# Contact Discovery in Mobile Messengers: Low-cost Attacks, Quantitative Analyses, and Efficient Mitigations

CHRISTOPH HAGEN, University of Würzburg, Germany
CHRISTIAN WEINERT, Royal Holloway, University of London, United Kingdom
CHRISTOPH SENDNER, University of Würzburg, Germany
ALEXANDRA DMITRIENKO, University of Würzburg, Germany
THOMAS SCHNEIDER, Technical University of Darmstadt, Germany

Contact discovery allows users of mobile messengers to conveniently connect with people in their address book. In this work, we demonstrate that severe privacy issues exist in currently deployed contact discovery methods and propose suitable mitigations.

Our study of three popular messengers (WhatsApp, Signal, and Telegram) shows that large-scale crawling attacks are (still) possible. Using an accurate database of mobile phone number prefixes and very few resources, we queried 10 % of US mobile phone numbers for WhatsApp and 100 % for Signal. For Telegram we find that its API exposes a wide range of sensitive information, even about numbers not registered with the service. We present interesting (cross-messenger) usage statistics, which also reveal that very few users change the default privacy settings.

Furthermore, we demonstrate that currently deployed hashing-based contact discovery protocols are severely broken by comparing three methods for efficient hash reversal. Most notably, we show that with the password cracking tool "JTR" we can iterate through the entire world-wide mobile phone number space in $< 150$ s on a consumer-grade GPU. We also propose a significantly improved rainbow table construction for non-uniformly distributed input domains that is of independent interest.

Regarding mitigations, we most notably propose two novel rate-limiting schemes: our *incremental* contact discovery for services without server-side contact storage strictly improves over Signal's current approach while being compatible with private set intersection, whereas our *differential* scheme allows even stricter rate limits at the overhead for service providers to store a small constant-size state that does not reveal any contact information.

CCS Concepts: • **Security and privacy** → *Privacy-preserving protocols*; *Cryptanalysis and other attacks*.

Additional Key Words and Phrases: mobile contact discovery, hash reversal, rainbow table, crawling, PSI

**ACM Reference Format:**
Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. 2022. Contact Discovery in Mobile Messengers: Low-cost Attacks, Quantitative Analyses, and Efficient Mitigations. *ACM Trans. Priv. Sec.* 0, 0, Article 0 (May 2022), 40 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Contact discovery is a procedure run by mobile messaging applications to determine which of the contacts in the user's address book are registered with the messaging service. Newly registered users can thus conveniently and instantly start messaging existing contacts based on their phone

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
2471-2566/2022/5-ART0 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

number without the need to exchange additional information like user names, email addresses, or other identifiers[1].

Centralized messaging platforms can generally learn the social graphs of their users by observing messages exchanged between them. Current approaches to protect against this type of traffic analysis are inefficient [101], with Signal attempting to improve their service in that regard [60]. While only active users are exposed to such analyses, the contact discovery process potentially reveals *all* contacts of users to the service provider, since they must in some way be matched with the server's database. This is one of the reasons why messengers like WhatsApp might not be compliant with the European GDPR in a business context [29, 97].

Cryptographic protocols for private set intersection (PSI) can perform this matching securely. Unfortunately, they are currently not efficient enough for mobile applications with billions of users [50]. Furthermore, even when deploying PSI protocols, this does not resolve all privacy issues related to contact discovery as they cannot prevent enumeration attacks, where an attacker attempts to discover which phone numbers are registered with the service.

**Leaking Social Graphs.** Worryingly, recent work [50] has shown that 5 out of 12 surveyed mobile messengers (including WhatsApp and Telegram) facilitate contact discovery by simply uploading *all* contacts from the user's address book[2] to the service provider and even store them on the server if no match is found [2]. The server can then notify the user about newly registered users, but can also construct the full social graph of each user. These graphs can be enriched with additional information linked to the phone numbers from other sources [15, 38, 39]. The main privacy issue here is that sensitive contact relationships can become known and could be used to scam, discriminate, or blackmail users, harm their reputation, or make them the target of an investigation. The server could also be compromised, resulting in the exposure of such sensitive information even if the provider is honest.

To alleviate these concerns, some mobile messaging applications (7 out of 12 messengers surveyed in [50], including Signal) implement a hashing-based contact discovery protocol, where phone numbers are transmitted to the server in hashed form. However, the low entropy of phone numbers indicates that it is most likely feasible for service providers to reverse the received hash values [65] and therefore, albeit all good intentions, there is no gain in privacy.

In addition to [50], we provide an updated survey of messengers regarding their contact discovery implementation based on an analysis of their privacy policies in Tab. 1. It turns out that even in 2022, the majority of surveyed services relies on hashing to process phone numbers "in a manner in which [they] cannot read the data" [24].

**Crawling.** Unfortunately, curious or compromised service providers are not the only threat. Malicious users or external parties might also be interested in extracting information about others. Since there are usually no noteworthy restrictions for signing up with such services, any third party can create a large number of user accounts to crawl this database for information by requesting data for (randomly) chosen phone numbers. Such enumeration attacks cannot be fully prevented, since legitimate users must be able to query the database for contacts. In practice, rate limiting is a well-established measure to effectively mitigate such attacks at a large scale, and one would assume that service providers apply reasonable limits to protect their platforms. As we show in § 4, this is not the case.

[1]Some mobile apps of social networks perform contact discovery also using email addresses stored in the address book.
[2]Assuming that users give the app permission to access contacts, which is very likely since otherwise they must manually enter their messenger contacts.

Table 1. Survey of messengers' contact discovery implementation based on an analysis of their privacy policies (updated version of [50]). Almost all messengers upload contact information either in the clear or in hashed form. Signal now runs contact discovery exclusively inside a trusted execution environment (Intel SGX) [64].

| Messenger | Plaintext | Hashed | TEE (Intel SGX) |
|---|---|---|---|
| Confide | ✗ | ✓ | ✗ |
| Dust | ✓ | ✗ | ✗ |
| ginlo | ✗ | ✓ | ✗ |
| KakaoTalk | ✓ | ✗ | ✗ |
| Signal[a] | ✗ | (✓) | ✓ |
| Telegram | ✓ | ✗ | ✗ |
| Threema | ✗ | ✓ | ✗ |
| Viber | ✓ | ✗ | ✗ |
| WhatsApp[b] | ✓ | ✓ | ✗ |
| Wickr Me | ✗ | ✓ | ✗ |
| Wire | ✗ | ✓ | ✗ |

[a]Signal removed the hashing-based contact discovery implementation from the Android app source code in September 2020.
[b]WhatsApp servers store phone numbers of contacts that are not registered in hashed form, however, initially receive all contacts in plain [102].

The simple information whether a specific phone number is registered with a certain messenger can be sensitive in many ways, especially when it can be linked to a person. For example, in countries where some services are banned [107], disobeying citizens could be identified and persecuted.

Comprehensive databases of phone numbers registered with a particular service can also allow attackers to perform exploitation at a larger scale. Since registering a phone number usually implies that the phone is active, such databases can be used as a reliable basis for automated sales or phishing calls. Such "robocalls" are already a massive problem in the US [51, 100] and recent studies show that telephone scams are unexpectedly successful [98]. Two prominent WhatsApp vulnerabilities, where spyware could be injected via voice calls [92] or where remote code execution was possible through specially crafted MP4 files [30], could have been used together with such a database to quickly compromise a significant number of mobile devices.

Which information can be collected with enumeration attacks depends on the service provider and the privacy settings available to and selected by the user. Examples for personal (meta) data that can commonly be extracted from a user's account include profile picture(s), nickname, status message, and the last time the user was online. In order to obtain such information, one can simply discover specific numbers or randomly search for users [88]. By tracking such data over time, it is possible to build accurate behavior models [9, 90, 110]. Matching such information with other social networks and publicly available data sources allows third parties to build even more detailed profiles [15, 38, 39]. From a commercial perspective, such knowledge can be utilized for targeted advertisement or scams; from a personal perspective for discrimination, blackmailing, or planning a crime; and from a nation state perspective to closely monitor or persecute citizens [18]. A feature of Telegram, the possibility to determine phone numbers associated with nicknames appearing in group chats, lead to the identification of "Comrade Major" [108] and potentially endangered many Hong Kong protesters [18].

**Our Contributions.** With a comprehensive study, we illustrate severe privacy issues that exist in currently deployed contact discovery methods by performing practical attacks both from the perspective of a curious service provider as well as malicious users. Furthermore, we propose

and evaluate two new contact discovery schemes that can be used in combination with a host of compatible mitigation techniques to prevent our attacks.

*Hash Reversal Attacks.* Curious service providers can exploit currently deployed hashing-based contact discovery methods, which are known to be vulnerable [28, 63, 65] yet are widely used (cf. [50] and Tab. 1), also in other types of applications, e.g., Apple's AirDrop protocol [43]. We quantify the practical efforts for service providers (or attackers who gain access to the server) for efficiently reversing hash values received from users by evaluating three approaches: (i) generating large-scale key-value stores of phone numbers and corresponding hash values for instantaneous dictionary lookups, (ii) hybrid brute-force attacks based on hashcat [93] and "John The Ripper" (JTR) [74], and (iii) a novel rainbow table construction. This involves bypassing scale limitations of existing key-value stores, reviewing source code of brute-force implementations to understand and avoid their performance bottlenecks, and implementing our own software when none exists, e.g., we implement our rainbow table construction in our own tool "RainbowPhones" for which already other use cases appeared [42].

We start off by compiling an accurate database of world-wide mobile phone prefixes (cf. § 2) that significantly narrows the search space compared to coarse-grained estimations of prior work [65] (from 811 trillion to 118 billion). In § 3, we then demonstrate that such hashes can be reversed in just 0.14 ms (amortized per hash) using a lookup database, or less than 0.5 ms when brute-forcing. Our rainbow table construction incorporates the non-uniform structure of all possible phone numbers in a specialized reduction function that produces a uniform output distribution via additional chain offsets and is of independent interest. We show that one can achieve a hit rate of over 99.99 % with an amortized lookup time of less than 50 ms while only requiring 30.4 GB storage space, which improves over classical rainbow tables by more than factor 9,400x in storage.

*Crawling Attacks.* For malicious registered users and outside attackers, we demonstrate that crawling the global databases of the following services is feasible: WhatsApp (the most popular mobile messenger in the world [35]), Signal (which is highly endorsed as a trustworthy, secure, and privacy-preserving messenger [81, 91]), and Telegram (which claims to be more secure than WhatsApp [94] yet often is debated [99] and turns out to be vulnerable [3]). Within a few weeks time, we were able to query 10 % of all US mobile phone numbers for WhatsApp and 100 % for Signal, which is at significantly larger scale than prior works [68, 88] that crawled only 10 million numbers for WhatsApp.

Our attacks require very few resources: the free Hushed [1] application for registering clients with new phone numbers, a VPN subscription for rotating IP addresses, and a single laptop, which we use to generate aggregate statistics coming from multiple Android emulators (where we automate interaction with closed-source WhatsApp instances) as well as our self-developed clients for Signal and Telegram based on their official API and SDK, respectively). We report the rate limits and countermeasures experienced during the process, as well as other interesting findings and statistics. We also find that Telegram's API reveals sensitive personal (meta) data, most notably how many users include non-registered numbers in their contacts.

*Mitigations.* We propose two novel contact discovery schemes that do not require server-side storage of client contacts, which is an important property for privacy protection but inherently challenging to achieve since the server this way cannot easily determine if requests change significantly with each invocation. The first scheme, *incremental* contact discovery (cf. § 5), is based on the observation that the database of registered users changes slowly and thus handles update requests for existing client contacts differently from discovery requests for new address book entries. Our evaluation shows that our approach enables deploying much stricter rate limits without degrading usability or privacy, i.e., legitimate requests are not rejected while even the most powerful adversaries cannot exploit our construction to learn more information about the server database

than with currently deployed schemes. In particular, the currently deployed rate limiting by Signal can be improved by a factor of 31.6x at the cost of negligible overhead (assuming the database of registered users changes 0.1 % per day).

The second scheme, *differential* contact discovery (cf. § 6), stores a small commitment of the client on the server, which is statistically hiding (thus exposing no contact information) and enables the server to determine the number of changed contacts for each request. This information can then be used to very effectively rate-limit the client while there are again no usability restrictions on the user side and curious service providers can learn no additional information compared with currently deployed schemes. Furthermore, we provide a comprehensive discussion on compatible mitigation techniques against both hash reversal and enumeration attacks in § 7, ranging from database partitioning and selective contact permissions to limiting contact discovery to mutual contacts.

Overall, our work provides a comprehensive study of privacy issues in mobile contact discovery and the methods deployed by three popular applications with billions of users. We investigate three attack strategies for hash reversal, explore enumeration attacks at a much larger scale than previous works [39, 88], and discuss a wide range of mitigation strategies, including our novel incremental and differential contact discovery that have the potential of real-world impact through deployment by Signal.

**Additional Contributions to Conference Publication.** With this article, we present a significantly extended version of our conference publication [40]. Concretely, we add the following significant contributions:

*New Hash Reversal Attacks.* In § 3, we implement additional hybrid mode brute-force attacks using hashcat [93] and John the Ripper (JTR) [74]. These attacks allow to sweep the entire world-wide mobile phone number space in under 150 s (our best previous brute-force attack using hashcat

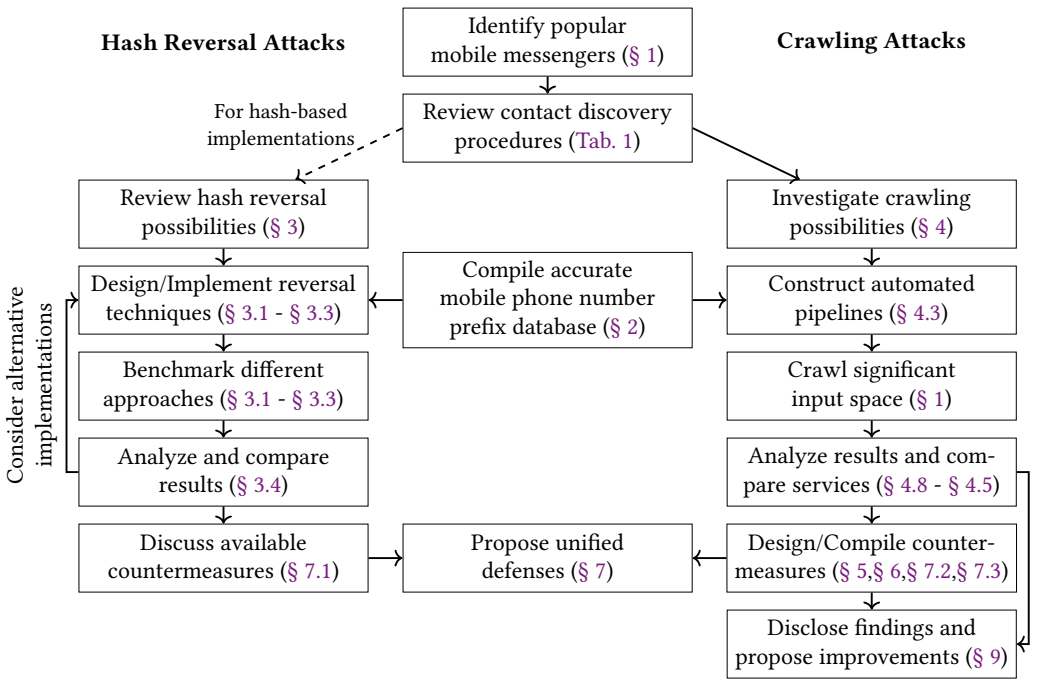



Fig. 1. Overview of our approach, the presented contributions, and the organization of our paper.

required more than 15 h for a full sweep [40]). Furthermore, we report performance results for the hash function SHA-256 in addition to SHA-1 (which was used by Signal for contact discovery), and implement two more hash reversal approaches based on on-disk databases as a low-budget alternative to in-memory key-value stores.

*New Contact Discovery Scheme.* In § 6, we propose a second new contact discovery scheme in addition to incremental contact discovery (cf. § 5). Our scheme called "differential contact discovery" allows the server to limit the number of changes in the client's inputs without storing them. We evaluate our scheme quantitatively, thereby showing that it is even superior to incremental contact discovery in terms of rate limiting with the low additional overhead of storing small constant-sized state information on both client and server side.

**Outline.** We first describe our approach to compile an accurate database of mobile phone numbers (§ 2), which we use to demonstrate efficient reversal of phone number hashes (§ 3). We also use this information to crawl WhatsApp, Signal, and Telegram, and present insights and statistics (§ 4). Regarding mitigations, we present our incremental and differential contact discovery schemes (§ 5 and § 6, respectively), and discuss further techniques (§ 7). We then provide an overview of related work (§ 8) and conclude with a report on our responsible disclosure process as well as ethical considerations and an update for 2022 (§ 9). We provide an overview of our approach, the presented contributions, and the organization of our paper in Fig. 1.

## 2 MOBILE PHONE NUMBER PREFIX DATABASE

In the following sections, we demonstrate privacy issues in currently deployed contact discovery methods by showing how alarmingly fast hashes of mobile phone numbers can be reversed (cf. § 3) and that the database crawling of popular mobile messaging services is feasible (cf. § 4). Both attacks can be performed more efficiently with an accurate database of all possible mobile phone number prefixes[3]. Hence, we first show how such a database can be built.

### 2.1 Phone Number Structure

International phone numbers conform to a specific structure to be globally unique: Each number starts with a country code (defined by the ITU-T standards E.123 and E.164, e.g., +1 for the US), followed by a country-specific prefix and a subscriber number. Valid prefixes for a country are usually determined by a government body and assigned to one or more telecommunication companies. These prefixes have blocks of subscriber numbers assigned to them, from which numbers can be chosen by the provider to be handed out to customers. The length of the subscriber numbers is specific for each prefix and can be fixed or in a specified range.

In the following, we describe how an accurate list of (mobile) phone number prefixes can be compiled, including the possible length of the subscriber number. A numbering plan database is maintained by the *International Telecommunication Union* (ITU) [49] and further national numbering plans are linked therein. This database comprises more than 250 countries (including autonomous cities, city states, oversea territories, and remote island groups) and more than 9,000 providers in total. In our experiments in § 4, we focus on the US, where there are 3,794 providers (including local branches). Considering the specified minimum and maximum length of phone numbers, the prefix database allows for ≈52 trillion possible phone numbers (≈1.6 billion in the US). However, when limiting the selection to mobile numbers only, the search space is reduced to ≈758 billion (≈0.5 billion in the US).

[3]Some messengers like WhatsApp and Signal also allow to register with landline phone numbers. We assume that very few users make use of this option, and also argue that gathering landline phone numbers is less attractive for attackers (e.g., when the goal is to infect smartphones with malware).

Table 2. Comparison of selected countries with regard to their amount and density of registrable mobile phone numbers.

| Country | # Numbers (in million) | # Numbers / Population |
|---|---|---|
| Cuba | 0.3 | 0.03 |
| Moldova | 6.3 | 1.9 |
| Australia | 48.6 | 1.9 |
| Canada | 76.0 | 2.0 |
| Japan | 127.9 | 1.0 |
| Russia | 327.0 | 2.2 |
| United States | 505.7 | 1.5 |
| Germany | 538.3 | 6.5 |
| China | 4,496.0 | 3.2 |
| Austria | 93,658.7 | 10,573.4 |

### 2.2 Database Preprocessing

As it turned out in our experiments, some of the numbers that are supposed to be valid according to the ITU still cannot be registered with the examined messaging applications. Therefore, we perform two additional preprocessing steps.

Google's `libphonenumber` library [36] can validate phone numbers against a rule-based representation of international numbering plans and is commonly used in Android applications to filter user inputs. By filtering out invalid numbers, the amount of possible mobile phone numbers can be reduced to ≈353 billion.

Furthermore, WhatsApp performs an online validation of the numbers before registration to check, for example, whether the respective number was banned before. This allows us to check all remaining prefixes against the WhatsApp registration/login API by requesting the registration of one number for each prefix and each possible length of the subscriber number. Several more prefixes are rejected by WhatsApp for reasons like "too long" or "too short". Our final database for our further experiments thus contains up to ≈118 billion mobile phone numbers (≈0.5 billion in the US[4]).

### 2.3 Differences in the Number Spaces

Interestingly, the amount of registrable mobile phone numbers greatly differs between countries and is not necessarily proportional to the country's population. In Tab. 2 we list a selection of countries with their amount of registrable mobile phone numbers (i.e., filtered by `libphonenumber` and our WhatsApp registration API check) and set it in relation to the population. The ratio between the number space and the population indicates whether the amount of resources spent enumerating the entire number space can yield a satisfactory amount of matches. For example, while the US and Germany have roughly the same amount of registrable mobile phone numbers, one would expect to find many more active phone numbers in the US due to the much larger population.

Our results show that small as well as less developed countries often have a limited number space and therefore can be crawled with very little effort. On the other hand, we observe some outliers: Austria, for example, has such a large number space that for every citizen there are more than 10,000 registrable mobile phone numbers available. While such a ratio seems to make crawling infeasible, one can still exploit that the phone numbers are typically not uniformly distributed, but given away in blocks. Hence, one could follow the strategy to first randomly check a few numbers

[4] `libphonenumber` and WhatsApp reject no US mobile prefixes.

for each possible prefix and then focus on the most fruitful prefixes to still cover a good portion of the population.

## 3 MOBILE PHONE NUMBER HASH REVERSAL

Although the possibility of reversing phone number hashes has been acknowledged before [28, 63, 65], the severity of the problem has not been quantified. The amount of possible mobile phone numbers that we calculated in § 2 indicates the feasibility of determining numbers based on their hash values. In the following, we show that *real-time* hash reversal is practical not only for service providers and adversaries with powerful resources, but even at a large scale using commodity hardware only. These results furthermore indicate that users' social graphs are widely at risk as a whole range of mobile messengers (7 out of 12 messengers surveyed in [50]) use hashing-based mobile contact discovery protocols.

**Threat Model.** Here we consider the scenario where users provide hashed mobile phone numbers of their address book entries to the service provider of a mobile messaging application during contact discovery. The adversary's goal is to learn the numbers from their hashed representation. For this, we assume the adversary has full access to the hashes received by the service provider. The adversary therefore might be the service provider itself (being "curious"), an insider (e.g., an administrator of the service provider), a third party who compromised the service provider's infrastructure, or a law enforcement or intelligence agency who forces the service provider to hand out data. Importantly, we assume the adversary has no control over users and does not tamper with the contact discovery protocol.

**Data Representation and Hash Primitive.** In all the experiments described in this section, we represent phone numbers as strings without spaces or dashes, and including their country code. We choose to represent phone numbers using digits only and not to include the "+"-sign (required by the E.164 format), since it yields space savings but does not affect runtime. We choose SHA-1 as our exemplary hash function, which was also used by Signal for contact discovery[5], and additionally investigate performance using the more recent and commonly used hash function SHA-256.

**Approaches.** We compare three different approaches to reverse hashes of mobile phone numbers: hash database (§ 3.1), brute force (§ 3.2) and rainbow tables (§ 3.3) – each suitable for different purposes and available resources.

**Hardware.** We run experiments on one node of a high-performance cluster with 30 virtual cores of two Intel Xeon Gold 6144 CPUs, 700 GB RAM and 13 TB NVMe-based storage. For brute-force approaches where GPU acceleration is possible, we use a system with an 8-core AMD 3700X CPU, 32 GB RAM and an NVIDIA RTX 2070 SUPER GPU.

### 3.1 Hash Database

The first approach utilizes a database that stores a mapping between phone numbers and their hash values. Once generated, such a database enables an attacker to perform constant-time lookups for each given hash value. To evaluate the performance of this method, we create an in-memory database based on the possible mobile phone numbers from § 2.2 (i.e., mobile phone numbers allowed by Google's `libphonenumber` and the WhatsApp registration API) paired with their SHA-1/SHA-256 hashes. We instantiate this approach using two in-memory key-value databases, Redis [87] and LMDB [16], for comparison. While such in-memory key-value databases represent the very best technology that is currently available, we additionally implement and evaluate two on-disk approaches: a naive key-value file and a traditional MySQL database.

[5]Before switching entirely to TEE-based contact discovery, Signal truncated the SHA-1 output to 10 B to reduce communication overhead while still producing unique hashes for all possible phone numbers.

*3.1.1 Redis.* Redis is an in-memory key-value database based on hash tables. Hash tables allow Redis near-instant lookup times while residing in RAM. One of the significant constraints of Redis is its volatile nature, where all data is lost between reboots. In contrast, LMDB's data resides on disk and can generally be persisted across reboots. In a production setting, Redis usually writes back all issued commands to a disk for persistence, although these settings were disabled for our setup to achieve maximum performance.

**Benchmarks.** We choose a Redis database due to its robustness, in-memory design, and near constant lookup-time [86]. Since one Redis instance cannot handle the required number of entries, we construct a cluster of 120 instances on our node. In contrast, we did not experience any insertion limitations with a single LMDB instance. Populating the table requires ≈1.3 h for both SHA-1 and SHA-256 in our experiments due to several bottlenecks, e.g., the interface to the Redis cluster can only be accessed through a network interface. With our setup, only 4 billion hashes (roughly 3.4 % of the considered number space) can fit into the RAM with only a minimal difference between SHA-1 and SHA-256. We perform batched lookups of 10,000 hashes, which on average take 1.9 s, resulting in an amortized lookup time of 0.19 ms.

*3.1.2 LMDB.* LMDB is an in-memory key-value database that utilizes memory mapping to link physical memory to the address space of the process, and utilizes an efficiently searchable B+ tree structure [23] to store data. In contrast to Redis, LMDB is only available as a library and not as a full service with an API. Hence, we implement the initialization and search functionality in Python and compile the code with LMDB's library version 0.9.29.

**Benchmarks.** We insert 4 billion pairs into 400 database files (the same amount that our Redis cluster can handle due to RAM constraints), which takes 15.5 min and 18.3 min for SHA-1 and SHA-256, respectively. Then, we look up 10,000 hashes, which takes 1.4 s on average. Thus, LMDB has an amortized lookup time of 0.14 ms and slightly outperforms Redis. LMDB also outperforms Redis with regard to space consumption as Redis already has a considerable initial overhead due to the cluster setting. The overhead amortizes when inserting around ten million keys; nevertheless our LMDB database is still only half the size of Redis with 268 GB and 354 GB for SHA-1 and SHA-256, respectively.

*3.1.3 Summary and On-Disk Approaches.* Both database engines are viable options for hash reversal databases. However, LMDB offers a smaller memory footprint and faster lookup times compared to Redis. If we would consider all 118 billion mobile phone numbers, the resulting database would be around 8 TB and 10.4 TB for LMDB (for SHA-1 and SHA-256, respectively) and 18 TB for Redis (which truncates keys to the same length). Requirements of this scale can be deemed feasible for nation state actors or globally operating service providers with moderate financial resources.

For attackers with consumer hardware, it is also feasible to store a full database on disk, which requires only 3.3 TB of storage for SHA-1 and 4.7 TB for SHA-256[6]. Compared to LMDB and Redis however, which additionally maintain in-memory hash tables for performance enhancements, this simple on-disk approach results in significantly higher lookup times, also due to disk access latencies. Concretely, looking up a batch of 10k numbers with our implementation in Rust by sequentially iterating through unsorted files requires 50 min and 71 min for SHA-1 and SHA-256, respectively.

Another low-budget on-disk alternative is a traditional MySQL database. In our experiments, it takes 13.7 h and 17 h to generate SHA-1 and SHA-256 MySQL databases, respectively. The lookup of 10,000 hashes using only 32 GB of RAM takes 24 s for both hash algorithms, performing by a magnitude worse than Redis and LMDB. The size of the databases is 472 GB and 633 GB for SHA-1 and SHA-256, respectively, which is similar to the other approaches.

[6]Assuming SHA-1/SHA-256 hashes of 20/32 bytes and 64-bit encoded phone numbers.

### 3.2 Brute-Force

Another possibility to reverse phone number hashes is to iteratively hash every element in the input domain until a matching hash is found. Popular choices for this task are the open-source tools hashcat [93] and "John the Ripper" (JTR) [74], which are often used to brute-force password hashes. Both tools offer multiple attack modes where an attacker can choose between so-called masked attacks (i.e., what is commonly known as brute-force attacks), dictionary attacks, and combinations of both, i.e., hybrid attacks like mutations over dictionaries. Advanced settings include elaborate plans such as Markov-Chains to possibly optimize the mutation based on predictions.

In the following, we describe evaluation results using both tools, hashcat and JTR, considering different attack modes. Based on our evaluation using hashcat, we identified hybrid attacks as superior to masked mode, and subsequently limit our evaluation of JTR to the hybrid mode.

*3.2.1 Hashcat.* Hashcat is built to efficiently parallelize the brute-force process and additionally allows to utilize GPUs to maximize performance. Both brute-forcing modes (*masked* and *hybrid*) allow the specification of *masks* that constrain the inputs according to a given structure. Specifically, masks in this context refer to the variable part of a phone number, whereas the country code and the provider-specific beginning of phone numbers is used as a fixed prefix for mutations. The length of a mask (where mutations begin) is consequently variable (since not all phone numbers have the same length) and entails different hash rates.

The conceptual difference between *masked* and *hybrid* mode is minimal and a performance difference was not expected prior to empirically evaluating both modes. In *masked* mode, all prefixes and their corresponding masks are written into one file (i.e., *hcmask* file), which hashcat uses to iterate over the entire mobile phone number space in a single execution. In contrast, *hybrid* mode allows the grouping of prefixes by their mask-lengths into ten files. Hashcat and JTR are therefore executed ten times while applying the mask length and referencing the corresponding prefix files to iterate over the entire mobile phone number space. After reviewing hashcat's source code, we discovered that in *masked* mode, every line of the file is executed separately, whereas in *hybrid* mode, hashcat can apply the same mask mutations to multiple prefixes simultaneously on the GPU. Both modes are used to model our input space of 118 billion mobile phone numbers (cf. § 2.2).

**Masked mode benchmarks.** Our GPU-based setup has a theoretical SHA-1/SHA-256 hash rate of 6.5/2.9 GHashes/s according to the hashcat benchmark, which would allow us to search the full mobile phone number space in less than 19/40 seconds. However, the true hash rate is significantly lower due to the overhead introduced by hashcat when distributing loads for processing. Since many of the prefixes have short subscriber numbers (e.g., 158,903 prefixes with a length of 4 digits), the overhead of distributing the masks is the bottleneck for the calculations, dropping the hash rate to 0.3 MHashes/s for 3-digit masks (less than 0.01 % efficiency). The hash rate reaches its plateau at around 56 MHashes/s for mask lengths larger than 5 (cf. Fig. 2a), which is still only 0.86/1.93 % of the theoretical hash rate.

In more detail, we describe the observed hashrates depending on the mask length as visualized in Fig. 2a: We observe only 2.1 kHashes/s for mask lengths below four; when masks of length five are added to the prefix, the hashrate rises to around 50 MHashes/s. For SHA-256, we observe nearly identical performance and behavior as for SHA-1. In Fig. 2b we show the time for completing the brute-force task for the different batch sizes 10k, 100k and 1M. The results for the different batch sizes are nearly identical for both hash algorithms – we attribute the constant duration to the limited hashrate. The slightly higher runtimes for SHA-256 are expected due to the algorithmic differences and the longer hash output in comparison to SHA-1. Most notably, a full search over the number space can be completed in 20.00/24.86 hours for batches of 1 million hashes. Fig. 2c shows the amortized lookup times per single hash based on batch size and execution time. As both,

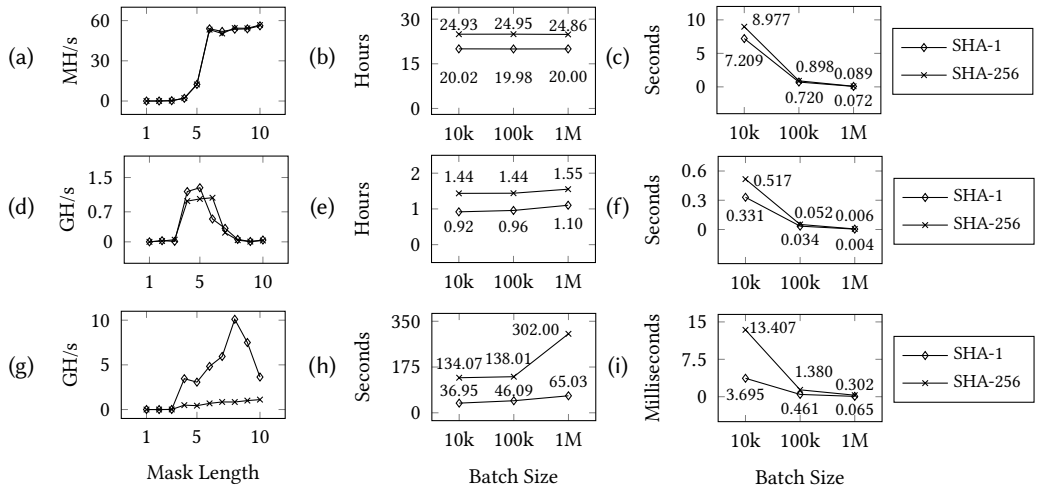



Fig. 2. Brute-force benchmark results for *masked* hashcat (a–c), *hybrid* hashcat (d–f), and *hybrid* JTR (g–i).

the hash rate and execution times of SHA-1 and SHA-256 are very similar, the amortized times are also nearly identical. The amortized lookup rate drops significantly to only 72/89 ms per hash for batches of 1 million hashes. Consequently, the practical results show that theoretical hash rates cannot be reached by simply deploying hashcat and that additional engineering effort would be required to optimize brute-force software for efficient phone number hash reversal.

**Hybrid mode benchmarks.** In hashcat's hybrid mode, we outperform our previous masked attacks tenfold. We observe hash rates of ≈1.3/1.0 GHashes/s for SHA-1/SHA-256 (cf. Fig. 2d). Again, the performance of SHA-1 and SHA-256 is similar. In contrast to *masked mode*, the hashrate drops for larger mask lengths due to under-utilization of the GPU as the higher number of samples created per prefix leads to longer lookup times of the hash internally. Thus, the total time for larger mask lengths is lower than for mask lengths around five. Still, the hash rate of ≈30 MHashes/s is over a magnitude higher than for the *masked mode*. In Fig. 2e, we observe slightly higher runtimes for larger batch sizes due to more comparisons against the input data. The brute-forcing performance is identical per run with around an hour for SHA-1 and 1.5 h for SHA-256. Fig. 2f depicts the amortized lookup times per batchsize, where we see an amortized lookup time of 4/6 ms for a batch size of one million mobile phone numbers hashed with SHA-1/SHA-256. In summary, we show that an attacker using hashcat can traverse the entire mobile phone number space in less than two hours using standard hardware and the initial engineering effort, as described in § 2. Such costs are negligible for nation state actors or service providers.

*3.2.2 JTR.* Additionally, we analyze hybrid-mode attacks with JTR [74], which are implemented by combining word-lists and mask attacks. We split the prefixes into different length-based word-lists according to their variable length. The basic structure is identical in both hybrid and masked mode. Internally, the word-lists are however handled much more efficiently and outperform masked attacks by several orders of magnitude.

The setup is identical to hashcat's hybrid mode, but our benchmarks of JTR's implementation show consistently high hash rates for longer masks (cf. Fig. 2g). The results show a hundredfold decrease in reversal time of phone number hashes compared to hashcat's masked attack. In more detail, we observe a peak hashrate of ≈10 GHashes/s for SHA-1. Our evaluation of SHA-256 shows

an increasing hashrate of ≈1 GHashes/s, indicating a fully utilized GPU for all mask lengths. We see in Fig. 2h a total traversal of the mobile phone number space in under ≈65 s for SHA-1 and under ≈300 s for SHA-256. For SHA-256, we see a near doubling of execution time from batch size 100k to 1M. We attribute this behavior to the increasing number of comparisons between the generated hash digests and input hashes. In fact, the amortized lookup time per hash is under a millisecond for both SHA-1 and SHA-256 as depicted in Fig. 2i.

We show here that an attacker can nearly instantly iterate through the entire mobile phone number space. Using naive hashing to protect mobile phone numbers guarantees no privacy as they are nearly instantly uncovered even with consumer-grade hardware. A nation state actor or service provider can apply this approach to immediately reverse hashes received from users at negligible costs.

### 3.3 Optimized Rainbow Tables

Rainbow tables are an interesting time-memory trade-off to reverse hashes (or any one-way function) from a limited input domain. Based on work from Hellman [44] and Oechslin [70], they consist of precomputed chains of plaintexts from the input domain and their corresponding hashes. These are chained together by a set of reduction functions, which map each hash back to a plaintext. By using this mapping in a deterministic chain, only the start and end of the chain must be stored to be able to search for all plaintexts in the chain. A large number of chains with random start points form a rainbow table, which can be searched by computing the chain for the given hash, and checking if the end point matches one of the entries in the table. If a match is found, then the chain can be computed from the corresponding start index to reveal the original plaintext. The length of the chains determines the time-memory trade-off: shorter chain lengths require more chains to store the same number of plaintexts, while longer chains increase the computation time for lookups. The success rate of lookups is determined by the number of chains, where special care has to be taken to limit the number of duplicate entries in the table by carefully choosing the reduction functions.

Each rainbow table is specific to the hash algorithm being used, as well as the specifications of the input domain, which determines the reduction functions. Conventional rainbow tables work by using a specific alphabet as well as a maximum input length, e.g., 8-digit ASCII numbers. Even though they can be used to work on phone numbers as well, they are extremely inefficient for this purpose: to cover numbers conforming to the E.164 standard (up to 15 digits), the size of the input domain would be $10^{15}$, requiring either huge storage capacity or extremely long chains to achieve acceptable hit rates. Implementations like Cryptohaze [8], where individual alphabets for each character position of the input domain can be specified, might initially appear to be a solution to optimize rainbow tables for (mobile) phone numbers. However, the unique structure of phone numbers limits the usefulness of this technique: the possibilities for each digit in a phone number are strongly dependent on the preceding digits.

We therefore design new reduction functions that always map a hash back into a valid phone number, yielding significant performance improvements. While we use our approach to optimize rainbow tables for phone numbers, our construction can also find application in other areas, e.g., advanced password cracking.

**Specialized Reduction Functions.** Our optimization relies on the specific structure of mobile phone numbers, which consist of a country code, a mobile prefix, and a subscriber number of a specific length (cf. § 2). A trivial reduction function could be defined by taking the first 64 bit $h_{64}$ of hash $h$ and calculating the modulus with the total amount of phone numbers $N$, giving us the index of the phone number in the table of all numbers.

Table 3. Parameters for our reduction function.

| Parameter | Explanation |
|---|---|
| $h_{64}$ | First 64 bit of hash $h$ encoded as an unsigned integer |
| $N$ | Number of possible plaintexts |
| $l_C$ | Length of the chains |
| $i_T$ | Index of the currently generated table |
| $i_C$ | Current index in the chain |

Table 4. Example for selecting the next phone number from a hash value for our improved rainbow table construction.

| Country Code + Prefix | # Subscriber Numbers | Offset |
|---|---|---|
| +1982738 | 10,000 | 0 |
| +172193 | 100,000 | 10,000 |
| +491511 | 10,000,000 | 110,000 |
| +49176 | 10,000,000 | 10,110,000 |

However, the modulo operation introduces non-uniformity in the output of the reduction function: The lower parts of the number space are more likely to be chosen if $N$ is not a divisor of $2^{64}$. We therefore introduce an offset into the calculation that varies for every chain index $i_C$. By choosing the offset as the division of $N$ by the chain length $l_C$, each chunk of the phone number space is more likely for one chain index, producing a uniform distribution overall.

Another issue is collisions in the reduction function: As soon as two different hashes produce the same phone number, then the chains merge and all successive elements of the chain will be duplicate entries. To prevent this, we vary our reduction function with every chain index as well as with every table[7], and define our reduction function as follows:

$$i = \left(h_{64} + \left(\left\lfloor \frac{N}{l_C} \right\rfloor + 65{,}536 \times i_T\right) \times i_C\right) \mod N,$$

where the different parameters are explained in Tab. 3 and the "magic" number 65,536 for the table offset was kept from the original RainbowCrack implementation [48]. This number improves the prevention of persistent hash collisions by shifting the hashes by a large amount for different tables, compared to smaller increments for the chain indices. The number is a power of two for faster multiplication through bit-shifting.

Intuitively, our algorithm concatenates ranges of valid mobile phone numbers into a virtual table, which we can index with a given hash. For each prefix, we store the amount of possible subscriber numbers and the offset of the range within the table. To select a valid number, we calculate the index from the 64-bit prefix of the given hash modulo the table size and perform a binary search for the closest smaller offset to determine the corresponding mobile prefix. Subtracting the offset from the index yields the subscriber number. For example, given Tab. 4 and index 3,849,382, we select the prefix +491511 and calculate the subscriber number as 3,849,382 − 110,000 = 3,739,382, yielding the valid mobile phone number +491511 3739382.

**Implementation.** We implement our optimized rainbow table construction based on the open-source version 1.2[8] of RainbowCrack [48]. To improve table generation and lookup performance,

[7]Rainbow tables are usually split into several files due to their large size.

[8]Newer versions of RainbowCrack that support multi-threading and GPU acceleration exist, but are not open-source [84].

we add multi-threading via OpenMP [72]. Hash operations are performed using OpenSSL [73]. The table generation is modified to receive the number specification as an additional parameter (a file with a list of phone number prefixes and the length of their subscriber numbers). Our open-source implementation called "RainbowPhones" is available at https://contact-discovery.github.io/.

**Benchmarks.** We generate tables for all registrable mobile phone numbers (118 billion numbers, cf. § 2) considering SHA-1 and SHA-256, and determine creation time and size depending on the desired success rate for lookups, as well as lookup rates. Our test system is the same as for database-based attacks, however, only 2 GB of RAM are utilized.

We store 100 million chains of length 1,000 in each file, which results in files of 1.6 GB with a creation time of ≈66 and ≈75 minutes each for SHA-1 and SHA-256, respectively. For a single file, we already achieve a success rate of over 50 % and an amortized lookup time of less than 18 ms and 21 ms for each hash when testing batches of 10,000 items with SHA-1 and SHA-256, respectively. With 19 files (30.4 GB created within 20.6 hours and 23.6 hours), the success rate is more than 99.99 % with an amortized lookup time of 37 ms and 41 ms for SHA-1 and SHA-256, respectively.

In comparison, a conventional rainbow table of all 7 to 15-digit numbers has an input domain more than 9,400x larger than ours, and (with similar success rates and the same chain length) would require approximately 230 TB of storage and a creation time of more than 26 years on our test system (which is a one-time expense). The table size can be reduced by increasing the chain length, but this would result in much slower lookups.

These measurements show that our improved rainbow table construction makes large-scale hash reversal of phone numbers practical even with commodity hardware and limited financial investments. Since the created tables have a size of only a few gigabytes, they can also be easily distributed.

### 3.4 Comparison of Hash Reversal Methods

Our results for the three different approaches are summarized in Tab. 5. Each approach is suitable for different application scenarios, as we discuss in the following.

A full in-memory hash database (cf. § 3.1) is an option only for well-funded adversaries that require real-time reversal. It is superior to the brute-force method and rainbow tables when considering lookup latencies.

Brute-force cracking (cf. § 3.2) is an option for a range of adversaries, from nation state actors to attackers with consumer-grade hardware. Batching allows to significantly improve the amortized lookup rate, making brute-force cracking attractive when a large number of hashes is to be reversed, e.g., when an attacker compromised a database.

Our optimized rainbow tables (cf. § 3.3) are the approach most suited for adversaries with commodity hardware, since these tables can be calculated in reasonable time, require only a few gigabytes of storage, can be easily customized to specific countries or number ranges and types, and can reverse dozens of phone number hashes per second. As such, rainbow tables allow hash reversal of phone numbers on CPUs with only moderate performance. It is also possible to easily share and use precomputed rainbow tables, which is done for conventional rainbow tables as well [83], despite their significantly larger size.

For other hash functions, we expect reversal and generation times to vary by a constant factor, depending on the computation time of the hash function [41] (except for hash databases where lookup times remain constant).

Our results show that hashing phone numbers for privacy reasons does not provide any protection, as it is easily possible to recover the original number from the hash. Thus, we strictly advise against the use of hashing-based protocols in their current form for contact discovery where users are identified by low-entropy identifiers such as phone numbers, short user names, or email addresses. The possibility of immediate hash reversal also has implications on other application areas where

Table 5. Comparison of phone number hash reversal methods for SHA-1 / SHA-256 (previous results from [40] are printed in []).

| Evaluation Criteria | Hash Database § 3.1 | Brute-Force § 3.2 | Rainbow Tables § 3.3 |
|---|---|---|---|
| Generation Time | 37.9 min [13 h] / 66.2 min | – | 20.6 h / 23.6 h |
| Memory Requirement | ≥ 3.3 TB / ≥ 4.7 TB | – | 30.4 GB |
| Lookup Time per 10k Batch | 1.4 s | 36.95 s [15.3 h] / 134.07 s | 369 s / 410 s |
| Best Amortized Time per Hash | 0.14 ms | 0.065 ms [57 ms] / 0.302 ms | 37 ms / 41 ms |
| GPU Acceleration | ✗ | ✓ | (✓) |

hashing of (mobile) phone numbers or email addresses is used for pseudonymization or contact matching, as in Apple's AirDrop protocol [42, 43]. In § 7.1, we discuss multiple ideas how to at least strengthen hashing-based protocols against the presented hash reversal methods.

## 4 USER DATABASE CRAWLING

We study three popular mobile messengers to quantify the threat of enumeration attacks based on our accurate phone number database from § 2.2: WhatsApp, Signal, and Telegram. All three messengers discover contacts based on phone numbers, yet differ in their implementation of the discovery service and the information exposed about registered users. We select these messengers due their popularity, their use of phone numbers as the primary identifier of the users, and their self-stated focus on security and privacy.

**Threat Model.** Here we consider an adversary who is a registered user and can query the contact discovery API of the service provider of a mobile messaging application. For each query containing a list of mobile phone numbers (potentially in hashed form) an adversary can learn which of the provided numbers are registered with the service along with further information about the associated accounts (e.g., profile pictures). The concrete contact discovery implementation is irrelevant and it might even be based on PSI (cf. § 7.1). The adversary's goal is to check as many numbers as possible and also collect all additional information as well as meta data provided for the associated accounts. The adversary may control one or even multiple user accounts, and is restricted to (ab)use the contact discovery API with well-formed queries. This implies that we assume no invasive attacks, e.g., compromising other users or the service provider's infrastructure.

### 4.1 Investigated Messengers

**WhatsApp.** WhatsApp is currently one of the most popular messengers in the world, with 2 billion users [35]. Launched in 2009, it was acquired by Facebook in 2014 for approximately 19.3 billion USD.

**Signal.** The Signal Messenger is an increasingly popular messenger focused on privacy, and is endorsed by many prominent members of the security community [91], as well as the recommended tool for public instant messaging of EU staff [81]. Signal's end-to-end-encryption protocol is also being used by other applications, such as WhatsApp, Facebook, and Skype. There are no recent statistics available regarding Signal's growth and active user base.

**Telegram.** Telegram is a cloud-based messenger that reported 400 million users in April 2020 [33]. Despite its own claims regarding security [94], Telegram is criticized due to its use of the self-developed MTProto encryption protocol, which turns out to be vulnerable against multiple attacks [3], as well as its lack of end-to-end encryption in standard and group chats [99].

### 4.2 Differences in Contact Discovery

Both WhatsApp and Telegram transmit the contacts of users in clear text to their servers (but encrypted during transit), where they are stored to allow the services to push updates (such as newly registered contacts) to the clients. WhatsApp stores phone numbers of its users in clear text on the server, while phone numbers not registered with WhatsApp are MD5-hashed with the country prefix prepended (according to court documents from 2014 [2]). Signal does not store contacts on the server. Instead, each client periodically sends hashes of the phone numbers stored in the address book to the service, which matches them against the list of registered users and responds with the intersection.

The different procedures illustrate a trade-off between usability and privacy: the approach of WhatsApp and Telegram can provide faster updates with less communication overhead, but needs to store sensitive data on the servers.

### 4.3 Test Setups

We evaluate the resistance of these three messengers against large-scale enumeration attacks with different setups.

**WhatsApp.** Because WhatsApp is closed-source, we run the official Android application in an emulator, and use the Android UI Automator framework to control the user interface. First, we insert 60,000 new phone numbers into the address book of the device, then start the client to initiate the contact discovery. After synchronization, we can automatically extract profile information about the registered users by stepping through the contact list.

New accounts are registered manually following the standard sign-up procedure with phone numbers obtained from the free Hushed [1] application. Interestingly, if the number provided by Hushed was previously registered by another user, the WhatsApp account is "inherited", including group memberships. A non-negligible percentage of the accounts we registered had been in active use, with personal and/or group messages arriving after account takeover. This in itself presents a significant privacy risk for these users, comparable to (and possibly worse than) privacy issues associated with disposable email addresses [46]. We did not use such accounts for our crawling attempts.

**Signal.** The Android client of Signal is open-source, which allows us to extract the requests for registration and contact discovery, and perform them efficiently through a Python script. We register new clients manually and use the authentication tokens created upon registration to perform subsequent calls to the contact discovery API. Signal (before switching entirely to TEE-based contact discovery) used truncated SHA-1 hashes of the phone numbers in the contact discovery request[9]. The response from the Signal server is either an error message if the rate limit has been reached, or the hashes of the phone numbers registered with Signal.

**Telegram.** Interactions with the Telegram service can be made through the official library TDLib [96], which is available for many systems and programming languages. To create a functioning client, each project using TDLib has to be registered with Telegram to receive an authentication token, which can be done with minimal effort. We use the C++ version to perform registration and contact discovery, and to potentially download additional information about Telegram users. The account registration is done manually by requesting a phone call to authenticate the number.

### 4.4 Ethical and Legal Considerations

We excessively query the contact discovery services of major mobile messengers, which we think is the only way to reliably estimate the success of our attacks in the real world. Similar considerations were made in previous works that evaluate attacks by crawling user data from production

[9]We used the legacy API; the new Intel SGX service does not use hashes.

systems (e.g., [104]). We do not interfere with the smooth operation of the services or negatively affect other users.

In coordination with the legal department of our institution, we design the data collection process as a pipeline creating only aggregate statistics to preserve user privacy and to comply with all requirements under the European General Data Protection Regulation (GDPR) [71], especially the data minimization principle (Article 5c) and regulations of the collection of data for scientific use (Article 89). Privacy sensitive information such as profile pictures are never stored, and all data processing is performed on a dedicated local machine.

### 4.5 Rate Limits and Abuse Protection

Each messenger applies different types of protection mechanisms to prevent abuse of the contact discovery service[10].

**WhatsApp.** WhatsApp does not disclose how it protects against data scraping. Our experiments in September 2019 show that accounts get banned when excessively using the contact discovery service. We observe that the rate limits have a leaky bucket structure, where new requests fill a virtual bucket of a certain size, which slowly empties over time according to a specified leak rate. Once a request exceeds the currently remaining bucket size, the rate limit is reached, and the request will be denied. We estimate the bucket size to be close to 120,000 contacts, while our crawling was stable when checking 60,000 new numbers per day. There seems to be no total limit of contacts per account: some of our test accounts were able to check over 2.8 million different numbers.

**Signal.** According to the source code [61], the Signal servers use a leaky bucket structure. However, the parameters are not publicly available. Our measurements show that the bucket size is 50,000 contacts, while the leak rate is approximately 200,000 new numbers per day. There are no bans for clients that exceed these limits: The requests simply fail, and can be tried again later. There is no global limit for an account, as the server does not store the contacts or hashes, and thus cannot determine how many different numbers each account has already checked.

While we only used Signal's hashing-based legacy API, Android clients at the time also synced with the new API based on Intel SGX and compared the results. We found that the new API has the same rate limits as the legacy API, allowing an attacker to use both with different inputs, and thus double the effective crawling rate.

Signal clients use an additional API to download encrypted profile pictures of discovered contacts. Separate rate limits exist to protect this data, with a leaky bucket size of 4,000 and a leak rate of around 180 profiles per hour.

**Telegram.** The mechanism used by Telegram to limit the contact discovery process differs from WhatsApp and Signal. As we find in our experiments, Telegram allows each account to add a maximum of 5,000 contacts, irrespective of the rate. Once this limit is exceeded, each account is limited to 100 new numbers per day. More requests result in a rate limit error, with multiple violations resulting in the ban of the phone number from the contact discovery service. The batch size for contact requests is 100 and performing consecutive requests with a delay of less than ≈8.3 s results in an immediate ban from the service.

In a response to the privacy issue discovered in August 2019 [18], where group members with hidden phone numbers can be identified through enumeration attacks, Telegram stated that once phone numbers are banned from contact discovery, they can only sync 5 contacts per day. We were not able to reproduce this behavior. Following our responsible disclosure, Telegram detailed additional defenses not triggered by our experiments (cf. § 9).

[10]There might be additional protections not triggered by our experiments.

### 4.6 Exposed User Data

All three messengers differ significantly regarding the amount of user data that is exposed.

**WhatsApp.** Users registered with WhatsApp can always be discovered by anyone through their phone number, yet the app has customizable settings for the profile picture, *About* text, and *Last Seen* information. The default for all these settings is *Everybody*, with the other options being *My Contacts* or *Nobody*. In recent Android versions it is no longer possible to save the profile picture of users through the UI, but it is possible to create screenshots through the Android Debug Bridge (ADB). The status text can be extracted through the UI Automator framework by accessing the text fields in the contact list view.

**Signal.** The Signal messenger is primarily focused on user privacy, and thus exposes almost no information about users through the contact discovery service. The only information available about registered users is their ability to receive voice and video calls. It is also possible to retrieve the encrypted profile picture of registered users through a separate API call, if they have set any [106]. However, user name and avatar can only be decrypted if the user has given explicit consented for the user requesting the information and has exchanged at least one message with them [59].

**Telegram.** Telegram exposes a variety of information about users through the contact discovery process. It is possible to access first, last, and user name, a short bio (similar to WhatsApp's *About*), a hint when the user was last online, all profile pictures of the user (up to 100), and the number of common groups. Some of this information can be restricted to contacts only by changing the default privacy settings of the account. There is also additional management information (such as the Telegram ID), which we do not detail here.

Surprisingly, Telegram also discloses information about numbers not registered with the service through an integer labeled `importer_count`. According to the API documentation [95], it indicates how many registered users store a particular number in their address book, and is 0 for registered numbers[11]. Importantly, it represents the *current* state of a number, and thus decrements once users remove the number from their contacts. As such, the `importer_count` is a source of interesting meta data when keeping a specific target under surveillance. Also, when crawlers attempt to compile comprehensive databases of likely active numbers for conducting sales or phishing calls (as motivated in § 1), having access to the `importer_count` increases the efficiency. And finally, numbers with non-zero values are good candidates to check on other messengers.

### 4.7 Our Evaluation Approach

We perform random lookups for mobile phone numbers in the US and collect statistics about the number of registered users, as well as the information exposed by them. The number space consists of 505.7 million mobile phone numbers (cf. § 2.2). We assume that almost all users sign up for these messengers with mobile numbers, and thus exclude landline and VoIP numbers from our search space. The US numbering plan currently includes 301 3-digit area codes, which are split into 1,000 subranges of 10,000 numbers each. These subranges are handed out individually to phone companies, and only 50,573 of the 301,000 possible subranges are currently in use for mobile phone numbers. To reach our crawling targets, we select numbers evenly from all subranges. While the enumeration success rate could be increased by using telephone number lists or directories as used for telephone surveys [58], this would come at the expense of lower coverage.

### 4.8 Our Crawling Results

The messengers have different rate limits, amount of available user information, and setup complexity. This results in different crawling speeds and number space coverage, and affects the type of

[11]Telegram clients use this count to suggest contacts who would benefit the most from registering.

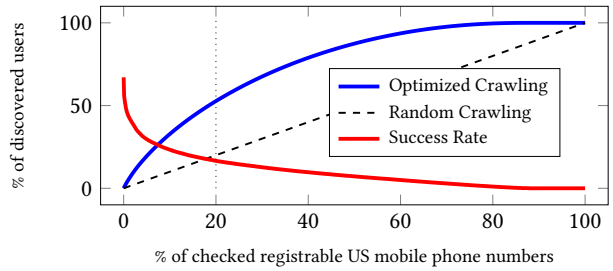



Fig. 3. Optimized crawling compared to random crawling based on the non-uniform distribution of registered WhatsApp users across the US mobile phone number space.

statistics that can be generated. However, note that the coverage achieved by our attack strategy is only limited by the resources (mainly time) spent on creating and operating accounts for crawling.

**WhatsApp.** For WhatsApp we use 25 accounts[12] over 34 days, each testing 60,000 numbers daily, which allows us to check 10 % of all US mobile phone numbers. For a subset of discovered users, we also check if they have public profile pictures by comparing their thumbnails to the default icon.

Our data shows that 5 million out of 50.5 million checked numbers are registered with WhatsApp, resulting in an average success rate of 9.8 % for enumerating random mobile phone numbers. The highest average for a single area code is 35.4 % for `718` (New York) and 35 % for `305` (Florida), while there are 209 subranges with a success rate higher than 50 % (the maximum is 67 % for a prefix in Florida). The non-uniform user distribution across the phone number space can be exploited to increase the initial success rate when enumerating entire countries, as shown in Fig. 3 for the US: with 20 % effort it is possible to discover more than 50 % of the registered users.

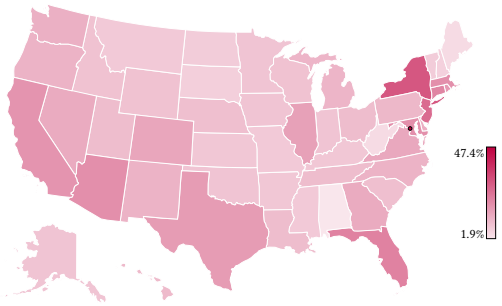
Extrapolating this data allows us to estimate the total number of WhatsApp accounts registered to US mobile phone numbers to be around 49.6 million. While there are no official numbers available, estimates from other sources place the number of monthly active WhatsApp users in the US at 25 million [21]. Our estimate deviates from this number, because our results include all registered numbers, not only active ones. Another statistic [22] estimates the number of US mobile phone numbers that accessed WhatsApp in 2019 at 68.1 million, which seems to be an overestimation based on our results.

For a random subset of 150,000 users we also analyzed the availability of profile pictures and *About* texts: 49.6 % have a publicly available profile picture and 89.7 % have a public *About* text. An analysis of the most popular *About* texts shows that the predefined (language-dependent) text is the most popular (77.6 %), followed by “Available” (6.71 %), and the empty string (0.81 %, including “.” and “*** no status ***”), while very few users enter custom texts.

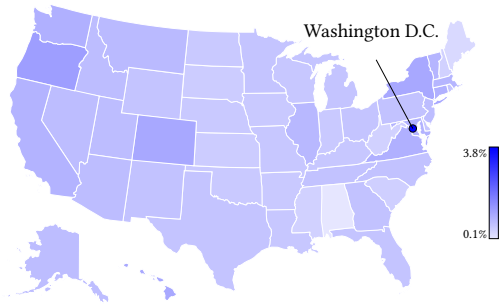
**Signal.** Our script for Signal uses 100 accounts over 25 days to check all 505 million mobile phone numbers in the US. Our results show that Signal currently has 2.5 million users registered in the US, of which 82.3 % have set an encrypted user name, and 47.8 % use an encrypted profile picture. We also cross-checked with WhatsApp to see if Signal users differ in their use of public profile pictures, and found that 42.3 % of Signal users are also registered on WhatsApp (cf. Tab. 7), and 46.3 % of them have a public profile picture there. While this is slightly lower than the average for WhatsApp users (49.6 %), it is not sufficient to indicate an increased privacy-awareness of Signal’s users, at least for profile pictures.

**Telegram.** For Telegram we use 20 accounts running for 20 days on random US mobile phone numbers. Since Telegram’s rate limits are very strict, only 100,000 numbers were checked during

[12]Less than 100 for Signal due to the overhead of running Android emulators.




(a) WhatsApp; the popularity is estimated based on enumerating 10 % of all possible US mobile phone numbers.




(b) Signal; Washington D.C. numbers are more than twice as likely to be registered with Signal than any other US area.

Fig. 4. Number of registered WhatsApp and Signal accounts of US states and Washington D.C. in relation to their population.

that time: 0.9 % of those are registered and 41.9 % have a non-zero `importer_count`. These numbers have a higher probability than random ones to be present on other messengers, with 20.2 % of the numbers being registered with WhatsApp and 1.1 % registered with Signal, compared to the average success rates of 9.8 % and 0.9 %, respectively. Of the discovered Telegram users, 44 % of the crawled users have at least one public profile picture, with 2 % of users having more than 10 pictures available.

**Summary and Comparison.** An overview of the tested messengers, our crawling setup, and our most important results is given in Tab. 6. Our crawling of WhatsApp, Signal, and Telegram provides insights into privacy aspects of these messengers with regard to their contact discovery service.

The first notable difference is the storage of the users' contact information, where both WhatsApp and Telegram retain this information on the server, while Signal chooses not to maintain a server-side state to better preserve the users' privacy. This practice unfortunately requires significantly higher rate limits for the contact discovery process, since all contacts of a user are compared on every sync, and the server has no possibility to compare them to previously synced numbers. While Telegram uses the server-side storage of contacts to enforce strict rate limits, WhatsApp nevertheless lets individual clients check millions of numbers.

With its focus on privacy, Signal excels in exposing almost no information about registered users, apart from their phone number. In contrast, WhatsApp exposes profile pictures and the *About* text for registered numbers, and requires users to opt-out of sharing this data by changing the default settings. Our results show that only half of all US users prevent such sharing by either not uploading an image or changing the settings. Telegram behaves even worse: it allows crawling multiple images and also additional information for each user. The `importer_count` offered by its API even provides information about users not registered with the service. This can help attackers to acquire likely active numbers, which can be searched on other platforms.

Our results also show that many users are registered with multiple services (cf. Tab. 7), with 42.3 % of Signal users also being active on WhatsApp. We only found 2 out of 10,129 checked users on all three platforms (i.e., less than 0.02 %). In Fig. 4, we visualize the popularity of WhatsApp and Signal for the individual US states and Washington D.C. On average, about 10 % of residents have mobile numbers from another state [32], which may obscure these results to some extent. Interestingly, Washington D.C. numbers are more than twice as often registered on Signal than numbers from any other state, with Washington D.C. also being the region with the most non-local numbers (55 %) [32].

Table 6. Comparison of surveyed messengers.

| Messengers | WhatsApp | Signal | Telegram |
|---|---|---|---|
| Contact Discovery Method | Clear | Hashing | Clear |
| Rate Limits | 60k / d | 120k / d | 5k + (100 / d) |
| Our Crawling Method | UI Automator | (Legacy) API | API |
| # US Numbers Checked | 46.2 M | 505.7 M | 0.1 M |
| Coverage of US Numbers | 10 % | 100 % | <0.02 % |
| Success Rate for Random US Number | 9.8 % | 0.5 % | 0.9 % |
| # US Users Found | 5.0 M | 2.5 M | 908 |
| # US Users (estimated) | 49.6 M | 2.5 M | 4.6 M |
| Default Privacy Settings / Information Exposure | | | |
| Profile Picture | Public | Explicit Share | Public |
| Status | Public | – | Public |
| Last Online | Public | – | Public |
| Option to Hide Being Online | ✗ | ✓ | ✓ |
| Option to Prevent Being Discovered by Phone Number | ✗ | ✗ | ✓ |

Table 7. Registration of multiple messengers for US mobile phone numbers.

| Users of \ also use | WhatsApp | Signal | Telegram |
|---|---|---|---|
| WhatsApp | – | 2.2 % | 5.1 % |
| Signal | 42.3 % | – | 8.6 % |
| Telegram | 46.5 % | 5.3 % | – |

## 5 INCREMENTAL CONTACT DISCOVERY (ICD)

To mitigate crawling attacks conducted by adversaries who are registered as users (cf. our threat model defined in § 4), we propose a new rate-limiting scheme for contact discovery in messengers without server-side contact storage (e.g., Signal). Setting strict limits for services without server-side contact storage is difficult, since the server cannot determine if the user's input in discovery requests changes significantly with each invocation. Our approach called *incremental contact discovery* (ICD) provides strict improvements over existing solutions, as it enables the service to enforce stricter rate limits with very little overhead, increased lookup performance, and without degrading usability or privacy. For a unified defense against both hash reversal and crawling attacks, ICD is trivially compatible with cryptographic private set intersection (PSI) protocols as well as other mitigation techniques as presented in § 7.

### 5.1 Approach

ICD is based on the observation that the database of registered users changes only gradually over time. Given that clients are able to store the last state for each of their contacts, they only need to query the server for changes since the last synchronization. Hence, if the server tracks database changes (new and unsubscribed users), clients who connect regularly only need to synchronize with the set of recent database changes. This enables the server to enforce stricter rate limits on the

full database, which is only needed for initial synchronization, for newly added client contacts, and whenever the client fails to regularly synchronize with the set of changes. Conversely, enumeration attacks require frequent changes to the client set, and thus will quickly exceed the rate limits when syncing with the full database.

**Assumptions.** In line with the observed rate-limiting approaches, we assume that each user has at most $N$ contacts, which can be synced in an interval $T_\Delta$. This set changes slowly, i.e., only by several contacts per day. Another reasonable assumption is that the server database of registered users does not significantly change within short time periods, e.g., only 0.5 % of users join or leave the service per day (cf. § 5.2).

**Algorithm.** The server of the service provider stores two sets of contacts: the full set $S$ and the delta set $S_\Delta$. $S$ contains all registered users, while $S_\Delta$ contains only information about users that registered or unregistered within the last $T$ days. Both sets, $S$ and $S_\Delta$, are associated with their own leaky buckets of size $N$, which are empty after $T$ and $T_\Delta$ days, respectively. The server stores leaky bucket values $t$ and $t_\Delta$ for each client, which represent the (future) points in time when the leaky buckets will be empty for requests to $S$ and $S_\Delta$, respectively.

A newly registered client syncs with the full set $S$ to receive the current state of the user's contacts. For subsequent syncs, the client only syncs with $S_\Delta$ to receive recently changed contacts, provided that it synchronizes at least every $T$ days. If the client is offline for a longer period of time, it can sync with $S$ again, since the leaky bucket associated with it will be empty. New contacts added by the user are initially synced with $S$ in order to learn their current state.

The synchronization with $S$ is given in Alg. 1. It takes as inputs the server's set $S$, the maximum number of contacts $N$, and the associated time $T$ which is required to drain a full bucket. The client provides the set of contacts $C$ and the server provides the client's corresponding bucket parameter $t$, i.e., the time when the client's leaky bucket is empty. The output is the set $D$ which is the intersection of $C$ with $S$, or an error, if the rate limit is exceeded.

**Algorithm 1** Synchronization with full set $S$

**Input:** $S, N, T, C, t$
**Output:** $D$
1: $t_{cur} \leftarrow \mathbf{max}(t, \mathbf{current_time}) + |C|/N \times T$
2: **if** $t_{cur} > \mathbf{current_time} + T$ **then**
3: | raise RateLimitExceededError
4: $t \leftarrow t_{cur}$
5: **return** $C \cap S$

**Algorithm 2** Synchronization with delta set $S_\Delta$

**Input:** $S, S_\Delta, N, T_\Delta, C_\Delta, t_\Delta$
**Output:** $R_\Delta$
1: $t_{cur} \leftarrow \mathbf{max}(t_\Delta, \mathbf{current_time}) + |C_\Delta|/N \times T_\Delta$
2: **if** $t_{cur} > \mathbf{current_time} + T_\Delta$ **then**
3: | raise RateLimitExceededError
4: $t_\Delta \leftarrow t_{cur}$
5: **return** $\{(x, x \in S) \text{ for } x \in C_\Delta \cap S_\Delta\}$

When a client initiates a sync with $S$, the algorithm calculates $t_{cur}$, the new (future) timestamp when the client's leaky bucket would be empty (line 1). Here, $|C|/N \times T$ represents the additional time which the bucket needs to drain. If $t_{cur}$ is further into the future than $T$ (line 2), this indicates that the maximum bucket size is reached, and the request will abort with an error (line 3). Otherwise, the leaky bucket is updated for the client (line 4), i.e., the time when the client's leaky bucket will be empty is stored, and the intersection between the client set $C$ and the server set $S$ is returned (line 5).

The synchronization with $S_\Delta$ shown in Alg. 2 is quite similar. Here, the server supplies $S$, $S_\Delta$, $N$, $T_\Delta$, and $t_\Delta$, and the client provides the previously synced contacts $C_\Delta$. The main difference to Alg. 1 is that it outputs $R_\Delta$, i.e., the requested contacts that changed (registered or unregistered) within the last $T$ days together with their current state (line 5). Note that $S$ is only used to check the state for contacts in $S_\Delta$.

**Implementation.** We provide an open-source proof-of-concept implementation of our ICD scheme written in Python at https://contact-discovery.github.io/. It uses Flask [69] to provide a REST API for performing contact discovery and can be useful for service providers and their developers to facilitate integration of our idea into real-world applications.

Table 8. Effect of change rate $CR$ on the optimal choice for $T$, the discovery rate $DR$ for ICD, and the improvement compared to Signal's leaky bucket approach.

| $CR$ (in %/d) | $T$ (in d) | $DR$ (in #contacts/d) | Improvement |
|---|---|---|---|
| 0.01 | 50.0 | 10.0 | 100.0x |
| 0.05 | 22.4 | 22.4 | 44.7x |
| 0.1 | 15.8 | 31.6 | 31.6x |
| 0.5 | 7.1 | 70.7 | 14.1x |
| 1.0 | 5.0 | 100.0 | 10.0x |
| 2.0 | 3.5 | 141.4 | 7.1x |

### 5.2 Evaluation

**Overhead.** Our incremental contact discovery introduces only minimal server-side storage overhead, since the only additional information is the set $S_\Delta$ (which is small compared to $S$), as well as the additional leaky bucket states for each user. The runtime is even improved, since subsequent contact discovery requests are only compared to the smaller set $S_\Delta$. On the client side, the additional storage overhead is introduced by the need to store a timestamp of the last sync to select the appropriate set to sync with, as well as a set of previously unsynced contacts $C_\Delta$.

**Improvement.** To evaluate our construction, we compare it to the leaky bucket approach currently deployed by Signal. Concretely, we compare the *discovery rate* of the schemes, i.e., the number of users that can be found by a single client within one day with a random lookup strategy. Rate-limiting schemes should minimize this rate for attackers without impacting usability for legitimate users. For Signal, the discovery rate is $DR = SR \cdot N/T_\Delta$, where $SR$ is the success rate for a single lookup, i.e., the ratio between registered users and all possible (mobile) phone numbers. Based on our findings in § 4.8, we assume $SR = 0.5\,\%$, $N = 50{,}000$, and $T_\Delta = 0.25$ days, which results in a discovery rate of $DR = 1{,}000$/day for Signal's leaky bucket approach.

For our construction, the discovery rate is the sum of the rates $DR_T$ and $DR_\Delta$ for the buckets $S$ and $S_\Delta$, respectively. While $DR_T$ is calculated as $DR_T = SR \cdot N/T$, $DR_\Delta$ is calculated as $DR_\Delta = SR \cdot CR \cdot N \cdot T/T_\Delta$, where $CR$ is the change rate of the server database. The discovery rate is minimized when $DR_T = DR_\Delta$, and it follows that $T = \sqrt{T_\Delta/CR}$. With Signal's parameters, the total discovery rate for our construction can be calculated as $DR = 1{,}000 \cdot \sqrt{CR}$/day, and the improvement factor is exactly $1/\sqrt{CR}$.

In reality, the expected change rate depends on the popularity of the platform: Telegram saw 1.5 M new registrations per day while growing from 300 M to 400 M users [33], corresponding to a daily change rate of ≈0.5 %. WhatsApp, reporting 2 billion users in February 2020 [35] (up from 1.5 billion in January 2018 [26]), increases its userbase by an average of 0.05 % per day. Compared to Signal's rate limiting scheme, ICD results in an improvement of 14.1x and 44.7x for Telegram's and WhatsApp's change rate, respectively (cf. Tab. 8). Even at a theoretical change rate of 25 % per day, incremental discovery is twice as effective as Signal's current approach. Crawling entire countries would only be feasible for very powerful attackers, as it would require over 100k registered accounts (at $CR = 0.05\,\%$) to crawl, e.g., the US in 24 hours. It should be noted that in practice the change rate $CR$ will fluctuate over time.

**Optimal Parameters for Incremental Contact Discovery.** Given that the popularity of mobile messengers varies over time, the change rate $CR$ of the server database is not a fixed value but varies continuously. This results in different optimal choices for the time $T$. The inevitable non-optimal values for $T$ between adjustments result in higher discovery rates than the possible

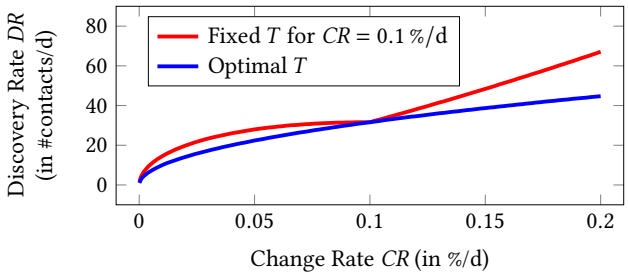



Fig. 5. Minimal discovery rate for different change rates $CR$ with optimal choices for $T$ (for Signal's parameters) compared to the discovery rate for fixed $T$ when estimating $CR = 0.1\,\%/\text{d}$.

minimum: If $CR$ is higher than expected, more users can be found by observing $S_\Delta$. If $CR$ is lower than expected, the rate limits for $S$ are too generous.

The relative error between the minimal and the actual discovery rate can be calculated as $ER = 0.5 \cdot |1 - CR/CR_{est}|$, where $CR$ is the actual change rate and $CR_{est}$ is the estimated one used for setting $T$. Thus, if the real change rate is underestimated by a factor of 2x, the discovery rate will be 50 % higher than intended. For the parameters used by Signal, Fig. 5 shows how the discovery rate behaves compared to the minimal one when a constant change rate of $CR = 0.1\,\%/\text{d}$ is assumed. Obviously, underestimating the change rate is more problematic than overestimating it. In a production environment it therefore may be beneficial to set $CR$ slightly higher than the expected value to deal with fluctuations. An implementation with dynamic sets, as outlined in § 5.3, could be an option for platforms where the change rate fluctuates more strongly and frequent adjustments of $T$ are required.

**Privacy Considerations.** If attackers can cover the whole number space every $T$ days, it is possible to find all newly registered users and to maintain an accurate database. This is not different from today, as attackers with this capacity can repeatedly sweep the full number space. Using the result from Alg. 2, users learn if a contact in their set has (un)registered in the last $T$ days, but this information can currently also be revealed by simply storing past discovery results.

### 5.3 Generalization

Our construction can be generalized to further decrease an attacker's efficiency. This can be achieved by using multiple sets containing the incremental changes of the server set over different time periods (e.g., one month, week, and day) such that the leak rate of $S$ can be further decreased. It is even possible to use sets dynamically chosen by the service without modifying the client: each client sends its timestamp of the last sync to the service, which can be used to perform contact discovery with the appropriate set.

For a unified defense against both hash reversal and crawling attacks, ICD can be combined with private set intersection (PSI) protocols (cf. § 7.1 and § 8.1) by applying PSI to each of the different set synchronizations individually. Likewise, ICD is compatible with all other mitigation techniques discussed in § 7.

## 6 DIFFERENTIAL CONTACT DISCOVERY (DCD)

We propose an alternative scheme for effective rate limiting of contact discovery queries called *differential contact discovery* (DCD). This scheme mitigates crawling attacks conducted by adversaries who are registered as users (cf. our threat model defined in § 4) by letting the service provider restrict the amount of changes to the client's contacts without the requirement to store any user contacts

on the server. The construction is efficient in terms of storage, computation, and transmission size. Furthermore, it is designed to be combined with Signal's Intel SGX API [64], thereby also providing reasonable privacy of the users' social graphs, and we additionally discuss potential combinations with cryptographic private set intersection (PSI) protocols for another unified defense against both hash reversal and crawling attacks.

### 6.1 Approach

For each contact discovery request, the client provides the contact set used in the previous invocation, along with the changes (i.e., additions and deletions), resulting in the contact set for the current request. The server creates and stores a statistically hiding and computationally binding commitment for the contacts, and gives the opening information to the client. The commitment, in combination with the set of previous contacts, allows the server to verifiably determine the changes to the user's contacts since the last request, and use this information to perform effective rate limiting. The server does not store the opening information, so that no information about the user's contacts is revealed in case of a data breach on the server side that leaks the commitment (as the commitment scheme is statistically hiding).

We realize this commitment through a cryptographic hash of the set of previous contacts (stored on the server) that includes a random nonce (or salt), which in turn is stored on the client side. Our construction requires only two hash operations to compute and verify the digest of the client's contact set.

**Assumptions.** The main requirement for the algorithm is that both client and server are able to store information about the previous contact discovery request. For the client, this includes all identifiers (e.g., phone numbers) and a nonce generated in the previous request. For the server, a single digest of a small constant size is stored for each user and updated with each request.

**Algorithm.** The client has two sets: $C_{prev} = \{c_1, c_2, \ldots, c_{m_{prev}}\}$ with the identifiers used in the previous request and $C_{cur}$ with all identifiers to use in the current request. From these sets, the client computes $C_{add} = C_{cur} \setminus C_{prev}$ with all newly added identifiers and $C_{del} = C_{prev} \setminus C_{cur}$ with all identifiers deleted since the previous request. The client then sends $C_{prev}$, $C_{add}$, and $C_{del}$ as well as the nonce $n_{prev}$ received in the previous request to the server[13].

The server first computes the hash $h_{prev} = H(n_{prev}|c_1|c_2|\ldots|c_{m_{prev}})$ and compares it to the value of $h_{prev}$ stored for the client from the previous request[14]. Matching digests confirm that the set $C_{prev}$ was used as $C_{cur}$ in the previous request. The server then constructs the current set of the client $C_{cur} = (C_{prev} \cup C_{add}) \setminus C_{del}$. It can now check the size of $C_{add}$ against the rate limit $N_{change}$, which determines how many new contacts the client may have. Furthermore, the server can check the size of $C_{cur}$ against the total rate limit $N$ to prevent clients from requesting the state of too many contacts simultaneously. If both checks succeed, the server generates a new random nonce $n_{cur}$ and calculates the current hash $h_{cur}$ from the nonce and the current set $C_{cur}$. The hash $h_{cur}$ is then stored as $h_{prev}$ for the client and the server sends $n_{cur}$ together with the current registration state of all identifiers in $C_{cur}$ to the client.

The client receives the information about the contacts, sets $C_{cur}$ as the new $C_{prev}$, and stores $n_{cur}$ for the next request. We depict the whole protocol including all algorithm steps in Fig. 6. For a statistically hiding and computationally binding commitment scheme in the random oracle model,

[13]For the first request, $n_{prev}$ can be left empty.

[14]For reproducible results, the server must always concatenate the elements in the sets the same way, e.g., by sorting the elements before concatenating.

**Input:**
Client: previous nonce $n_{prev}$, previous input set $C_{prev} = \{c_1, \ldots, c_{m_{prev}}\}$, new input set $C_{cur} = \{c_1, \ldots, c_{m_{cur}}\}$
Server: set of all users $S$, previous hash $h_{prev}$, rate limits $N_{change}$ and $N$
Common: symmetric security parameter $\kappa$

**Output:**
Client: $C_{cur} \cap S$ or $\perp$ (if rate limits exceeded, i.e., $|C_{cur} \setminus C_{prev}| > N_{change}$ or $|C_{cur}| > N$), new nonce $n_{cur}$
Server: new hash $h_{cur}$

| Client | | Server |
|---|---|---|
| $C_{add} = C_{cur} \setminus C_{prev}$<br>$C_{del} = C_{prev} \setminus C_{cur}$ | | |
| | $\xrightarrow{C_{prev}, C_{add}, C_{del}, n_{prev}}$ | |
| | | If $h_{prev} \neq H(n_{prev} \vert c_1 \vert c_2 \vert \ldots \vert c_{m_{prev}})$ then **abort**<br>$C_{cur} = (C_{prev} \cup C_{add}) \setminus C_{del}$<br>If $\vert C_{add} \vert > N_{change}$ or $\vert C_{cur} \vert > N$ then **abort**<br>$n_{cur} \in_R \{0,1\}^{3\kappa}$<br>$h_{cur} = H(n_{cur} \vert c_1 \vert c_2 \vert \ldots \vert c_{m_{cur}})$ |
| | $\xleftarrow{C_{cur} \cap S, n_{cur}}$ | |
| **store** $n_{cur}$ as $n_{prev}$<br>**store** $C_{cur}$ as $C_{prev}$ | | **store** $h_{cur}$ as $h_{prev}$ |

Fig. 6. Differential contact discovery.

we choose $H$ as a cryptographic hash function $H = \{0,1\}^* \rightarrow \{0,1\}^{2\kappa}$ and $n_{cur}$ as uniformly random in $\{0,1\}^{3\kappa}$, where $\kappa$ is a symmetric security parameter, e.g., $\kappa = 128$[15].

## 6.2 Evaluation

**Overhead.** The overhead of DCD can be quantified in terms of required computation and storage for client and server as well as the communication between the two.

The client must store a single nonce as well as the contacts used in the previous request, which requires $O(N)$ additional storage, where $N$ is the total number of allowed contacts. The transmitted data to the server is in $O(N)$ as well, as it consists of all previous contacts, the contacts added and deleted since the previous request, and a single nonce. Considering the worst case (i.e., all contacts changed since the previous request), both the stored and transmitted client data are twice as large as with a conventional approach sharing only current contacts. In comparison to conventional contact discovery, the server must perform two additional hash computations over the inputs ($O(N)$) and store a single hash value for each client ($O(1)$).

In terms of concrete communication overhead, assume there are $m = m_{prev} = m_{cur}$ contacts $c_i$ in a client's address book, each contact's mobile phone number can be represented with 64 bit, and each nonce $n$ is 384 bit in size. As visualized in Fig. 6, the DCD protocol requires $4m|c| + 2|n|$ communication (in comparison to at most $2m|c|$ for regular contact discovery schemes). Although this doubles the concrete overhead, in an extreme case with $m = 100{,}000$, this results in a total network communication of only 3.2 MB. Hence, the network overhead is completely acceptable.

[15]Informally, the commitment scheme is computationally binding as finding a collision in $H$ requires $O(2^\kappa)$ queries due to the birthday paradox, and statistically hiding as trying all $2^{3\kappa}$ possible nonces $n_{cur}$ for a commitment $h_{cur}$ will lead to on average $2^{3\kappa}/2^{2\kappa} = 2^\kappa$ plausible matches for any $C'_{cur}$.

**Improvement.** For a conventional contact discovery service without server-side contact storage, rate limiting can only be performed against the total number of contacts in the discovery requests. Since legitimate users may have thousands of contacts, a client can therefore query the same large amount of new numbers with each request. In contrast, our construction allows the server to determine the exact number of changed contacts for each user, which can reasonably be assumed to be a small fraction of the total amount. The rate-limiting improvement against enumeration attacks can therefore be expressed as the ratio between the amount of total and the amount of changed contacts a user can query. Assuming that at most 1 % of the totally allowed contacts $N$ change per day, Signal's rate limits (50,000 total contacts, four times per day) could be throttled to $N_{change} = 500$ changed contacts per day, an improvement by a factor of 400x. Additionally, with our construction, users could be allowed to check the state of their (unchanged) contacts as often as they want.

**Privacy Considerations.** Compared to conventional contact discovery, DCD exposes no additional information about users to a curious or compromised provider, as contact identifiers are not stored on the server. The client temporarily exposes the contacts from the previous request, which reveals the added and deleted contact identifiers to the server during the request, but this information could also be computed by the server by simply storing the identifiers of the last contact discovery for each client.

Furthermore, the cryptographic hash values stored for each user do not reveal any information about contacts included in the users' requests. This is achieved through the statistical hiding property of our commitment scheme, which uses a sufficiently sized random nonce. The binding property of our commitment scheme ensures that the probability for a computationally bounded client to find $n'_{prev} \neq n_{prev}$ and $C'_{prev} \neq C_{prev}$ with $h_{prev} = H(n'_{prev}|C'_{prev})$ is negligible.

Our scheme can also easily prevent *identifier stuffing*. For this kind of attack, several identifiers are concatenated into a single identifier for one request and are treated separately for the next, e.g., by choosing $c_3 = c_1|c_2$, $C_1 = \{c_3\}$, and $C_2 = \{c_1, c_2\}$. Both sets would then produce the same hash and suggest no change to the contacts, although different identifiers would be checked in each request. We can therefore either enforce a specific length for all identifiers (e.g., by hashing them) or include a dedicated separator (e.g., "-") between identifiers in the hash calculation.

**Deployment Considerations.** With DCD, service providers can freely implement their preferred rate-limiting strategy: strategies as used by Telegram (limit the daily changes when the client's contact limit is exceeded) can be implemented just as easily as limiting the number of changed contacts in a specific interval (i.e., leaky bucket).

In the real world, users may delete and reinstall the client application, or otherwise reset the client state (e.g., by registering a new number). In this case, the client is not able to provide the last nonce and contact state in the request, and must re-initialize the protocol with an empty $n_{prev}$. Service providers should either view all contacts as changed for these requests, or limit the number of initial requests within a defined time interval. A reasonable threshold must then be determined to strike a balance between protection of the service and hindering legitimate user behavior.

**Application to Signal.** Signal's current approach to private contact discovery using Intel SGX can trivially support our scheme, since all operations (comparisons, hashing, data storage) can be efficiently performed inside the enclave.

Furthermore, the previously discussed potential limitation of DCD with regard to client-side storage across installations can be amended by employing Signal's secure value recovery [62], which allows clients to securely store data with the Signal service and recover them using a weak secret. While the system cannot provide secure storage for the changing set of the users' contacts due to performance reasons, the system can assist users with new installations to continue the protocol when their contacts did not change since the last invocation. In this case, the user only

has to retrieve the last nonce. This can be done efficiently without requiring repeated interactions with secure value recovery. For this, the user initially creates a dedicated private key for contact discovery, places it in Signal's secure value recovery storage, and reveals the corresponding public key to the contact discovery server. The server can then encrypt each new nonce with the public key associated with the user and save it in an arbitrary storage system. To recover the last nonce in case of a new installation, the client can request the encrypted nonce from the contact discovery server and use the private key retrieved from secure value recovery to decrypt it.

### 6.3 Generalization

DCD as described above works with arbitrary contact identifiers, e.g., phone numbers or hashed phone numbers. With minor adjustments, it is also possible to combine DCD with private set intersection (PSI) protocols (cf. § 7.1 and § 8.1) as an alternative to Signal's current approach to provide confidentiality of the client's contacts during the request. However, one requirement when selecting a PSI protocol is that the encrypted/blinded contact identifiers can be persistent across many protocol invocations. This is necessary to allow the server to verify the set of previous contacts through the hash-based verification. Additionally, integrating PSI protocols with DCD does not completely conform to the ideal functionality of set intersection as the number of changed contacts is explicitly revealed to the server for the purpose of rate limiting. However, DCD is unconditionally compatible with all other mitigation techniques discussed in § 7.

### 6.4 Comparison to Incremental Contact Discovery (ICD)

DCD is superior to ICD (cf. § 5) with regard to the rate limits that can be applied. This is because DCD directly reveals the number of changed contacts to the server in order to prevent enumeration attacks instead of relying on other assumptions (e.g., slow server set changes). Deploying the incremental approach on top of DCD is possible but provides no additional benefits, since the rate limits are already stricter: The first DCD request is similar to the full set synchronization in ICD and subsequent DCD requests are similar to the change set synchronization in ICD. However, full set synchronization in ICD must be performed more frequently (new install/client longer offline) than initial requests in DCD (only new installs). Moreover, the rate limit for change set synchronizations in ICD must consider multiple daily requests with unchanged contacts, which do not count against the rate limit in DCD.

To compare the two schemes quantitatively, we select the parameters for ICD as in § 5.2 (using $CR = 0.5\,\%$). This results in a discovery rate of $DR_{ICD} \approx 70.7/\text{day}$ (which is the number of users that can be found by a single client within one day with a random lookup strategy). Using the same parameters for DCD, we achieve a discovery rate of $DR_{DCD} = SR * N_{change} \approx 2.5/\text{day}$ (using $N_{change} = 500$). This is an improvement by factor 28x over ICD and by factor 400x over Signal's leaky bucket approach (cf. Tab. 8 and § 6.2). However, such comparisons heavily depend on the assumptions for $CR$, $T_\Delta$, $N$, and $N_{change}$ ($SR$ is a constant factor in both cases). The main advantage of ICD though is the straight-forward integration with PSI protocols for a unified defense against both hash reversal and crawling attacks.

## 7 MITIGATION TECHNIQUES

The privacy problems discussed in the previous sections stem from two distinct issues: hashes of phone numbers used in mobile contact discovery can be easily reversed and all possible phone numbers can be enumerated in contact discovery requests. We first discuss mitigations against hash reversal attacks and then measures to detect as well as prevent crawling via enumeration attacks in addition to our novel rate-limiting schemes presented in § 5 and § 6. To achieve optimal

results, multiple measures from all categories must be combined. In fact, it is possible to combine all mitigations with a few exceptions that we discuss explicitly.

### 7.1 Hash Reversal Mitigations

**Private Set Intersection.** Protocols for private set intersection (PSI, cf. § 8.1) can compute the intersection between the registered user database and the users' address books in a privacy-preserving manner. Thus, provably secure PSI protocols in contact discovery entirely prevent attacks where curious service providers can learn the users' social graph. However, PSI does not prevent enumeration attacks, and in fact interferes with efforts to detect if clients replace the majority of inputs for each execution. Thus, these protocols must be combined with protections against enumeration attacks by restricting the number of protocol executions and inputs to the minimum.

Moreover, PSI protocols currently do not achieve practical performance for a very large number of users (cf. § 8.1). One of the currently most performant constructions [50] requires each user to initially download an encrypted and compressed database of ≈8 GiB for a service like WhatsApp with about 2 billion users [35]. More performant PSI designs either rely on rather unrealistic trust assumptions (e.g., non-colluding servers) or on trusted hardware [64] that provides no provable security guarantees and often suffers from side-channel vulnerabilities [10].

**Database Partitioning.** Previous works [50] and the Signal developers [63] have considered to make PSI practical by partitioning the user database into smaller subsets to reduce the communication overhead. If the database is split (based on number prefixes into continents, countries, states, or even regions), the service provider learns only incomplete information about a user's social graph. There are limitations to the practicality of this approach, mainly that users with diverse contacts will incur a heavy performance penalty by having to match with many partitions. It is questionable that a reasonable performance/privacy trade-off can be achieved for popular services with a diverse user base.

**Strengthened Hashing-based Protocols.** Hashing-based contact discovery schemes are an insecure alternative to PSI: as demonstrated in § 3, phone number hashes can be brute-forced on consumer hardware when using hash functions such as SHA1 and SHA-256. However, there are several strategies from the domain of password hashing that increase the difficulty of hash reversal, which can be adapted to improve the strength of phone number hashes.

While it is not possible to introduce individual salts for each identifier as with password hashing, a global salt inserted into the hash function can prevent reusable rainbow tables (cf. § 3.3). Rotating the salt in short intervals also makes hash databases (cf. § 3.1) less attractive, but requires the client and server to regularly recompute the identifiers used in the protocol. Also, brute-force attacks are still a viable option for attackers.

Another alternative is to perform *key streching*, i.e., performing multiple rounds of the hash function. This increases the brute-force time for an attacker by a constant factor, but also puts the same burden on client devices and the server. As such, it can only be used to a limited extent, since users of slower devices will experience usability issues. Key derivation functions like bcrypt [82] or Argon2 [7], which are specifically designed to be non-parallelizable and resist brute-force attacks, can significantly reduce speedup through parallelization on GPUs and ASICs [41]. For example, existing benchmarks show that with bcrypt only 2.9 kHashes/s and with Argon2 only 2.6 Hashes/s can be computed on a GPU compared to 794.6 MHashes/s with SHA-1 [41]. There are however reports of FPGAs designed specifically to crack bcrypt hashes [89].

These measures will not be sufficient against very powerful adversaries, but can at least increase the costs of hash reversal attacks by a factor of even millions. However, the performance penalty will also affect clients when hashing their contacts, as well as the server, when updating the database.

**Alternative Identifiers.** Privacy-concerned users should be able to provide an alternative identifier (e.g., a user name or email address, which is the standard for social networks) instead of their phone number. Random or user-chosen identifiers with high entropy increase the search space for brute-force attacks and thus improve resistance against hash reversal. This however does not protect the phone numbers in the user's address book from being exposed through hash reversal, and also requires users to exchange these identifiers before discovering each other (which decreases usability and somewhat contradicts the notion of contact discovery). Signal plans to introduce alternative identifiers [66].

**Larger Phone Number Space.** In § 2.3, we observed that some countries have a much larger number space than others, which makes crawling these countries much more difficult. Telecommunication companies of vulnerable countries could therefore agree to maintain larger number blocks to increase the search space for attackers. However, enlargement of the search space will only affect new phone numbers issued by the providers, leaving legacy numbers less protected. It will also not increase protection against enumeration attacks based on phone number lists resulting from data breaches.

**Selective Contact Permissions.** iOS and Android require apps to ask for permission to access the user's address book, which is currently a binary choice. This choice could be extended to allow the selection of only specific contacts or contact groups to be revealed to third-party apps, e.g., via attribute check boxes such as "private" in the phone's address book. There already exist wrapper apps for specific messengers with similar functionality (e.g., WhatsBox [4] for WhatsApp). While being a technically sound solution, note that this type of mitigation shifts the responsibility of protecting sensitive contacts entirely to users. Until system-wide options are available, users may prefer to not store sensitive contacts in the phone's address book if messengers have access permissions, or revoke contact access permissions completely.

**On-Demand Registration Check.** Contact discovery is a convenience feature offered by most messengers to give users an overview of all their address book contacts that can be found on the service. There are, however, alternative approaches: Apple's iMessage only checks the availability of users when a conversation is started or continued, and falls back to regular text messages when the chat partner is not registered. This on-demand check creates a seamless user experience, without the necessity to repeatedly query the registration state of all contacts in the address book. While this approach does not exhibit some of the problems associated with other forms of contact discovery (e.g., automatically exposing sensitive contacts to the server and handling a large number of contacts), it requires the availability of another messaging option with all contacts to use when iMessage communication is not available. This is difficult to achieve, or may not be desirable, e.g., due to the lack of end-to-end encryption, or the resulting cost when using text messages.

### 7.2 Crawling Detection

**Honeypot numbers.** Rate limits cannot prevent attackers from crawling at a low rate. As suggested in [54], service providers could use honeypots for detecting such attackers, i.e., they could acquire and register several phone numbers, and detect if any of these numbers are matched during contact discovery. A positive match would indicate either a false positive (e.g., a typo when storing a contact) or an attempt of crawling. Due to the potential of false positives, it would be more reasonable to closely monitor the activity of such accounts rather than blocking them instantly. Note that this mitigation technique cannot be directly combined with PSI (as PSI does not reveal any output to the service provider). However, in practice the clients will report the PSI output to the service provider in order to receive additional user data, and then the matches with honeypot numbers can be detected.

**Comparison to Data Leaks.** Data scraping can be performed with higher efficiency if the attacker can check phone numbers which are more likely to be registered with the service (cf. § 4.8). Attackers can use databases of phone numbers that can be found online (cf. § 8.2), often from other crawling campaigns or hacks, to increase the initial success rate. It would therefore be possible to compare the synchronized contacts of accounts against known data leaks and identify accounts that overwhelmingly query numbers from leaked databases. This approach has the disadvantages that service providers must acquire and store databases of data breaches, and can only detect accounts exploiting known data leaks. It can also be thwarted by including random identifiers in each request, and may produce a high number of false positives. Here, the same restrictions for a combination with PSI apply as for honeypot numbers.

**Further Heuristics.** Service providers could use a combination of several heuristics to detect abnormal user behavior that indicates a crawling attempt. Such heuristics could include an unusually large amount of contacts in the address book, exceptionally many syncing requests, constantly changing contacts, and the lack of messaging activity. However, using such heuristics to automatically ban accounts is error-prone. They can also be circumvented by more sophisticated attackers that adapt their behavior to evade detection. The need to store additional metadata about users may also create additional privacy risks for users. Furthermore, a combination with PSI, which hides certain properties of contact discovery requests, reduces the number of available heuristics.

### 7.3 Crawling Prevention

We now discuss mitigation strategies that can help to inherently prevent crawling attacks. Furthermore, since many messenger apps give users the possibility to add additional information to their profile, we also discuss countermeasures that can limit the exposure of sensitive private information through the scraping of user profiles.

**Stricter Rate Limits.** Rate limits are a trade-off between user experience and protection of the service. If set too low, users with no malicious intent but unusual usage patterns (e.g., a large number of contacts) will exceed these limits and suffer from a bad user experience, especially for services with a large and diverse user base.

However, we argue that private users have no more than 10,000 contacts in their address book (Signal states similar numbers [50] and Google's contact management service limits the maximum number to 25,000 [37]). Therefore, the contact discovery service should not allow syncing more numbers than in this order of magnitude at any point in time. Exceptions could be made for businesses, non-profit organizations, or celebrities after performing extended validation.

We furthermore argue that private users do not change many of their contacts frequently. The operators of Writethat.name observed that even professional users have only about 250 new contacts per year [105]. Therefore, service providers could penalize users when detecting frequent contact changes. Additional total limits for the number of contacts can detect accounts crawling at slow rates.

Facebook (WhatsApp's parent company) informed us during responsible disclosure that they see legitimate use cases where users synchronize more contacts (e.g., enterprises with 200,000 contacts)[16]. We recommend to handle such business customers differently than private users. In response to our findings showing that data scraping is currently possible even at a country-level scale (cf. § 4), Facebook informed us that they have improved WhatsApp's contact synchronization feature to detect such attacks much earlier (cf. § 9).

**Limiting Exposure of Sensitive Information.** Since preventing enumeration attacks entirely is impossible, the information collected about users through this process should be kept minimal.

[16]This definition of "legitimate" is interesting, since WhatsApp's terms of service prohibit *non-personal* use of their services [103].

While Signal behaves exemplarily and by default reveals no public profile pictures or status information, WhatsApp and Telegram should set corresponding default settings or at least limit access to user data to contacts of the user by default. Furthermore, users may take actions to protect themselves from exposure of private information by thinking carefully what information to include into public fields, such as profile pictures and status text, and checking whether there are privacy settings that can limit the visibility of this information. Additionally, on-device machine learning techniques could be applied to automatically educate users about the sensitivity of shared content, e.g., when extended nudity or children are detected in uploaded profile pictures. However, we point out that such a form of client-side content scanning may cause severe controversy as recently seen with Apple's announcement of expanded protections for children in iMessage [31].

**Mutual Contacts.** Mobile messengers could offer a setting for users to let them only be discovered on the service by contacts in their address book to prevent third parties from obtaining any information about them. Combining such a setting with a privacy-preserving contact discovery technique like PSI poses an interesting challenge.

**Temporary Banning.** Detections of possible abuse of the contact discovery API can be met with a temporary ban of the feature for the account. Disabling only specific features is preferable to a complete ban due to the possibility of false positives. By communicating the reason for the time-out to the users, they can adapt their behavior and still use other features. An exponentially increasing time-out duration can block repeated offenders.

**Shadow Banning.** Another possibility is to return randomized information once the rate limits are exceeded for an account. While this can poison attacker data and delay enumeration attacks, it is relatively easy for attackers to detect shadow banning by including identifiers with known states into the requests. Once the rate limits are known to the attacker, this strategy looses effectiveness.

**CAPTCHAs.** In countless web applications, CAPTCHAs are in place to prevent automated API abuse. Even though there are ways to circumvent CAPTCHAs, including optical character recognition (OCR) software, machine learning techniques [109], or simply paying other humans for solutions [67], they still can significantly slow down an attack or at least increase the cost of abuse. Compared to temporary or shadow banning, CAPTCHAs have the advantage of being a low barrier for falsely flagged human users to regain their accounts without the necessity to wait for extended periods of time or to contact a support team.

**Account Verification.** The scale at which enumeration attacks can be conducted depends mainly on two factors: the rate limits determining the crawling rate for individual accounts and the number of accounts accessible to the attacker. Many enumeration attacks, including the ones presented here (cf. § 4), are only scalable if attackers can register many accounts without significant effort. Extended verification of accounts upon registration (e.g., through identity checks) can make it more difficult for attackers to launch large-scale attacks. To not decrease usability for regular users, account verification could be restricted to individuals with unusual, yet benign, usage patterns, such as celebrities, journalists, or politicians. Additional scrutiny should be associated with these accounts to prevent abuse of these elevated privileges. Of course, service providers would face significant costs to operate a reliable verification infrastructure.

## 8 RELATED WORK

We review related works on PSI protocols, enumeration attacks, user tracking, and hash reversal.

### 8.1 Private Set Intersection (PSI)

PSI protocols can be used for mobile private contact discovery to hinder hash reversal attacks (cf. § 3). Most PSI protocols consider a scenario where the input sets of both parties have roughly the same size (e.g., [57, 75–79]). However, in contact discovery, the provider has orders of magnitude more

entries in the server database than users have contacts in their address book. Thus, there has been research on *unbalanced* PSI protocols, where the input set of one party is much larger than the other [13, 14, 25, 50, 55].

Today's best known protocols [50] also provide efficient implementations with reasonable runtimes on modern smartphones. Unfortunately, their limitation is the amount of data that needs to be transferred to the client in order to obtain an encrypted representation of the server's database: for $2^{28}$ registered users (the estimated number of active users on Telegram [20]) it is necessary to transfer ≈1 GiB, for $2^{31}$ registered users (a bit more than the estimated number of users on WhatsApp [20]) even ≈8 GiB are necessary. Moreover, even PSI protocols cannot prevent enumeration attacks, as discussed in § 7.1.

The Signal developers concluded that current PSI protocols are not practical for deployment [64], and also argue that the required non-collusion assumption for more efficient solutions with multiple servers [50] is unrealistic. Instead, they introduced a beta version [64] that utilizes Intel Software Guard Extensions (SGX) for securely performing contact discovery in a trusted execution environment. However, Intel SGX provides no provable security guarantees and there have been many severe attacks (most notably "Foreshadow" [10]). Given the scope of such attacks and that fixes often require hardware changes, the Intel SGX-based contact discovery service is less secure than cryptographic PSI protocols.

### 8.2 Enumeration Attacks

Popular applications for enumeration attacks include, e.g., finding vulnerable devices by scanning all IPv4 addresses and ports. In the following, we focus on such attacks on social networks and mobile messengers.

For 8 popular social networks, Balduzzi et al. [5] fed about 10 million cleverly generated email addresses into the search interface, allowing them to identify 1.2 million user profiles without experiencing any countermeasures. After crawling these profiles with methods similar to [6], they correlated the profiles from different networks to obtain a combined profile that in many cases contained friend lists, location information, and sexual preferences. Upon the responsible disclosure of their findings, Facebook and XING quickly established reasonable rate limits for search queries. We hope for similar deployment of countermeasures by responsively disclosing our findings on mobile messengers (cf. § 9).

Schrittwieser et al. [68, 88] were the first to investigate enumeration attacks on mobile messengers, including WhatsApp. For the area code of San Diego, they automatically tested 10 million numbers within 2.5 hours without noticing severe limitations. Since then, service providers established at least some countermeasures. We revisit enumeration attacks at a substantially larger scale (cf. § 4) and demonstrate that the currently deployed countermeasures are insufficient to prevent large-scale enumeration attacks.

For the Korean messenger KakaoTalk, enumeration attacks were demonstrated in [52, 53]. The authors automatically collected ≈50,000 user profiles by enumerating 100,000 number sequences that could potentially be phone numbers. They discovered how to obtain the user names associated with these profiles and found that ≈70 % of users chose their real name (or a name that could be a real name), allowing identification of many users. As countermeasures, the authors propose the detection of known misuse patterns and anomaly detection for repeated queries. In contrast, in § 4 we automatically perform enumeration attacks at a much larger scale on popular messengers used world-wide. By testing only valid mobile phone numbers, we increase the efficiency of our attacks. We propose further mitigations in § 7.

In [15], the authors describe an Android-based system to automatically conduct enumeration attacks for different mobile messengers by triggering and recording API calls via the debug bridge.

In their evaluation, they enumerate 100,000 Chinese numbers for WeChat and correlate the results with other messengers. We perform evaluations of different messengers at a larger scale, also assessing currently deployed countermeasures against enumeration attacks (cf. § 4).

Gupta et al. [38, 39] obtained personal information from reverse-lookup services, which they correlated with public profiles on social networks like Facebook, in order to then run personalized phishing attacks on messengers like WhatsApp. From about 1 million enumerated Indian numbers, they were able to target about 250,000 users across different platforms.

Enumeration attacks were also used to automatically harvest Facebook profiles associated with phone numbers even when the numbers are hidden in the profiles [54]. The authors experienced rather strict countermeasures that limit the number of possible queries to 300 before a "security check" in form of a CAPTCHA is triggered. By automatically creating many fake accounts and setting appropriately slow crawling rates, it was still possible to test around 200,000 Californian and Korean phone numbers within 15 days, leading to a success rate of 12 % and 25 %, respectively. While acquiring phone numbers is more cumbersome than generating email addresses, we nevertheless report much faster enumeration attacks that harvest profiles of mobile messenger users (cf. § 4).

Recently, a data set containing personal data of more than 530 million Facebook users was found online, including phone numbers and email addresses [45]. Facebook justified that there has been no hack of their systems, rather malicious actors misused the "contact importer" feature (which behaves similar to contact discovery in WhatsApp) for data scraping [19]. Interestingly, the issue of insufficient protection for the contact importer API was known to Facebook prior to September 2019, whereas almost identical issues in WhatsApp were only addressed after our responsible disclosure in September 2019 (cf. § 9). Similar sized leaks of scraped data have recently been reported for LinkedIn [11] and Clubhouse [12]. Such publicly available data sets containing active phone numbers could be utilized to further optimize the success rate when crawling contact discovery services (cf. § 4.8).

In 2017, Loran Kloeze developed the Chrome extension "WhatsAllApp" that allows to misuse WhatsApp's web interface for enumeration attacks and collecting profile pictures, display names, and status information [56]. After disclosing his approach, Facebook pointed out (non-default) privacy settings available to the user to hide this information, and stated that WhatsApp detects abuse based on measures that identify and block data scraping [27]. In § 4, we investigate the effectiveness of their measures and find that we can perform attacks at a country-level scale, even with few resources. We also observe that few users change the default settings.

There exist other open-source projects that enable automated crawling of WhatsApp users and extracting personal information, e.g., [34, 80]. However, frequent changes of the WhatsApp API and code often break these tools, which are mostly abandoned after some time, or cease operation after receiving legal threats [47].

### 8.3 User Tracking

In 2014, Buchenscheit et al. [9] conducted a user study where they tracked online status of participants for one month, which allowed them to infer much about the participants' daily routines and conversations (w.r.t. duration and chat partners). Other user studies report the "Last Seen" feature as the users' biggest privacy concern in WhatsApp [17, 85].

Researchers also monitored the online status of 1,000 randomly selected users from different countries for 9 months [90]. They published statistics on the observed behavior w.r.t. the average usage time per day and the usage throughout the day. Despite the clearly anomalous usage patterns of the monitoring, the authors did not experience any countermeasures.

"WhatsSpy" is an open-source tool that monitors the online status, profile pictures, and status messages of selected numbers—provided the default privacy options are set [110]. It abuses the fact

that WhatsApp indicates when a user is online [111], even if the "Last Seen" feature is disabled. The tool was discontinued in 2016 to prevent low-level abuse [112] as the developer found more than 45,000 active installations and companies using the prototype commercially.

In this context, our user database crawling attacks could be used to efficiently find new users to track and our discovery of Telegram's `importer_count` label gives even more monitoring possibilities (cf. § 4).

### 8.4 Hash Reversal

Reversing hashes is mostly used for "recovering" passwords, which are commonly stored only in hashed form. Various hash reversal tools exist, either relying on brute-forcing [74, 93] or rainbow tables [84]. The practice of adding a unique salt to each hash makes reversal hard at a large scale, but is not suitable for contact discovery [50, 63]. In contrast, our mitigation proposed in § 7.1 uses a *global* salt.

It is well known that hashing of personally identifiable information (PII), including phone numbers, is not sufficient due to the small pre-image space [28, 63]. The PSI literature therefore has proposed many secure alternatives for matching PII, which are currently orders of magnitudes slower than insecure hashing-based protocols (cf. § 8.1).

In [65], the authors show that the specific structure of PII makes attacks much easier in practice. Regarding phone numbers, they give an upper bound of 811 trillion possible numbers world-wide, for which brute-forcing takes around 11 days assuming SHA-256 hashes and a hash rate of 844 MH/s. For specific countries, they also run experiments showing that reversing an MD5 or SHA-256 hash for a German phone number takes at most 2.5 hours. In § 2, we give much more accurate estimations for the amount of possible (mobile) phone numbers and show in § 3 that using novel techniques and optimizations, hash reversal is much faster and can even be performed on-the-fly.

## 9 CONCLUSION

Mobile contact discovery is a challenging topic for privacy researchers in many aspects. In this paper, we took an attacker's perspective and scrutinized currently deployed contact discovery services of three popular mobile messengers: WhatsApp, Signal, and Telegram. We revisited known attacks and using novel techniques we quantified the efforts required for curious service providers and malicious users to collect sensitive user data at a large scale. Shockingly, we were able to demonstrate that even resource-constrained attackers are able to collect data of billions of users, which can be abused for various purposes. While we proposed several technical mitigations for service providers to prevent such attacks in the future, currently the most effective protection measure for users is to revise the existing privacy settings. Thus, we advocate to raise awareness among regular users about the seriousness of privacy issues in mobile messengers and educate them about the precautions they can take right now.

**Responsible Disclosure.** In our paper, we demonstrate methods that allow to invade the privacy of billions of mobile messenger users by using only very few resources. We therefore initiated the official responsible disclosure process with all messengers we investigated (WhatsApp, Signal, and Telegram) before the paper submission and shared our findings to prevent exploitation by maleficent imitators.

Signal acknowledged the issue of enumeration attacks as not fully preventable, yet nevertheless adjusted their rate limits in the weeks following our disclosure and implemented further defenses against crawling. Facebook acknowledged and rewarded our findings as part of their bug bounty program, and has deployed improved defenses for WhatsApp's contact synchronization. Telegram responded to our responsible disclosure by elaborating on additional data scraping countermeasures beyond the rate limits detected by us. They are allegedly triggered when attackers use existing

databases of active phone numbers and higher conversion rates than ours occur. In such cases, contact discovery is stopped after 20 to 100 matches, instead of 5,000 as measured by us.

**Update for 2022.** Our study of mobile messengers, including large-scale crawling attacks to determine rate limits as described in § 4, was mainly conducted in fall 2019. We briefly outline how the situation, also in response to our responsible disclosure, changed till 2022. WhatsApp did not disclose to us the changes they implemented in response to our disclosure. Our analysis suggests that the limits for daily contact changes have been significantly reduced (but are still in the 4-digit range), while a single larger request directly after the registration of an account is possible. Signal significantly lowered the amount contacts each client can synchronize while implementing a routine to allow for uninterrupted use for clients with a very large number of contacts. However, we were asked to not publish details to not encourage low-level abuse. Furthermore, Signal's hashing-based API was discontinued and recent client code changes hint at a new contact discovery service based on HSMs being introduced in the near future as an alternative to the currently used Intel SGX enclave. Nevertheless, we point out that vulnerable hashing-based protocols are still widely used in 2022 in other messengers (e.g., Wickr Me and Wire) as surveyed in [50] and shown in Tab. 1. There appear to be no changes to Telegram's behavior with regard to mobile contact discovery.

**Ethical Considerations.** The experiments in this work were conducted in coordination with the ethical and legal departments of our institution. Special care was taken to ensure the privacy of the affected users, as detailed in § 4.4.

## ACKNOWLEDGMENTS

We thank the anonymous reviewers for their constructive feedback. Furthermore, we thank Robin Hundt, Lukas Nothelfer, Florian Plesker, Oliver Schick, and Sebastian Schindler for their invaluable help with the implementation of our attacks.

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 850990 PSOTI). It was supported 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", and by the BMBF and HMWK within ATHENE.

## REFERENCES

[1] Affinityclick. 2013 (accessed May 1, 2022). Hushed - Private Phone Numbers, Talk and Text. https://hushed.com/
[2] Parry Aftab. 2014. Findings under the Personal Information Protection and Electronic Documents Act (PIPEDA). https://parryaftab.blogspot.com/2014/03/what-does-whatsapp-collect-that.html
[3] Martin Albrecht, Lenka Mareková, Kenneth Paterson, and Igors Stepanovs. 2022. Four Attacks and a Proof for Telegram. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE.
[4] Backes SRT. 2013 (accessed March 30, 2021). WhatsBox - GDPR Compliant WhatsApp. https://www.backes-srt.com/en/solutions-2/whatsbox/
[5] Marco Balduzzi, Christian Platzer, Thorsten Holz, Engin Kirda, Davide Balzarotti, and Christopher Kruegel. 2010. Abusing Social Networks for Automated User Profiling. In *Recent Advances in Intrusion Detection (RAID)*. Springer, 422–441.
[6] Leyla Bilge, Thorsten Strufe, Davide Balzarotti, and Engin Kirda. 2009. All Your Contacts Are Belong to Us: Automated Identity Theft Attacks on Social Networks. In *International Conference on World Wide Web (WWW)*. ACM, 551–560.
[7] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. 2016. Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications. In *IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 292–302.
[8] BitWeasil. 2012 (accessed May 1, 2022). Cryptohaze. http://www.cryptohaze.com
[9] Andreas Buchenscheit, Bastian Könings, Andreas Neubert, Florian Schaub, Matthias Schneider, and Frank Kargl. 2014. Privacy Implications of Presence Sharing in Mobile Messaging Applications. In *International Conference on Mobile and Ubiquitous Multimedia*. ACM, 20–29.

[10] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *USENIX Security Symposium*. USENIX Association, 991–1008.
[11] Katie Canales. 2021. Hackers Scraped Data from 500 Million LinkedIn Users. https://www.businessinsider.com/linkedin-data-scraped-500-million-users-for-sale-online-2021-4
[12] Katie Canales. 2021. Scraped Personal Data of 1.3 Million Clubhouse Users has Reportedly Leaked Online. https://www.businessinsider.com/clubhouse-data-leak-1-million-users-2021-4
[13] Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. 2018. Labeled PSI from Fully Homomorphic Encryption with Malicious Security. In *ACM Conference on Computer and Communications Security (CCS)*. ACM, 1223–1237.
[14] Hao Chen, Kim Laine, and Peter Rindal. 2017. Fast Private Set Intersection from Homomorphic Encryption. In *ACM Conference on Computer and Communications Security (CCS)*. ACM, 1243–1255.
[15] Yao Cheng, Lingyun Ying, Sibei Jiao, Purui Su, and Dengguo Feng. 2013. Bind Your Phone Number with Caution: Automated User Profiling Through Address Book Matching on Smartphone. In *ACM ASIA Conference on Computer and Communications Security (ASIACCS)*. ACM, 335–340.
[16] Howard Chu. 2015 (accessed May 1, 2022). LMDB Website. Available: http://www.lmdb.tech/doc/.
[17] Karen Church and Rodrigo de Oliveira. 2013. What's Up with WhatsApp? Comparing Mobile Instant Messaging Behaviors with Traditional SMS. In *Human-Computer Interaction with Mobile Devices and Services (MobileHCI)*. ACM, 352–361.
[18] Catalin Cimpanu. 2019. Hong Kong Protesters Warn of Telegram Feature that can Disclose Their Identities. https://www.zdnet.com/article/hong-kong-protesters-warn-of-telegram-feature-that-can-disclose-their-identities/
[19] Mike Clark. 2021. The Facts on News Reports About Facebook Data. https://about.fb.com/news/2021/04/facts-on-news-reports-about-facebook-data/
[20] J Clement. 2019. Most Popular Global Mobile Messenger Apps. https://www.statista.com/statistics/258749/most-popular-global-mobile-messenger-apps
[21] J Clement. 2019. Most Popular Mobile Messaging Apps in the United States as of June 2019. https://www.statista.com/statistics/350461/mobile-messenger-app-usage-usa/
[22] J Clement. 2019. Number of WhatsApp Users in the United States from 2019 to 2023. https://www.statista.com/statistics/558290/number-of-whatsapp-users-usa/
[23] Douglas Comer. 1979. Ubiquitous B-Tree. *Comput. Surveys* 11, 2 (June 1979), 121–137.
[24] Confide, Inc. 2022. Confide Privacy Policy. https://getconfide.com/privacy
[25] Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. 2021. Labeled PSI from Homomorphic Encryption with Reduced Computation and Communication. In *ACM Conference on Computer and Communications Security (CCS)*. ACM, 1135–1150.
[26] Josh Constine. 2018. WhatsApp Hits 1.5 Billion Monthly Users. $19B? Not so Bad. https://techcrunch.com/2018/01/31/whatsapp-hits-1-5-billion-monthly-users-19b-not-so-bad/
[27] Joseph Cox. 2017. Building a Database of WhatsApp Users Can Be Pretty Easy. https://www.vice.com/en/article/wnw4vw/building-a-database-of-whatsapp-users-can-be-pretty-easy
[28] Levent Demir, Amrit Kumar, Mathieu Cunche, and Cédric Lauradoux. 2018. The Pitfalls of Hashing for Privacy. *IEEE Communications Surveys and Tutorials* 20, 1 (2018), 551–565.
[29] Deutsche Welle. 2019. New EU Data Law Forces Firms to Ban WhatsApp, Snapchat from Phones. https://www.dw.com/en/new-eu-data-law-forces-firms-to-ban-whatsapp-snapchat-from-phones/a-44076861
[30] Zak Doffman. 2019. New WhatsApp Threat Confirmed: Android And iOS Users At Risk From Malicious Video Files. https://www.forbes.com/sites/zakdoffman/2019/11/16/new-whatsapp-threat-confirmed-android-and-ios-users-at-risk-from-malicious-video-files/
[31] Zak Doffman. 2021. Apple's iMessage Safety Update Is A Major Change For iPhone Privacy. https://www.forbes.com/sites/zakdoffman/2021/11/13/apples-billion-iphone-users-shock-imessage-update-after-security-warnings/
[32] Meredith Dost and Kyley McGeeney. 2016. Moving Without Changing Your Cellphone Number: A Predicament for Pollsters. https://www.pewresearch.org/methods/2016/08/01/moving-without-changing-your-cellphone-number-a-predicament-for-pollsters/
[33] Pavel Durov. 2020. 400 Million Users, 20,000 Stickers, Quizzes 2.0 and 400K EUR for Creators of Educational Tests. https://telegram.org/blog/400-million
[34] Jose Estrada. 2018 (accessed May 1, 2022). WhatsApp Scraping. https://github.com/JMGama/WhatsApp-Scraping
[35] Facebook, Inc. 2020. Two Billion Users — Connecting the World Privately. https://about.fb.com/news/2020/02/two-billion-users/
[36] Google. 2010 (accessed May 1, 2022). Google's Common Java, C++ and JavaScript Library for Parsing, Formatting, and Validating International Phone Numbers. https://github.com/google/libphonenumber

[37] Google. 2022 (accessed May 1, 2022). I'm Getting a Contacts Error - Contacts Help. https://support.google.com/contacts/answer/148779
[38] Srishti Gupta. 2016. Emerging Threats Abusing Phone Numbers Exploiting Cross-Platform Features. In *International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*. IEEE, 1339–1341.
[39] Srishti Gupta, Payas Gupta, Mustaque Ahamad, and Ponnurangam Kumaraguru. 2016. Exploiting Phone Numbers and Cross-Application Features in Targeted Mobile Attacks. In *Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM@CCS)*. ACM, 73–82.
[40] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. 2021. All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers. In *Network & Distributed System Security Symposium (NDSS)*. Internet Society.
[41] George Hatzivasilis. 2017. Password-Hashing Status. *Cryptography* 1, 2 (2017), 10.
[42] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. 2021. AirCollect: Efficiently Recovering Hashed Phone Numbers Leaked via Apple AirDrop. In *WISEC*. ACM, 371–373. https://ia.cr/2021/893
[43] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. 2021. PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop. In *USENIX Security Symposium*. USENIX Association, 3577–3594. https://ia.cr/2021/481
[44] Martin Hellman. 1980. A Cryptanalytic Time-Memory Trade-Off. *Transactions on Information Theory* 26, 4 (1980), 401–406.
[45] Aaron Holmes. 2021. 533 Million Facebook Users' Phone Numbers and Personal Data have been Leaked Online. https://www.businessinsider.com/stolen-data-of-533-million-facebook-users-leaked-online-2021-4
[46] Hang Hu, Peng Peng, and Gang Wang. 2019. Characterizing Pixel Tracking through the Lens of Disposable Email Services. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 365–379.
[47] Ali Hubail. 2015 (accessed May 1, 2022). Interface to WhatsApp Messenger—Fed up with the F**king Legal Threats. https://github.com/venomous0x/WhatsAPI
[48] inAudible-NG. 2017 (accessed May 1, 2022). RainbowCrack-NG: Free and Open-Source Software to Generate and Use Rainbow Tables. https://github.com/inAudible-NG/RainbowCrack-NG
[49] ITU Telecommunication Standardization Sector. 2022 (accessed May 1, 2022). National Numbering Plans. https://www.itu.int/oth/T0202.aspx?parent=T0202
[50] Daniel Kales, Christian Rechberger, Matthias Senker, Thomas Schneider, and Christian Weinert. 2019. Mobile Private Contact Discovery at Scale. In *USENIX Security Symposium*. USENIX Association, 1447–1464. https://ia.cr/2019/517
[51] Samantha Murphy Kelly. 2021. Yes, You Are Getting Lots of Robocalls Again. https://edition.cnn.com/2021/03/04/tech/robocalls-pre-pandemic-levels/index.html
[52] Eunhyun Kim, Kyungwon Park, Hyoungshick Kim, and Jaeseung Song. 2014. I've Got Your Number: - Harvesting Users' Personal Data via Contacts Sync for the KakaoTalk Messenger. In *Workshop on Information Security Applications (WISA)*. Springer, 55–67.
[53] Eunhyun Kim, Kyungwon Park, Hyoungshick Kim, and Jaeseung Song. 2015. Design and Analysis of Enumeration Attacks on Finding Friends with Phone Numbers: A Case Study with KakaoTalk. *Computers & Security* 52 (2015), 267–275.
[54] Jinwoo Kim, Kuyju Kim, Junsung Cho, Hyoungshick Kim, and Sebastian Schrittwieser. 2017. Hello, Facebook! Here Is the Stalkers' Paradise!: Design and Analysis of Enumeration Attack Using Phone Numbers on Facebook. In *Information Security Practice and Experience*. Springer, 663–677.
[55] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. 2017. Private Set Intersection for Unequal Set Sizes with Mobile Applications. *Proceedings on Privacy Enhancing Technologies (PoPETs)* 2017, 4 (2017), 177–197.
[56] Loran Kloeze. 2017. Collecting Huge Amounts of Data with WhatsApp. https://www.lorankloeze.nl/2017/05/07/collecting-huge-amounts-of-data-with-whatsapp/
[57] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient Batched Oblivious PRF with Applications to Private Set Intersection. In *ACM Conference on Computer and Communications Security (CCS)*. ACM, 818–829.
[58] James M. Lepkowski. 2011. Telephone Sampling: Frames and Selection Techniques. In *International Encyclopedia of Statistical Science*. Springer, 1585–1586.
[59] Joshua Lund. 2017. Encrypted Profiles for Signal Now in Public Beta. https://signal.org/blog/signal-profiles-beta/
[60] Joshua Lund. 2018. Technology Preview: Sealed Sender for Signal. https://signal.org/blog/sealed-sender/
[61] Joshua Lund. 2019. Signal-Server. https://github.com/signalapp/Signal-Server
[62] Joshua Lund. 2019. Technology Preview for Secure Value Recovery. https://signal.org/blog/secure-value-recovery/
[63] Moxie Marlinspike. 2014. The Difficulty Of Private Contact Discovery. https://signal.org/blog/contact-discovery/

[64] Moxie Marlinspike. 2017. Technology Preview: Private Contact Discovery for Signal. https://signal.org/blog/private-contact-discovery
[65] Matthias Marx, Ephraim Zimmer, Tobias Mueller, Maximilian Blochberger, and Hannes Federrath. 2018. Hashing of Personally Identifiable Information is Not Sufficient. In *Sicherheit*. Gesellschaft für Informatik e.V., 55–68.
[66] Signal Messenger. 2020. Introducing Signal PINs. https://signal.org/blog/signal-pins/
[67] Marti Motoyama, Kirill Levchenko, Chris Kanich, Damon McCoy, Geoffrey M. Voelker, and Stefan Savage. 2010. Re: CAPTCHAs-Understanding CAPTCHA-Solving Services in an Economic Context. In *USENIX Security Symposium*. USENIX Association, 435–462. http://www.usenix.org/events/sec10/tech/full_papers/Motoyama.pdf
[68] Robin Mueller, Sebastian Schrittwieser, Peter Frühwirt, Peter Kieseberg, and Edgar R. Weippl. 2014. What's New with WhatsApp & Co.? Revisiting the Security of Smartphone Messaging Applications. In *Information Integration and Web-based Applications & Services*. ACM, 142–151.
[69] Adrian Mönnich. 2010 (accessed May 1, 2022). Flask. https://palletsprojects.com/p/flask
[70] Philippe Oechslin. 2003. Making a Faster Cryptanalytic Time-Memory Trade-Off. In *CRYPTO*. Springer, 617–630.
[71] Official Journal of the European Union. 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council. https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN
[72] OpenMP. 2022 (accessed May 1, 2022). The OpenMP API Specification for Parallel Programming. https://www.openmp.org
[73] OpenSSL Software Foundation. 2022 (accessed May 1, 2022). OpenSSL: Cryptography and SSL/TLS Toolkit. https://www.openssl.org
[74] Openwall. 2022 (accessed May 1, 2022). John the Ripper Password Cracker. https://www.openwall.com/john/
[75] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2019. SpOT-Light: Lightweight Private Set Intersection from Sparse OT Extension. In *CRYPTO*. Springer, 401–431.
[76] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security Symposium*. USENIX Association, 515–530.
[77] Benny Pinkas, Thomas Schneider, Christian Weinert, and Udi Wieder. 2018. Efficient Circuit-Based PSI via Cuckoo Hashing. In *EUROCRYPT*. Springer, 125–157. https://ia.cr/2018/120
[78] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster Private Set Intersection Based on OT Extension. In *USENIX Security Symposium*. USENIX Association, 797–812.
[79] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable Private Set Intersection Based on OT Extension. *Transactions on Privacy and Security (TOPS)* 21, 2 (2018), 7:1–7:35.
[80] Sebin PJ. 2017 (accessed May 1, 2022). WhatsApp Crawler. https://gitlab.com/jishnutp/whatsapp-crawler
[81] Jon Porter. 2020. Signal Becomes European Commission's Messaging App of Choice in Security Clampdown. https://www.theverge.com/2020/2/24/21150918/european-commission-signal-encrypted-messaging
[82] Niels Provos and David Mazières. 1999. A Future-Adaptable Password Scheme. In *USENIX Annual Technical Conference (ATC)*. USENIX Association, 81–91.
[83] RainbowCrack Project. 2022 (accessed May 1, 2022). List of Rainbow Tables. http://project-rainbowcrack.com/table.htm
[84] RainbowCrack Project. 2022 (accessed May 1, 2022). RainbowCrack. http://project-rainbowcrack.com/
[85] Yasmeen Rashidi, Kami Vaniea, and L. Jean Camp. 2016. Understanding Saudis' Privacy Concerns When Using WhatsApp. In *Workshop on Usable Security (USEC)*. Internet Society.
[86] Salvatore Sanfilippo. 2022 (accessed May 1, 2022). Redis Commands - GET. https://redis.io/commands/get
[87] Salvatore Sanfilippo. 2022 (accessed May 1, 2022). Redis Website. https://redis.io/
[88] Sebastian Schrittwieser, Peter Frühwirt, Peter Kieseberg, Manuel Leithner, Martin Mulazzani, Markus Huber, and Edgar R. Weippl. 2012. Guess Who's Texting You? Evaluating the Security of Smartphone Messaging Applications. In *Network & Distributed System Security Symposium (NDSS)*. Internet Society.
[89] Scattered Secrets. 2020. Bcrypt Password Cracking Extremely Slow? Not If You Are Using Hundreds of FPGAs! https://scatteredsecrets.medium.com/bcrypt-password-cracking-extremely-slow-not-if-you-are-using-hundreds-of-fpgas-7ae42e3272f6
[90] Security Research Group FAU Erlangen-Nürnberg. 2014 (accessed May 1, 2022). Online Status Monitor. https://onlinestatusmonitor.com/
[91] Signal. 2022 (accessed May 1, 2022). Signal Homepage. https://signal.org
[92] Mehul Srivastava. 2019. WhatsApp Voice Calls Used to Inject Israeli Spyware on Phones. https://www.ft.com/content/4da1117e-756c-11e9-be7d-6d846537acab
[93] Jens Steube and Gabriele Gristina. 2022 (accessed May 1, 2022). hashcat - World's Fastest and Most Advanced Password Recovery Utility. https://hashcat.net/
[94] Telegram. 2020 (accessed May 1, 2022). Telegram FAQ: How Secure is Telegram? https://telegram.org/faq#q-how-secure-is-telegram

[95] Telegram. 2022 (accessed May 1, 2022). TDLib: importedContacts Class Reference. https://core.telegram.org/tdlib/docs/classtd_1_1td__api_1_1imported_contacts.html
[96] Telegram. 2022 (accessed May 1, 2022). Telegram Database Library. https://core.telegram.org/tdlib
[97] Tom Slack. 2019. Is WhatsApp in Breach of the GDPR? A Lawyer's View. https://guild.co/blog/is-whatsapp-in-breach-of-the-gdpr-a-lawyers-view/
[98] Huahong Tu, Adam Doupé, Ziming Zhao, and Gail-Joon Ahn. 2019. Users Really Do Answer Telephone Scams. In *USENIX Security Symposium*. USENIX Association, 1327–1340.
[99] William Turton. 2016. Why You Should Stop Using Telegram Right Now. https://gizmodo.com/why-you-should-stop-using-telegram-right-now-1782557415
[100] Lisa Vaas. 2019. Robocalls Now Flooding US Phones with 200m Calls per Day. https://nakedsecurity.sophos.com/2019/09/17/robocalls-now-flooding-us-phones-with-200m-calls-per-day/
[101] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 137–152.
[102] WhatsApp LLC. 2022 (accessed May 1, 2022). About Contact Upload. https://faq.whatsapp.com/general/contacts/about-contact-upload
[103] WhatsApp LLC. 2022 (accessed May 1, 2022). WhatsApp Legal Info. https://www.whatsapp.com/legal?eea=0#terms-of-service
[104] Gilbert Wondracek, Thorsten Holz, Engin Kirda, and Christopher Kruegel. 2010. A Practical Attack to De-anonymize Social Network Users. In *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 223–238.
[105] WriteThat.Name. 2013 (accessed March 30, 2021). Your Address Book Automagically Updated. http://writethat.name/
[106] x0rz. 2018. A Look Into Signal's Encrypted Profiles. https://blog.0day.rocks/a-look-into-signals-encrypted-profiles-5491908186c1
[107] Maria Xynou and Arturo Filastò. 2021. How Countries Attempt to Block Signal Private Messenger App Around the World. https://ooni.org/post/2021-how-signal-private-messenger-blocked-around-the-world/
[108] Liliya Yapparova and Alexey Kovalev. 2019. Comrade Major. https://meduza.io/en/feature/2019/08/11/comrade-major
[109] Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. 2018. Yet Another Text CAPTCHA Solver: A Generative Adversarial Network Based Approach. In *ACM Conference on Computer and Communications Security (CCS)*. ACM, 332–348.
[110] Maikel Zweerink. 2015. WhatsApp Privacy is Broken! https://maikel.pro/blog/en-whatsapp-privacy-options-are-illusions/
[111] Maikel Zweerink. 2015. WhatsApp Privacy Problem Explained in Detail. https://maikel.pro/blog/en-whatsapp-privacy-problem-explained-in-detail/
[112] Maikel Zweerink. 2016. PoC WhatsSpy Public Support Ending Today. https://maikel.pro/blog/whatsspy-public-support-ending-today