bucket is calculated as $p_1 = H(x)$, where H is another hash function that maps the input to a position in the table of buckets. In case this bucket is already full, the tag is stored in the second possible bucket at position $p_2 = p_1 \oplus H(t_x)$. Note that it is always possible to determine the other candidate bucket p_j just from knowing its tag t_x and the current position p_i: $p_j = p_i \oplus H(t_x)$. If both buckets are full, one tag in one of the buckets is chosen at random, removed from that bucket, and moved to its other possible bucket. This procedure is repeated recursively until no more relocations are necessary.

To check whether an item is contained in the Cuckoo filter, one computes its tag and both possible bucket locations and compares the tags stored there for equality. For deleting the item, the matching tag is removed from the filter.

Due to hash collisions, two items may produce equal tags. As a consequence, lookups can lead to false positives. The false positive probability ε_{max} is mainly dependent on the tagsize v and also slightly on the bucket size b since larger buckets result in more possible collisions within each bucket.

3.5 Unbalanced PSI with Precomputation

For private contact discovery, the following properties are desired: (i) the server performs the computationally expensive tasks; (ii) all computationally expensive and communication intensive tasks are performed only once; and (iii) the actual intersection computation is very fast and also allows for efficient updates. Therefore, [40] suggest to use PSI protocols with precomputation, where most time consuming tasks are performed ahead of the actual intersection.

Our PSI protocols for unbalanced set sizes share a common structure. Following the precomputation approach of [40], they are divided into the following four phases: (i) The *base phase* is completely independent of any input data and consists, e.g., of OT precomputation. Its complexity is linear in the maximum number of contacts a client expects to match in future protocol executions before the base phase is re-run. (ii) The complexity of the *setup phase* is linear in the size of the large set held by the server. It involves encrypting all elements in the server database via PRF evaluations as described in §3.3 and inserting them into a Cuckoo filter for compact representation, which is transferred to the client. (iii) During the *online phase* client and server jointly perform OPRF evaluations on all elements of the client. The client then looks up all received encryptions in the Cuckoo filter to determine the intersection. Thus, the complexity of the online phase is only linear in the size of the small client set. (iv) Changes in the server database trigger the *update phase*, where the Cuckoo filter on the client side is updated by sending a small delta for each inserted or deleted database entry.

4 Optimizing OPRF-based PSI Protocols

We propose more efficient database representations and PRFs, give the full descriptions for our optimized NR- and GC-

PSI protocols, enable security against malicious clients, and suggest multiple extensions to further increase practicality.

4.1 More Efficient Database Representations

Realistic Cuckoo Filter Parameters. Resende and de Freitas Aranha [59] propose using Cuckoo filters as an extension to the DH-based PSI protocol of [7] and they perform experiments to find optimal Cuckoo filter parameters based on the number of server items and the desired error probability. While their findings are directly applicable to our use case, they set very aggressive Cuckoo filter parameters (tagsize $v = 16$, bucket size $b = 3$) and settle for a maximum *false positive probability* (FPP) of $\varepsilon_{max} \approx 2^{-13}$. We find this FPP not practical since it implies that about one in 10 clients performing PSI for 2^{10} elements receives a false positive.

Instead, we propose to use tagsize $v = 32$ to reach a FPP of $\varepsilon_{max} \approx 2^{-29}$ or tagsize $v = 42$ to reach a FPP of $\varepsilon_{max} \approx 2^{-39}$ while still maintaining a bucket size of $b = 3$. For our experiments, we choose the parameter set $v = 32, b = 3$, and choose the size of the Cuckoo filter to have a load factor of $\approx 66\%$, leading to a Cuckoo filter size of 6 MiB per 2^{20} items.

Novel Cuckoo Filter Compression. The size of Cuckoo filters can be reduced by applying a simple but effective compression technique that to the best of our knowledge was not considered before: For each entry of a Cuckoo filter, an additional bit is transmitted that indicates whether this entry is empty or holds a tag. The entry itself is only transmitted if it is not empty. This way, the filter is represented as a bit map and a list of tags. For a Cuckoo filter storing n items with tagsize v, bucket size b, and load factor l, this reduces the size from $\frac{n}{l} \cdot v$ bits to $\frac{n}{l} + n \cdot v$ bits. In the example above, the size of the Cuckoo filter is reduced from 6 MiB to 4.19 MiB, i.e., by $\approx 30\%$. An advanced version of the compression technique presented above encodes the number of tags (0 to b) in each bucket with $\log_2(b+1)$ bits instead of sending b bits per bucket. This is possible since the actual position of each tag within a bucket is not important.

This compression technique is especially useful for very sparse Cuckoo filters, which appear in use cases where the set of items is expected to grow fast (e.g., during the release phase of a new messaging application). For example, if only 10 % of a Cuckoo filter storing a maximum of 2^{20} items is occupied, it can be compressed by a factor of 8.3x.

In concurrent and independent work, Breslow and Jayasena [15] proposed *Morton filters*, which combine these compression techniques with cache-optimized layouts and further optimizations. Morton filters provide higher insertion, lookup, and deletion throughput than traditional Cuckoo filters, while usually having equal or slightly lower storage costs. We leave the evaluation and usage of Morton filters in our protocols for future work.

Better Cuckoo Filter Updates. In [59], when performing an update after new elements are inserted into or deleted from the server's set, each encrypted element to be updated is sent to the client where it is inserted into the existing Cuckoo filter. However, for Cuckoo filters, all information required to insert a new item is its tag and the index of one of its candidate buckets. From this information, it is possible to calculate the second candidate bucket in case relocations are necessary. The same information is also sufficient to delete an item. For example, the bucket index in a Cuckoo filter storing $n = 2^{28}$ items with bucket size $b = 3$ and load factor $\approx 66\%$ can be represented with 27 bits. This results in sending 59 bits per updated element for tagsize $v = 32$. In comparison, in [59] an encrypted element is represented by one point on the GLS-254 binary elliptic curve, which results in 256 bits of communication when using point compression with two trace bits, which needs 4.3x more communication than our approach.

4.2 More Efficient PRF for GC-PSI

During the online phase of the GC-PSI protocol, both parties interactively evaluate an OPRF on the client's items using garbled circuits. For each of the client's items, the server prepares a garbled circuit $\widetilde{PRF}_k$ that evaluates the chosen PRF under the server's key k. The choice of this PRF has a significant impact on both the runtime and the communication complexity of the overall protocol. Several improvements for Yao's GC protocol [62] have appeared in recent years that changed the desired properties of the functions to be evaluated. Most notably is the Free-XOR [42] optimization, which allows XOR gates to be evaluated securely "for free", meaning all necessary operations can be performed locally without any communication between the parties. This optimization has lead to research in the area of ciphers with a low number of AND and instead many free XOR gates.

In previous GC-PSI implementations, the choice of the PRF was AES-128. Using the optimized S-Box implementation of [13], an AES-128 circuit (without key schedule) has 5,120 AND gates [32], serving as a baseline for comparison.

In this section, we focus on variants of LowMC [2], a highly parameterizable block cipher designed for use cases in multiparty computation (MPC) and fully-homomorphic encryption (FHE). [40] mentioned the possibility of using LowMC instead of AES for GC-PSI. We look at several instantiations of LowMC and present optimized parameter sets specifically for the use case of GC-PSI and mobile contact discovery. In the following, we give a short description of LowMC and highlight the different parameter choices.

LowMC [2] is a block cipher where block size n, key size k, number of S-Boxes per substitution layer m, and allowed data complexity d can be chosen freely up to some sanity constraints. The required number of rounds r to reach the security claims is then derived from these parameters.

Data Complexity. The data complexity of a cipher is the number of plaintext-ciphertext pairs allowed to be released before the security claims no longer hold. In the GC-PSI protocol, we can exactly control the maximum number of published plaintext-ciphertext pairs by limiting the number of client queries, and therefore can reduce the number of LowMC rounds required for security. We set the allowed data complexity to be $d = 2^{64}$, allowing for 2^{20} contact discoveries of 2^{10} items for each of the 2^{28} clients, while still being below the security margin by a factor of over 100x. For smaller-scale applications, we also give a parameter set for 2^{32} total data complexity, which suffices to run 2^{20} queries of 2^{10} items each. While we could also use this parameter set for larger-scale applications, the system needs to be re-keyed after the data complexity has been reached.

Key Schedule. In many MPC applications using OPRF evaluations, one party knows the entire secret key and can, therefore, perform any key-scheduling algorithm (e.g., for AES or LowMC) offline. The circuit is then modified to take the expanded key as an input. In many cases, this can be a performance improvement since the key-schedule algorithm does not have to be computed using the MPC protocol. However, when performing OPRF evaluations using garbled circuits, the party holding the secret key needs to send wire labels for each input bit, increasing the communication. While for AES-128, only 11x more wire labels need to be transferred for the expanded key, some instantiations of LowMC require several hundreds of rounds. Sending labels for the expanded key essentially removes the advantage of the lower AND count that comes with such a large number of rounds. However, we observe that in the GC-PSI protocol the OPRF evaluation is always performed with the same key. Thus, we can bundle all of the client's circuits together into one large circuit and evaluate the key-schedule only once. This means that we only need to send the wire labels corresponding to the non-expanded key once, and therefore save ≈ 2 KiB for each subsequent client item when using a 128-bit key. It is also possible to only evaluate parts of the garbled circuit if the number of client items is lower than the number of precomputed circuits.

LowMC Instances. For use in our GC-PSI protocol, we highlight several LowMC instances, exploring different parameter choices. In Tab. 2, we give the parameters and compare the number of AND gates to AES-128. The number of rounds is calculated according to the LowMCv3 round formula[5], which was updated by the LowMC team to take new cryptanalysis of LowMC (cf. [23, 24, 58]) into consideration. We can observe some interesting properties: LowMC instances (1) and (2) require the same number of rounds to be secure, but instance (1) has the maximum number of possible S-Boxes, while (2) does not. Since instance (2) provides the same security as (1) while requiring fewer S-Boxes, and therefore a lower amount of AND gates, it should always be preferred. LowMC instance (3) has the smallest possible S-Box layer with only one S-Box per round and also the lowest number of AND gates. While its 208 rounds can be a drawback in some protocols, Yao's GC protocol [62] has a constant number of communication rounds and therefore the large number of LowMC rounds does not decrease performance in high-latency networks. Additionally, using the optimizations presented by [22], the large number of linear layer computations can be reduced, bringing the evaluation time of (3) close to (1) and (2). For these reasons, we recommend the use of instance (3) for GC-PSI, which requires 8.2x fewer AND gates than standard AES-128 (6). Thus, we perform all performance evaluations using instance (3). For use cases with small data complexity requirements, we recommend LowMC instance (4), which is a small improvement of 8.3 % in runtime and communication compared to (3). For completeness and direct comparison to AES-128, we also give a variant of LowMC with data complexity of 2^{128} in (5).

[5] https://github.com/LowMC/lowmc/blob/master/determine_rounds.py

	PRF	n	k	m	d	r	#ANDs
(1)	LowMC	128	128	42	2^{64}	13	1,638
(2)	LowMC	128	128	31	2^{64}	13	1,209
(3)	**LowMC**	**128**	**128**	**1**	$\mathbf{2^{64}}$	**208**	**624**
(4)	LowMC	128	128	1	2^{32}	192	576
(5)	LowMC	128	128	1	2^{128}	287	861
(6)	AES-128	128	128	16	2^{128}	10	5,120

Table 2: Comparison of PRF instances for use in the GC-PSI protocol. The recommended instance is highlighted in bold.

4.3 Optimized GC-PSI Protocol

The idea of using Yao's GC protocol for OPRF evaluations was first proposed in [52] and used to construct a PSI protocol in the precomputation setting in [40].

The full protocol description is given in Fig. 1.We propose an optimization that halves the online communication for the OTs (which is the only communication in the online phase). This optimization is of independent interest as it improves the practicality of Yao's GC protocol in arbitrary use cases with precomputation. It is based on the observation that with the Free-XOR technique [42] for Yao's GC protocol [62], the client receives one of the two labels l^0 and $l^1 = l^0 \oplus \Delta$ via OT depending on its input bit, where l^0 is chosen at random and Δ is a random global constant only known by the garbler. A natural consideration would be to replace the real OTs, as used in [40], with correlated OTs (C-OTs) (cf. §3.1). Unfortunately, since the client input is unknown in the base phase, this prevents either the precomputation of the garbled circuits or the OTs. This is because in the online phase when using OT precomputation [8], the random messages r^0 and r^1 obtained by the sender in the base phase need to be swapped

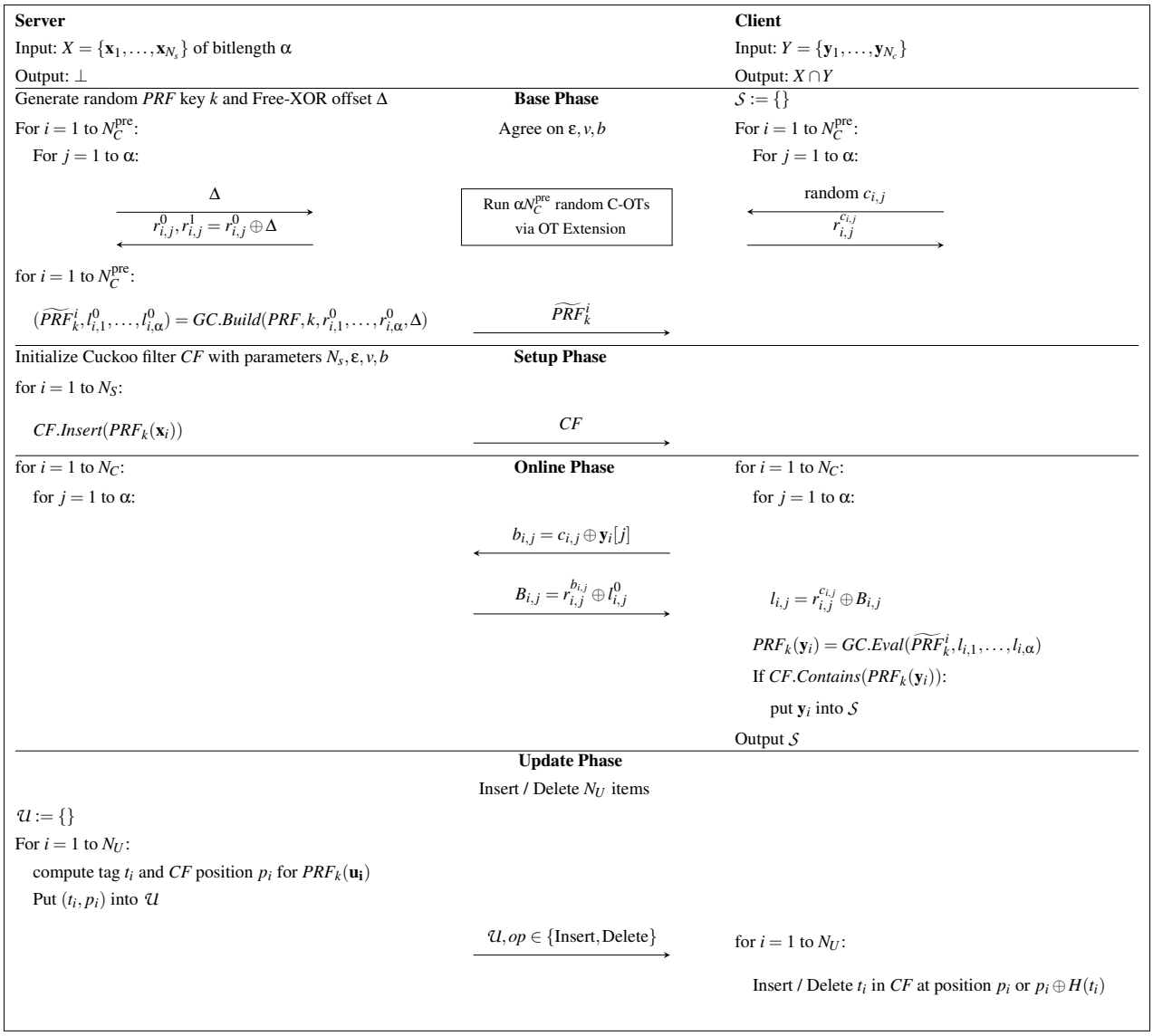


Figure 1: Our optimized GC-PSI protocol (based on [40, 52, 59]). Wire labels are computed as $l_{i,j}^0 = r_{i,j}^0 \oplus \delta_{i,j}$ and $l_{i,j}^1 = l_{i,j}^0 \oplus \Delta$, where the values $\delta_{i,j}$ are chosen at random while building the garbled circuit. $N_C^{\text{pre}} \geq N_C$ denotes the number of precomputed OTs and garbled circuits; the base phase must be repeated before further online phase executions once N_C^{pre} queries are exceeded.

in case the random choice made by the receiver differs from its actual input. Thus, it would be necessary to swap input wire labels in the garbled circuits, which requires recomputing and resending at least the first layer of those circuits.

Our novel precomputation method circumvents this dilemma: In the base phase we run C-OTs via OT extension s.t. the garbler on input Δ learns the random but correlated values r^0 and $r^1 = r^0 \oplus \Delta$, whereas the evaluator upon random choice c learns r^c. For garbling we choose the labels for the input wires of the circuit as $l^0 = r^0 \oplus \delta$ and $l^1 = l^0 \oplus \Delta$. Here, δ is a newly introduced random value that in contrast to Δ is not global but chosen individually for each label pair. In the online phase of the protocol, the evaluator sends a correction bit $b = c \oplus y$ stating whether its random choice c differs from the actual input y. The garbler responds with $B = r^b \oplus l^0$. This way, the evaluator learns either δ or $\delta \oplus \Delta$. It then sets the label for its input to $l = r^c \oplus B$. As one can easily verify for the four possible combinations of random choices c and correction bits b, the evaluator always retrieves the correct label.

The security of the C-OT precomputation is based on the same arguments as standard OT precomputation [8] and since we use a fresh uniformly random δ for each wire label, the resulting wire label is also uniformly random. In other words, we resolve the problem by fixing the wire labels but if necessary swapping the masks required to retrieve the correct label from the initial C-OT result.

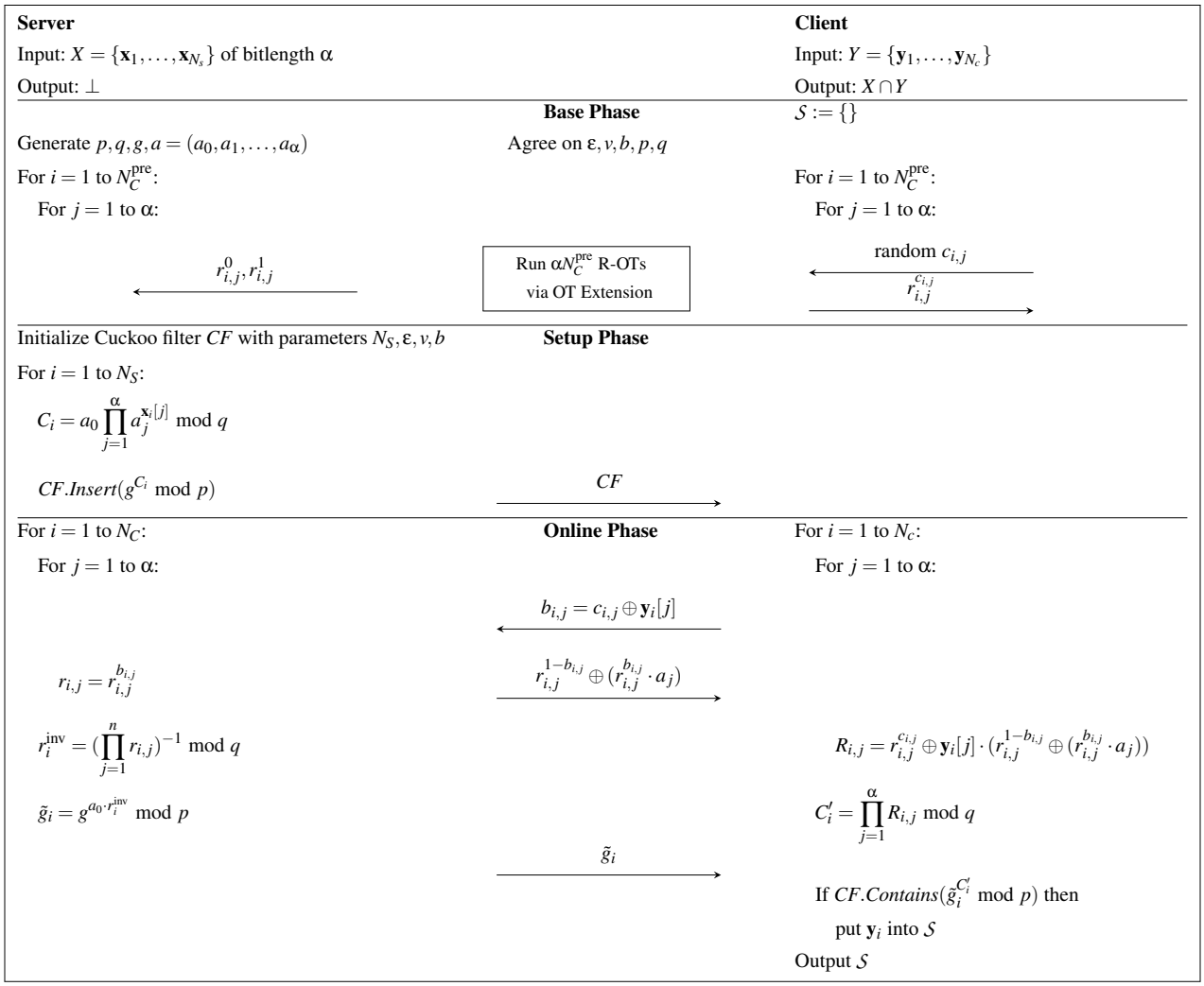


Figure 2: Our optimized NR-PSI protocol (based on [31, 40, 59]). When using a plain finite field, the modulus p is prime, q is a prime divisor of $p-1$, $g \in \mathbb{Z}_p^*$ is of order q, and $a_0, a_1, \ldots, a_\alpha$ as well as $r_{i,j}^0, r_{i,j}^1$ are random numbers in $\mathbb{Z}_q^*$. The update phase is omitted since it is similar to the GC-PSI protocol (cf. Fig. 1), except using the NR-PRF to compute tag t_i and CF position p_i.

4.4 Optimized NR-PSI Protocol

The usage of the Naor-Reingold PRF (NR-PRF) [47] for PSI was first proposed in [31] and the resulting PSI protocol transformed into the precomputation setting in [40]. The NR-PRF for key k and element x_i is defined as

$$f_k(x_i) = g^{a_0 \cdot \prod_{j=1}^{\alpha} a_i^{x_{i,j}}} \mod p, \tag{1}$$

where, when using a plain finite field, p is a prime, q is a prime divisor of $p-1$, $g \in \mathbb{Z}_p^*$ is a generator of order q, $a_0, a_1, \ldots, a_\alpha$ are random numbers in $\mathbb{Z}_q^*$ forming key k, and α is the bitlength of element x_i.

Among all protocols for mobile contact discovery evaluated in [40], NR-PSI is the only protocol besides GC-PSI that can easily be made secure against malicious clients by employing malicious secure OT extensions (cf. §4.5). Furthermore, according to the empirical performance comparison in [40], the NR-PSI protocol causes $\approx$ 30x less communication overhead than GC-PSI without our optimizations. This is why we also consider the NR-PSI protocol in this work and compare it to our optimized GC-PSI implementation in §6.

The full protocol description is given in Fig. 2. We propose an optimization that improves the online communication for OTs by factor 2x. The optimization is based on the observation that in the definitions of [31] the client chooses between a random r and $r \cdot a$ depending on the current bit of its input element. This implies that C-OTs (cf. §3.1) can be used instead of real OTs, thereby sending only one message in the size of the symmetric security parameter instead of the two messages when using the OTe protocols of [3].

Since we use the precomputation form of [40], we propose a novel combination of OT precomputation [8] and C-OT [3]. As in OT precomputation, the client sends a correction bit b stating whether its random choice c in the precomputation

phase equals its real input. Depending on b, the server then decides which of the two random messages obtained during OT precomputation is chosen as r and which is used to mask the correlated message $r \cdot a$ that is sent to the client. Likewise, the client either proceeds with the message obtained during OT precomputation as r or uses this message to unmask the received correlated message.

4.5 Malicious Security

As observed already in [40], the only messages sent by the client in the GC-PSI and NR-PSI protocols are those in the base OT and OT extension protocols as well as the correction bits during the online phase when applying OT precomputation [8]. Therefore, both protocols can easily be made secure against a malicious client by using a maliciously secure OTe protocol such as [4] or [39], together with maliciously secure base OTs such as [50]. As the OT extension contributes only a small percentage to the total runtime of the PSI protocols and today's maliciously secure OTe protocols are only slightly less efficient than the passively secure OT extension of [3], the total runtime of the PSI protocols does not increase by a noticeable amount when replacing the OTe protocols. Please note that enumeration attacks (i.e., querying the server repeatedly with different inputs) are still possible when using our protocols. However, even an ideal functionality for PSI (e.g., a trusted third party) and currently deployed non-private contact discovery methods cannot prevent this. We recommend to employ well-established measures like rate limiting to mitigate such attacks.

The case of a malicious server is different: it could, for example, send wrong wire labels, use wrong circuit descriptions, or send a wrong server set. In general, the client does not reveal the intersection result to the server, so a malicious server can only influence the correctness of the client's computation, but cannot learn any information about the client's items when using maliciously secure OTs. Unfortunately, in most mobile messaging applications, the client sends information about the intersection (most likely even the entire intersection) to the server. This allows a malicious server to learn information about the client's items that are not part of the intersection of the two actual input sets. Therefore, we need to assume a semi-honest server in such scenarios. Preventing malicious behavior on the server side could be done by combining our protocols with a trusted execution environment for hardware-enforced code and remote attestation capabilities s.t. the server's protocol deviation possibilities are restricted to wrong inputs for the Cuckoo filter construction. However, assuming a semi-honest server is reasonable since there are legal requirements and financial incentives for a service provider to behave correctly: once misconduct gets known publicly, users will abandon the malicious service and switch to a more trustworthy alternative.

4.6 Further Extensions

The bottleneck for very large server sets is the communication required to send the Cuckoo filter to the client. For example, a compressed Cuckoo filter for 2^{28} server items with false positive probability $\varepsilon_{max} \approx 2^{-29}$ has a size of ≈ 1 GiB, which is prohibitively large for transmission on mobile network speeds and data plans. For even larger server databases, the protocols eventually become impractical. For example, for a server database with 2^{31} entries, it would be necessary to download a Cuckoo filter of size ≈ 8 GiB. Therefore, we describe how to reduce the overall client-server communication to be logarithmic in the size of the server database. We propose further extensions to increase practicality in App. A.

Combination with Private Information Retrieval (PIR). In their PIR-PSI protocol, Demmler et al. [21] propose the use of multiple non-colluding servers together with a multi-server PIR protocol. Applied to our PSI protocols, the extension works as follows: After the server prepared the Cuckoo filter, it is not transmitted to the client, but to a second non-colluding server instead. Since the Cuckoo filter only contains the results of PRF evaluations, the second server does not learn anything about the items in the main server's set. The client then performs the OPRF evaluation for each of its items with the first server and then runs a multi-server PIR protocol to retrieve the fingerprints stored in the Cuckoo filter.

The communication complexity for the multi-server PIR lookup is $O(\kappa \log n)$, where κ is the symmetric security parameter and n the size of the server database [14, 21]. Since the overall client-server communication therefore is logarithmic and not linear in the size of the server database, our protocols are expected to remain practical even for server databases with more than a billion items. In practice, the remaining challenge for messaging services is to find a trustworthy partner operating the second PIR server while at the same time making it credible to users that no collusion is happening.

5 Android Implementation

To demonstrate the feasibility of our optimized PSI protocols for performing private contact discovery on mobile devices, we provide implementations for smartphones running on Android.[6] Previous works [34, 40] presented experiments on dedicated mobile devices, but the performance of these implementations was not sufficient for real-world usage. For example, the Java implementation of [40], which is based on the ObliVM framework [43], takes more than a second to evaluate a single garbled AES-128 circuit. In our implementation, we make use of native C/C++ code support in Android and also use hardware acceleration for cryptographic operations available in modern smartphones. More precisely, native

[6] https://contact-discovery.github.io

AES-128 instructions are used both as a PRNG and during the creation and evaluation of the garbled circuit. These features allow our implementation to reach truly practical performance. Compared to the Java-based implementation of [40], we evaluate a garbled AES-128 circuit more than 1,000x faster.

5.1 Base OTs and OT Extension

For performing base OTs, we use the OT protocol of Chou and Orlandi [20] with the additional verification step proposed by Doerner et al. [26]. Together with the (C-)OT extension protocol of Keller, Orsini, and Scholl [39], this results in a maliciously secure protocol (cf. [26]).

Our OT implementation is based on `libOTe` by Rindal [60], which is heavily optimized for the x86 architecture. Thus, we ported large parts of the library to the ARMv8 architecture to achieve high performance on mobile devices. At the same time, we kept the library compatible with its x86 counterpart to facilitate natural development of client-server applications.

5.2 GC-PSI Implementation

For the GC-PSI protocol, we implement Yao's GC protocol (cf. §3.2) with Free-XOR [42] and Half-Gates [63], resulting in no communication for XOR-gates and two wire labels of κ bits each per AND gate, where $\kappa = 128$ is the symmetric security parameter.

For creating and evaluating the garbled tables, the most efficient choice today is fixed-key AES [11], mainly due to the hardware support for AES that is widespread in modern x86 CPUs. The ARM Cryptography Extensions (CE) introduced in the ARMv8 architecture similarly provide hardware instructions for AES, SHA-1, and SHA-2 variants, resulting in AES speedups of factor 35x compared to a standard AES software implementation. This allows us to also use fixed-key AES [11] for garbling in our implementation.[7] Additionally, the ARMv8 architecture provides instructions for vector operations on 128-bit registers (the so-called NEON instruction set), which we use to efficiently work with 128-bit wire labels. In Tab. 7 in App. B, we demonstrate the wide availability of ARM CE in most recent smartphone processors.

5.3 NR-PSI Implementation

For implementing the NR-PSI protocol, we use the modified `libOTe` version described in §5.1 for C-OT precomputation as well as the `GNU GMP`[8] library for modular arithmetic operations and the `MIRACL`[9] library for instantiating the protocol with elliptic curve P-256. The advantage of instantiating the NR-PSI protocol with ECC instead of using a plain finite field with comparable security parameters is that the size of the values $\tilde{g}_i$ transferred during the online phase (cf. Fig. 2) is reduced by factor 8x. Also, computationally expensive modular exponentiations are replaced with point multiplications. We refer to this variant as ECC-NR-PSI in the following. All libraries are compiled specifically for the ARMv8 architecture.

[7] As recently reported by [29], many secure computation implementations use fixed-key AES incorrectly. However, according to [29], our instantiation for garbling following the definitions of [63] is not affected. In contrast, `libOTe` [60] is currently vulnerable. The suggested fixes however are not expected to result in a significant negative performance impact [29].

[8] https://gmplib.org

[9] https://github.com/miracl/MIRACL

6 Performance Evaluation

We empirically evaluate the performance of our optimized GC-PSI and NR-PSI protocols and compare them to other unbalanced PSI protocols from the literature.

Benchmark Settings. For easy comparison to related work, we choose similar sizes for the server's and the client's set: $N_s \in \{2^{20}, 2^{24}, 2^{26}, 2^{28}\}$ and $N_c \in \{1, 2^8, 2^{10}\}$. Here, $N_c = 1$ represents the case where a client wants to check a new contact. All items have a bitlength of $\alpha = 128$. We instantiate all primitives and protocols with 128-bit security.

In all of our experiments, the sever is equipped with an Intel Core^{TM} i7-4600U CPU @ 2.6 GHz and 16 GiB of RAM. The client is a Google Pixel XL 2 smartphone with a Snapdragon 835 CPU @ 2.45 GHz and 4 GiB of RAM. We consider two network settings: (i) an IEEE 802.11ac WiFi connection with $\approx$ 230 Mbit/s down-/upload and 70 ms RTT and (ii) a mobile LTE connection with 42 Mbit/s download ($S \rightarrow C$), 4 Mbit/s upload ($S \leftarrow C$), and 80 ms RTT.

Note that the LTE network speeds are real-world parameters and exhibit a significant difference in the down- and upload rates. This is common in commercially available data plans and often not taken into account in previous evaluations.

6.1 GC-PSI and NR-PSI Protocol

The runtime and communication costs for the base, setup, and online phase of our protocols are shown in Tab. 3, Tab. 4, and Tab. 5, respectively, and are averaged over 100 executions (except for the setup phase, where we chose 10 or less executions due to the larger runtime). We use LowMC instance (3) from Tab. 2 for the evaluation. In all tests, only a single thread was used for both the server and the client. Since all phases of our protocols can be parallelized trivially, we expect a near-linear speedup when using multiple threads, except in situations where the bottleneck is network bandwidth. Furthermore, note that in the base and online phases of the GC-PSI protocol, only one party actually performs the computationally expensive task of garbling or evaluating the circuit. Therefore, if both parties are ready, the base and online phases of the GC-PSI protocol can be interleaved in a pipelined fashion, where the server sends the garbled circuits and the client evaluates them as soon as parts of them are available. This

Parameters		Time [s]		Comm. [MiB]	
N_c^{pre}	Protocol	WiFi	LTE	$S \rightarrow C$	$S \leftarrow C$
2^{10}	AES-GC-PSI	7.14	38.98	162.52	2.02
	LowMC-GC-PSI	1.85	6.57	22.01	2.02
	ECC-NR-PSI	**0.61**	**4.21**	**0.01**	**1.99**

Table 3: Base phase of our PSI protocols. Precomputation for checking N_c^{pre} client contacts. Best results marked in bold.

Parameters		Server Setup [s]	Transmission [s]		Comm. [MiB]
N_s	Protocol		WiFi	LTE	$S \rightarrow C$
2^{28}	AES-GC-PSI	**23.94**	32.66	211.30	1072
	LowMC-GC-PSI	1,869.13			
	ECC-NR-PSI	52,332.38			
2^{26}	AES-GC-PSI	**4.87**	8.13	52.55	268
	LowMC-GC-PSI	467.29			
	ECC-NR-PSI	12,787.79			
2^{24}	AES-GC-PSI	**1.12**	2.13	13.05	67
	LowMC-GC-PSI	116.66			
	ECC-NR-PSI	3,297.96			
2^{20}	AES-GC-PSI	**0.06**	0.25	0.63	4.19
	LowMC-GC-PSI	7.27			
	ECC-NR-PSI	241.54			

Table 4: Setup phase of our PSI protocols. Server setup run once for *all* clients. The Cuckoo filter parameters are set as described in §4.1 ($\varepsilon_{max} = 2^{-29.4}, v = 32, b = 3$). Best results marked in bold. Note that the size of the client set does not influence the runtime of the setup phase and the client does not send any data during the setup phase in any protocol.

method can reduce the runtime of the combined base and online phase to the runtime of the slower phase.

We observe that using LowMC instead of AES in the GC-PSI protocol leads to 7.4x less communication and thus to a much smaller runtime in the base phase, while the online phase of both protocol versions is very comparable. Only during the one-time setup phase, the AES version is more efficient due to AES-NI instructions. Using a hardware-accelerated implementation of LowMC could reduce this runtime close to the one of AES, but we again stress that the setup phase is a one-time cost. This confirms our choice of LowMC over AES as the PRF in GC-PSI.

ECC-NR-PSI is the most efficient protocol during the base phase since it does not send garbled circuits to the client: compared to the LowMC version of GC-PSI, it requires 12x less communication. The ECC-NR-PSI online phase is slightly slower than both GC-PSI protocols, while being the fastest for a single item. The one-time setup phase of the ECC-NR-PSI protocol is much slower than both GC-PSI protocol versions due to elliptic curve operations.

6.2 Comparison with Related Work

We now highlight differences to other works in the literature and compare our optimized GC- and NR-PSI protocols and implementations to other unbalanced PSI implementations available for Android in Tab. 6. Comparisons with implementations for the x86 architecture are given in App. D.

Parameters		Time [s]		Comm. [KiB]	
N_c	Protocol	WiFi	LTE	$S \rightarrow C$	$S \leftarrow C$
2^{10}	AES-GC-PSI	**1.43**	**1.86**	**2,048**	**16.00**
	LowMC-GC-PSI	1.71	2.02	**2,048**	**16.00**
	ECC-NR-PSI	2.31	2.32	4,147	**16.00**
2^8	AES-GC-PSI	**0.34**	**0.47**	**512**	**4.00**
	LowMC-GC-PSI	0.37	0.48	**512**	**4.00**
	ECC-NR-PSI	0.61	0.61	1,037	**4.00**
1	AES-GC-PSI	0.03	0.03	**2.00**	**0.02**
	LowMC-GC-PSI	0.04	0.05	**2.00**	**0.02**
	ECC-NR-PSI	**0.01**	**0.02**	4.06	0.04

Table 5: Online phase of our PSI protocols. Best results marked in bold. The influence of the server set size on runtime and communication is negligible and therefore not listed.

Chen et al. [18, 19]. The protocols of [18, 19] for unbalanced PSI are based on leveled fully homomorphic encryption (FHE). They both work as follows: the client encrypts all its items and sends them to the server, which then computes the intersection under encryption with all of its own items and returns the result in encrypted form. The client can then decrypt the received ciphertexts to find the intersection.

The protocol in [19] is only defined for 32 bit strings, a limitation that stems from the parameter choice of the FHE scheme. Since the universe of possible items is larger than 2^{32} in the use case of contact discovery, we exclude this protocol from further comparisons. However, this limitation was lifted in the subsequent work [18] where arbitrary length items are supported. The benefits of [18] compared to our protocols are that the client is not required to store any data and that the total communication is sublinear in the size of the server database. For example, for $N_s = 2^{28}$, the total communication in the protocol of [18] is only 18.4 MB.

However, there is a huge computational overhead during the online phase of the protocol: even on a high-end server it takes more than 12 s on 32 threads to compute the intersection with $N_c = 1024$ client elements. Unfortunately, the online phase needs to be repeated whenever there are updates on client or server side. Also, due to the employed FHE batching optimizations, the runtime for a single item is almost equal to the runtime for thousands of items. Assuming that each of the $N_s = 2^{28}$ registered clients runs one update per day, this would require the service provider to pay for $2^{28} \cdot 12.1 \cdot 32 \approx 28.9$ million core hours every day. In contrast, the online phases of our protocols run in ≈ 2 s for $N_c = 1024$ in the WiFi setting on a single-threaded smartphone and require no cryptographic operations on server side. The evaluation of [18] was performed on two servers with Intel Xeon CPUs in a 10 Gbit/s local network. Therefore, it is also unclear how the FHE encryption and decryption routines perform in a mobile setting on real smartphones.

Resende and de Freitas Aranha [59]. In [59], the authors present implementation improvements for the PSI protocol of [7]. For each element in the client's set, they perform 3

Parameters N_s	Parameters N_c	PSI Protocol	Base + Online Time [s] WiFi	Base + Online Time [s] LTE	Communication [MiB] $S \rightarrow C$	Communication [MiB] $S \leftarrow C$	Setup Communication / Client Storage [MiB]	Setup Transfer [s] WiFi	Setup Transfer [s] LTE	Server Setup [s]
2^{28}	1,024	AES-GC-PSI [40]	1,507.73	2,742.66	177.23	4.00	1,380.25	42.05	272.06	**26.70**
		NR-PSI [40]	171.23	221.20	64.25	2.02	1,380.25	42.05	272.06	194,130.21
		LowMC-GC-PSI (Ours)	3.54	8.59	22.01	2.02	**1,072.00**	**32.66**	**211.30**	1,869.13
		ECC-NR-PSI (Ours)	**2.92**	**6.53**	**4.07**	**2.00**	**1,072.00**	**32.66**	**211.30**	52,332.38
	1	AES-GC-PSI [40]	1.53	2.95	0.18	0.02	1,380.25	42.05	272.06	**26.70**
		NR-PSI [40]	0.17	0.21	0.06	**0.01**	1,380.25	42.05	272.06	194,130.21
		LowMC-GC-PSI (Ours)	0.17	0.18	0.04	0.02	**1,072.00**	**32.66**	**211.30**	1,869.13
		ECC-NR-PSI (Ours)	**0.13**	**0.13**	**0.01**	**0.01**	**1,072.00**	**32.66**	**211.30**	52,332.38
2^{24}	1,024	AES-GC-PSI [40]	1,507.73	2,742.66	177.23	4.00	86.26	2.74	16.80	**1.18**
		NR-PSI [40]	171.23	221.20	64.25	2.02	86.26	2.74	16.80	12,174.40
		LowMC-GC-PSI (Ours)	3.54	8.59	22.01	2.02	**67.00**	**2.13**	**13.05**	116.66
		ECC-NR-PSI (Ours)	**2.92**	**6.53**	**4.07**	**2.00**	**67.00**	**2.13**	**13.05**	3,297.96
	1	AES-GC-PSI [40]	1.53	2.95	0.18	0.02	86.26	2.74	16.80	**1.18**
		NR-PSI [40]	0.17	0.21	0.06	**0.01**	86.26	2.74	16.80	12,174.40
		LowMC-GC-PSI (Ours)	0.17	0.18	0.04	0.02	**67.00**	**2.13**	**13.05**	116.66
		ECC-NR-PSI (Ours)	**0.13**	**0.13**	**0.01**	**0.01**	**67.00**	**2.13**	**13.05**	3,297.96
2^{20}	1,024	AES-GC-PSI [40]	1,507.73	2,742.66	177.23	4.00	5.39	0.32	0.81	**0.05**
		NR-PSI [40]	171.23	221.20	64.25	2.02	5.39	0.32	0.81	758.40
		LowMC-GC-PSI (Ours)	3.54	8.59	22.01	2.02	**4.19**	**0.25**	**0.63**	7.27
		ECC-NR-PSI (Ours)	**2.92**	**6.53**	**4.07**	**2.00**	**4.19**	**0.25**	**0.63**	241.54
	1	AES-GC-PSI [40]	1.53	2.95	0.18	0.02	5.39	0.32	0.81	**0.05**
		NR-PSI [40]	0.17	0.21	0.06	**0.01**	5.39	0.32	0.81	758.40
		LowMC-GC-PSI (Ours)	0.17	0.18	0.04	0.02	**4.19**	**0.25**	**0.63**	7.27
		ECC-NR-PSI (Ours)	**0.13**	**0.13**	**0.01**	**0.01**	**4.19**	**0.25**	**0.63**	241.54

Table 6: Comparison of PSI protocols with smartphone implementations. Numbers for protocols of [40] are obtained by running their implementations in our benchmarking environment. In all tests $N_c^{\text{pre}} = N_c$. Best in class marked in bold.

point multiplications and transmit 2 group elements. This results in a lower communication than our approaches (64 B for 2 group elements vs. 22 KiB per garbled circuit vs. 6 KiB per item in NR-PSI). However, one major contribution of [59] is a significant optimization of the GLS-254 curve for x86 CPUs. It is therefore unclear how their protocol performs on smartphones with ARMv8-A hardware. Furthermore, their Cuckoo filters parameters allow for a false positive probability that is too high for real-world deployment (cf. §4.1). Finally, their protocol assumes semi-honest adversaries, and while a maliciously secure variant [38] of their basic protocol exists, its performance has not yet been evaluated.

Kiss et al. [40]. In [40], the authors consider various semi-honest PSI protocols, from which their GC-PSI and NR-PSI protocols are the foundation of our work. Their Android implementation (in pure Java) takes about 1.5 s for a single oblivious AES evaluation in their GC-PSI protocol. The authors therefore conclude that instead their ECC-DH-PSI protocol is most suited for the mobile use case since the evaluation time for a single item is 23 ms. However, both of our optimized protocols with security against malicious clients are more than competitive with an evaluation time of less than 2 ms for a single item. For $N_c = 1024$ client elements, the combined base and online time of our optimized GC- and NR-PSI protocols improves by more than a factor of 300x and 30x, respectively, compared to the unoptimized semi-honest implementations of [40] in both the WiFi and the LTE network setting. Also, the total communication during the base and online phase improves by factors 7.5x and 10.9x compared to the respective GC- and NR-PSI protocols of [40].

7 Conclusion

Our native implementations of our optimized NR- and GC-PSI protocols are two almost equivalently outstanding solutions for large-scale mobile private contact discovery with security against malicious clients. The Signal developers stated that to actually deploy PSI-based contact discovery, it would need to be able to handle a server database with 1 billion users while address books are assumed to contain up to 10,000 contacts. In terms of latency, lookups are required to take less than 2 s, while in terms of throughput a single core should be able to handle 1,600 contacts per second. Clearly, we cannot meet these demanding requirements yet. Therefore, as part of future work, we suggest to implement and evaluate our proposed extensions (especially the combination with PIR) to take the next important steps towards real-world deployment.

Acknowledgments

This work was co-funded by the DFG as part of project E4 within the CRC 1119 CROSSING and project A.1 within the RTG 2050 “Privacy and Trust for Mobile Users”, by the BMBF and the HMWK within CRISP, and by the European Union’s Horizon 2020 research and innovation programme under grant agreement No 644052 (HECTOR). Daniel Kales has been supported by iov42 Ltd.

References

[1] WhatsApp Legal Info. https://www.whatsapp.com/legal, 2019.

[2] Martin R. Albrecht, Christian Rechberger, Thomas Schneider, Tyge Tiessen, and Michael Zohner. Ciphers for MPC and FHE. In *EUROCRYPT*, volume 9056 of *LNCS*, pages 430–454. Springer, 2015.

[3] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer and Extensions for Faster Secure Computation. In *CCS*, pages 535–548. ACM, 2013.

[4] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer Extensions with Security for Malicious Adversaries. In *EUROCRYPT*, volume 9056 of *LNCS*, pages 673–701. Springer, 2015.

[5] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More Efficient Oblivious Transfer Extensions. *Journal of Cryptology*, 30(3):805–858, 2017.

[6] N. Asokan, Alexandra Dmitrienko, Marcin Nagy, Elena Reshetova, Ahmad-Reza Sadeghi, Thomas Schneider, and Stanislaus Stelle. CrowdShare: Secure Mobile Resource Sharing. In *ACNS*, volume 7954 of *LNCS*, pages 432–440. Springer, 2013.

[7] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering GATTACA: Efficient and Secure Testing of Fully-Sequenced Human Genomes. In *CCS*, pages 691–702. ACM, 2011.

[8] Donald Beaver. Precomputing Oblivious Transfer. In *CRYPTO*, volume 963 of *LNCS*, pages 97–109. Springer, 1995.

[9] Donald Beaver. Correlated Pseudorandomness and the Complexity of Private Computations. In *STOC*, pages 479–488. ACM, 1996.

[10] Donald Beaver, Silvio Micali, and Phillip Rogaway. The Round Complexity of Secure Protocols (Extended Abstract). In *STOC*, pages 503–513. ACM, 1990.

[11] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient Garbling from a Fixed-Key Blockcipher. In *IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society, 2013.

[12] Burton H. Bloom. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, 1970.

[13] Joan Boyar and René Peralta. A New Combinational Logic Minimization Technique with Applications to Cryptology. In *Symposium on Experimental Algorithms*, volume 6049 of *LNCS*, pages 178–189. Springer, 2010.

[14] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing: Improvements and Extensions. In *CCS*, pages 1292–1303. ACM, 2016.

[15] Alexander Breslow and Nuwan Jayasena. Morton Filters: Faster, Space-Efficient Cuckoo Filters via Biasing, Compression, and Decoupled Logical Sparsity. *Proceedings of the VLDB Endowment (PVLDB)*, 11(9):1041–1055, 2018.

[16] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *USENIX Security*, pages 991–1008. USENIX Association, 2018.

[17] Henry Carter, Benjamin Mood, Patrick Traynor, and Kevin R. B. Butler. Secure Outsourced Garbled Circuit Evaluation for Mobile Devices. In *USENIX Security*, pages 289–304. USENIX Association, 2013.

[18] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *CCS*, pages 1223–1237. ACM, 2018.

[19] Hao Chen, Kim Laine, and Peter Rindal. Fast Private Set Intersection from Homomorphic Encryption. In *CCS*, pages 1243–1255. ACM, 2017.

[20] Tung Chou and Claudio Orlandi. The Simplest Protocol for Oblivious Transfer. In *LATINCRYPT*, volume 9230 of *LNCS*, pages 40–58. Springer, 2015.

[21] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling Private Contact Discovery. *PoPETs*, 2018(4):159–178, 2018.

[22] Itai Dinur, Daniel Kales, Angela Promitzer, Sebastian Ramacher, and Christian Rechberger. Linear Equivalence of Block Ciphers with Partial Non-Linear Layers: Application to LowMC. In *EUROCRYPT*, volume 11476 of *LNCS*, pages 343–372. Springer, 2019.

[23] Itai Dinur, Yunwen Liu, Willi Meier, and Qingju Wang. Optimized Interpolation Attacks on LowMC. In *ASIACRYPT*, volume 9453 of *LNCS*, pages 535–560. Springer, 2015.

[24] Christoph Dobraunig, Maria Eichlseder, and Florian Mendel. Higher-Order Cryptanalysis of LowMC. In *ICISC*, volume 9558 of *LNCS*, pages 87–101. Springer, 2015.

[25] Yevgeniy Dodis and Aleksandr Yampolskiy. A Verifiable Random Function with Short Proofs and Keys. In *PKC*, volume 3386 of *LNCS*, pages 416–431. Springer, 2005.

[26] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Secure Two-party Threshold ECDSA from ECDSA Assumptions. In *IEEE Symposium on Security and Privacy*, pages 980–997. IEEE Computer Society, 2018.

[27] Bin Fan, David G. Andersen, Michael Kaminsky, and Michael Mitzenmacher. Cuckoo Filter: Practically Better Than Bloom. In *Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, pages 75–88. ACM, 2014.

[28] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword Search and Oblivious Pseudorandom Functions. In *TCC*, volume 3378 of *LNCS*, pages 303–324. Springer, 2005.

[29] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. Efficient and Secure Multiparty Computation from Fixed-Key Block Ciphers. *IACR Cryptology ePrint Archive*, 2019:074, 2019. https://ia.cr/2019/074.

[30] Carmit Hazay and Yehuda Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. In *TCC*, volume 4948 of *LNCS*, pages 155–175. Springer, 2008.

[31] Carmit Hazay and Yehuda Lindell. Efficient Protocols for Set Intersection and Pattern Matching with Security Against Malicious and Covert Adversaries. *Journal of Cryptology*, 23(3):422–456, 2010.

[32] Wilko Henecka and Thomas Schneider. Faster secure two-party computation with less memory. In *ASIACCS*, pages 437–446. ACM, 2013.

[33] Kashmir Hill. Facebook recommended that this psychiatrist's patients friend each other. https://splinternews.com/facebook-recommended-that-this-psychiatrists-patients-f-1793861472, 2016.

[34] Yan Huang, Peter Chapman, and David Evans. Privacy-Preserving Applications on Smartphones. In *HotSec*, pages 4–4. USENIX Association, 2011.

[35] Russell Impagliazzo and Steven Rudich. Limits on the Provable Consequences of One-way Permutations. In *STOC*, pages 44–61. ACM, 1989.

[36] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending Oblivious Transfers Efficiently. In *CRYPTO*, volume 2729 of *LNCS*, pages 145–161. Springer, 2003.

[37] Stanislaw Jarecki and Xiaomin Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*, volume 5444 of *LNCS*, pages 577–594. Springer, 2009.

[38] Stanislaw Jarecki and Xiaomin Liu. Fast Secure Computation of Set Intersection. In *SCN*, volume 6280 of *LNCS*, pages 418–435. Springer, 2010.

[39] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively Secure OT Extension with Optimal Overhead. In *CRYPTO*, volume 9215 of *LNCS*, pages 724–741. Springer, 2015.

[40] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private Set Intersection for Unequal Set Sizes with Mobile Applications. *PoPETs*, 2017(4):177–197, 2017.

[41] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In *CCS*, pages 818–829. ACM, 2016.

[42] Vladimir Kolesnikov and Thomas Schneider. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[43] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. ObliVM: A Programming Framework for Secure Computation. In *IEEE Symposium on Security and Privacy*, pages 359–376. IEEE Computer Society, 2015.

[44] Moxie Marlinspike. The Difficulty Of Private Contact Discovery. https://signal.org/blog/contact-discovery, 2014.

[45] Moxie Marlinspike. Technology Preview: Private Contact Discovery for Signal. https://signal.org/blog/private-contact-discovery, 2017.

[46] Benjamin Mood, Debayan Gupta, Kevin R. B. Butler, and Joan Feigenbaum. Reuse It Or Lose It: More Efficient Secure Computation Through Reuse of Encrypted Values. In *CCS*, pages 582–596. ACM, 2014.

[47] Moni Naor and Omer Reingold. Number-Theoretic Constructions of Efficient Pseudo-Random Functions. *Journal of the ACM*, 51(2):231–262, 2004.

[48] Rasmus Pagh and Flemming Friche Rodler. Cuckoo Hashing. In *Annual European Symposium on Algorithms*, volume 2161 of *LNCS*, pages 121–133. Springer, 2001.

[49] Panagiotis Papadopoulos, Antonios A. Chariton, Elias Athanasopoulos, and Evangelos P. Markatos. Where's Wally?: How to Privately Discover your Friends on the Internet. In *ASIACCS*, pages 425–430. ACM, 2018.

[50] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A Framework for Efficient and Composable Oblivious Transfer. In *CRYPTO*, volume 5157 of *LNCS*, pages 554–571. Springer, 2008.

[51] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security*, pages 515–530. USENIX Association, 2015.

[52] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure Two-Party Computation Is Practical. In *ASIACRYPT*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.

[53] Benny Pinkas, Thomas Schneider, Oleksandr Tkachenko, and Avishay Yanai. Efficient Circuit-based PSI with Linear Communication. In *EUROCRYPT*, volume 11476 of *LNCS*, pages 122–153. Springer, 2019.

[54] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. Efficient Circuit-Based PSI via Cuckoo Hashing. In *EUROCRYPT*, volume 10822 of *LNCS*, pages 125–157. Springer, 2018.

[55] Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster Private Set Intersection Based on OT Extension. In *USENIX Security*, pages 797–812. USENIX Association, 2014.

[56] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable Private Set Intersection Based on OT Extension. *ACM Transactions on Privacy and Security*, 21(2):7:1–7:35, 2018.

[57] Michael Rabin. How to Exchange Secrets with Oblivious Transfer. In *Technical Report TR-81*. Aiken Computation Laboratory: Harvard University, 1981.

[58] Christian Rechberger, Hadi Soleimany, and Tyge Tiessen. Cryptanalysis of Low-Data Instances of Full LowMCv2. *IACR Transactions on Symmetric Cryptology*, 2018(3):163–181, 2018.

[59] Amanda Cristina Davi Resende and Diego de Freitas Aranha. Faster Unbalanced Private Set Intersection. In *FC*, LNCS. Springer, 2018.

[60] Peter Rindal. libOTe: A fast, portable, and easy to use Oblivious Transfer Library. https://github.com/osu-crypto/libOTe.

[61] Statista. Most Popular Global Mobile Messenger Apps. https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps, 2019.

[62] Andrew Chi-Chih Yao. How to Generate and Exchange Secrets (Extended Abstract). In *FOCS*, pages 162–167. IEEE, 1986.

[63] Samee Zahur, Mike Rosulek, and David Evans. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In *EUROCRYPT*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.

A Protocol Extensions

We propose further extensions for improving practicality.

Combination with FHE Protocols. Protocols for unbalanced PSI based on fully homomorphic encryption (FHE), e.g., [18], are computationally expensive and thus much slower during the online phase than our protocols (cf. §6.2). However, their advantage is that the total amount of communication is sublinear in the size of the server database. When clients install a new messaging application and are not connected to a high-speed WiFi network, such FHE-based protocols likely produce faster contact discovery results, which leads to higher user satisfaction. Thus, we recommend the following hybrid use of contact discovery protocols: Directly after installation of a mobile messaging application, a FHE-based protocol (e.g., [18]) is used to perform the initial contact discovery. Then, while the phone is charging overnight and is connected to a WiFi network, the base and setup phase of one of our protocols is performed. This leads to very efficient online phases for future protocol runs, which are performed regularly when updates on client or server side happen (potentially over mobile data plans where communication matters). See also §6.2 for a more detailed comparison between FHE-based unbalanced PSI protocols and our work.

Dedicated Server for Cuckoo Filter Membership Tests. In many scenarios, a large number of clients is part of a single organization. For example, consider the mobile malware detection scenario discussed in [40], where all applications installed on a client's smartphone are checked against a database of malicious applications. When employing such a malware detection service in an enterprise context, a company usually buys a volume license for all of its employees.

To reduce the overall data communication, the company could host a dedicated server which would receive the large encrypted database of server items represented as a compressed Cuckoo filter once. If a client then wants to compute the intersection between installed and malicious applications, it only communicates with the malware detection service provider to

perform OPRF evaluations and then hands off the encrypted items to the trusted company server, which performs the set intersection on behalf of the clients and reports back the result. Since this trusted server does not have knowledge of the PRF key, it cannot directly deduce which items the client holds.

However, since the OPRF result is deterministic when using the same secret key, the trusted server can learn when multiple clients request the same item. Furthermore, it could interact with the malware detection service provider itself to obtain encryptions of known items, which it can compare to the encrypted items of the clients. However, this kind of leakage can be argued to be acceptable in many settings, such as the company-internal setting mentioned above.

Partitioning the Database. A simple solution to reduce the required communication during the setup phase is to partition the server database s.t. clients only download Cuckoo filters relevant for the contacts in their address book (for example w.r.t. number prefixes, states, countries, or regions).

Assuming that the majority of users has contacts in only very few such partitions, this approach leads to practical data transmission sizes even for services with billions of users. In the worst case (i.e., a user has contacts in all partitions or prefers to leak no information at all), multiple runs of our protocols can cover the worldwide user base.

However, this solution presents a significant performance / privacy trade-off since clients leak information about their social graph. For example, intelligence agencies might find it suspicious if US citizens evidently have contacts in middle eastern countries. How severe the privacy of users is threatened also depends on how fine-grained the chosen partitions are: if they are too small, it might even be possible to identify an individual just by observing Cuckoo filter downloads.

B ARM Cryptography Extensions (CE)

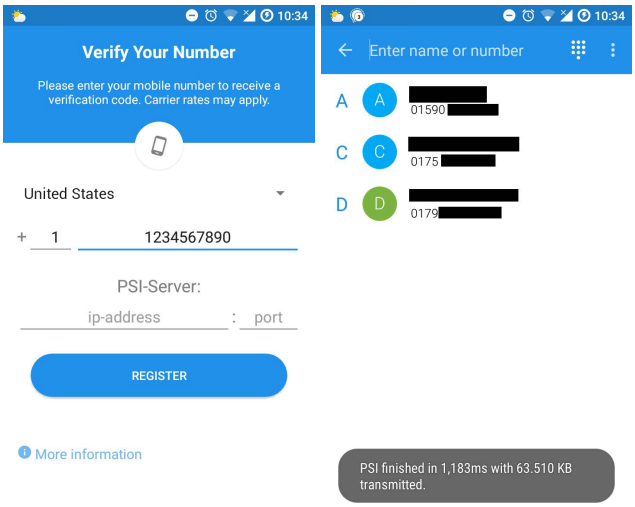
The wide availability of the ARM Cryptography Extensions (CE) in modern smartphone processors is highlighted in Tab. 7.

System-on-a-Chip (SoC)	Example Smartphones and Tablets	CE
Apple A4, A5, A6	iPhone 4, iPad, iPad 2, iPhone 5	✗
Apple A7, A8, A9	iPhone (5s,6), iPad Air, iPad mini 2	✓
Apple A10, A11, A12	iPhone (7,8,X,Xs), iPad (2018), iPad Pro	✓
Snapdragon 801	HTC One (E8), OnePlus One	✗
Snapdragon 805	Galaxy S5+, Nexus 6	✗
Snapdragon 808	Nexus 5X, LG G4, Moto X Style	✓
Snapdragon 810	OnePlus 2, Nexus 6P, Sony Xperia Z5	✓
Snapdragon 820	OnePlus 3, Galaxy S7, LG G5	✓
Snapdragon 821	Google Pixel (XL), LG G6	✓
Snapdragon 835	Google Pixel 2 (XL), Galaxy S8	✓
Snapdragon 845	OnePlus 6, Galaxy S9, Sony Xperia Z2	✓

Table 7: Availability of ARM Cryptography Extensions (CE) in modern smartphone and tablet systems-on-a-chip (SoCs).

C Signal Integration Demonstrator

As a proof-of-concept, we modified the client application of the open-source messenger Signal to perform contact discovery using our PSI protocols. To be able to run the modified client with the official servers, the integration works as follows: Whenever Signal triggers the contact discovery routine, we run one of the PSI protocols with our own PSI server[10]. The resulting matches are then used as input for the unmodified Signal contact discovery routine. This way, the official Signal server only learns the hashes of phone numbers which are already registered to the service. Our changes to the user interface of the Android version of the Signal application are depicted in Fig. 3.



(a) Signal registration. (b) Contact discovery result.

Figure 3: Screenshots of our prototype integration into the open-source messenger Signal.

D Comparison of Unbalanced PSI Protocols on the x86 Architecture

The goal of our paper is to provide efficient private contact discovery for mobile messaging applications via improved unbalanced PSI protocols with implementations optimized for smartphones. Therefore, we focus our implementation and evaluation efforts on the mobile use case and perform our experiments on real smartphones with ARMv8 architecture. However, to present the complete picture, we give a comparison to protocols for unbalanced PSI running on x86 hardware and communicating in a local network in Tab. 8.

[10] In practice, this PSI server would be run by Signal and use the actual database of Signal users.

Parameters N_s	Parameters N_c	Protocol	Online Time [s]	Online Communication [MiB]	Setup Communication / Client Storage [MiB]	Server Setup [s]
2^{28}	1,024	[59]	∗0.16	0.07	806	∗182
		[18]	∗12.10	18.57	0	∗4,628
		LowMC-GC-PSI (Ours)	0.93	24.01	1,072	1,869
		ECC-NR-PSI (Ours)	1.34	6.06	1,072	52,332
2^{24}	11,041	[59]	0.71	0.67	48	342
		[19]	44.70	23.20	0	71
		[18]	20.10	41.48	0	656
		LowMC-GC-PSI (Ours)	12.51	258.79	67	117
		ECC-NR-PSI (Ours)	11.94	65.24	67	3,298
	5,535	[59]	0.35	0.34	48	342
		[19]	40.10	20.10	0	64
		[18]	22.01	16.39	0	806
		LowMC-GC-PSI (Ours)	5.63	129.73	67	117
		ECC-NR-PSI (Ours)	5.93	32.71	67	3,298
2^{20}	11,041	[59]	0.71	0.67	3	22
		[19]	6.40	11.50	0	6.4
		[18]	4.49	14.34	0	43
		LowMC-GC-PSI (Ours)	12.51	258.79	4.2	7.3
		ECC-NR-PSI (Ours)	11.94	65.24	4.2	242
	5,535	[59]	0.35	0.34	3	22
		[19]	4.30	5.60	0	4.3
		[18]	4.23	11.50	0	43
		LowMC-GC-PSI (Ours)	5.63	129.73	4.2	7.3
		ECC-NR-PSI (Ours)	5.93	32.71	4.2	242

Table 8: Comparison of unbalanced PSI protocols in the LAN setting (10 Gbit/s, 0.02 ms RTT) on PC hardware (x86 architecture). Numbers for other protocols are taken from [18]. All numbers are from single-core executions, except those marked with ∗, which was an execution with 32 cores on the server side and 4 cores on the client side. The bit length α of all items is 128, except for [19], where $\alpha = 32$ due to limitations of the protocol.