Mutual Contact Discovery*

Jaap-Henk Hoepman
Radboud University Nijmegen
Karlstad University
University of Groningen
(jhh@cs.ru.nl)

May 11, 2023

Abstract

Messaging services allow new users to find existing contacts that already use that service through a process called contact discovery. Existing users are similarly informed of new users that are already on their contact list. This creates a privacy issue: when you join and enable contact discovery, anyone already on the service that has your number on their contact list gets notified that you joined. Even if you don't know that person, or if it is an ex or former colleague that you long parted with and whose contact details you deleted long ago. To solve this, we propose a *mutual* contact discovery protocol, that only allow users to discover each other when both are (still) in each other's contact list. Mutual contact discovery has the additional advantage that it can be implemented in a more privacy friendly fashion (e.g., protecting the social graph from the server) than traditional, one-sided contact discovery, without necessarily relying on trusted hardware.

1 Introduction

Many messaging services like WhatsApp[1] or Signal[2] are based on the observation that "access to an existing social graph makes building social apps much easier" [28], and thus use an existing set of globally unique identifiers (typically mobile phone numbers) to identify their users. This allows such services to notify their users when any of their existing contacts join the service. To implement this *contact discovery* process, users need to allow the messaging service to access the contact list or address book they maintain locally on their smartphone

*Previous versions of this paper overstate the properties of the protocol and contain an erroneous proof.

Version: Thu May 11 14:50:39 2023 +0200 / arxiv-v3 / mutual-contact-discovery.tex

[1] https://www.whatsapp.com

[2] https://signal.org

(or other device), so that the service can match any entry on this list with the list of new users that recently joined. Contact discovery creates privacy issues: a naive implementation (where a user asks the service provider whether any of the phone numbers on their contact list happen to be subscribed as well) allows the service provider to learn the mobile phone number of all contacts of all its users, even when these contacts are not member of the service at all [16]. This is in particular a concern for people that shield their mobile phone number for privacy reasons. As a result, messaging services like Signal try to make the discovery process more privacy friendly by limiting the amount of information they learn [27]. As we will discuss in section 2, the protocols currently in use are somewhat limited in the protection they offer, are not very efficient in practice, or rely on non-standard assumptions.

When users switch to a supposedly privacy friendly messaging app like Signal, they are surprised to find out that contact discovery is *one-sided*. When you enable contact discovery, anyone already on Signal that has your number on their contact list (like a former colleague, former patient, or an ex that you long parted with but has kept your phone number as a contact) gets notified that you joined.[3] Privacy conscious people expect a somewhat different behaviour: people should only be allowed to discover that someone else is on a messaging service when *both of them* have the phone number of the other person on their contact list. We call this *mutual contact discovery*. In mutual contact discovery, the fact that you have someone else's number in your contact list is only a signal that you allow this person to discover you. Knowing someone's number and adding it to your contact list does *not* give you immediate permission to discover the other person: this only happens if that person also has *your* number on *their* contact list (and when both of you have contact discovery enabled, of course).

Note that more fine-grained approaches for mutual contact discovery are very important in practice. For example, users that have a phone number in their contact list to recognise a harasser calling, may want to have the option to disallow such numbers from discovering them. Conversely, people deleting contacts from their contact list may not want the deleted contact to be actively notified about this. We leave these refinements for further study, but note that such options can easily be added to the protocols discussed in this paper.

In this paper we study privacy friendly protocols for mutual contact discovery. We first discuss existing schemes for (one-sided) contact discovery and related problems like private matchmaking and private set intersection in section 2, and show how these do not satisfactorily solve the problem at hand. We then set the stage in section 3, describing the system model, threat model, security and privacy requirements, and primitives used. After that we present our main protocol for mutual contact discovery in section 4. This protocol offers stronger privacy guarantees in a more adversarial model than typically as-

[3] See e.g., https://twitter.com/DorotheaBaur/status/1349340273357291528 .

sumed: only mutual contacts can discover each other, and the matching server cannot learn the identities of non-members, cannot reconstruct part of the social graph and does not even learn the size of the contact lists of individual users. The protocol is not yet perfect however: a malicious mutual contact may prevent itself from being discovered. The protocol assumes sender anonymous communication channels, and uses the existing social graph as the source for certified identifiers. This is a stronger assumption than typically supposed by existing contact discovery protocols, but we will argue this is still a reasonable assumption to make in section 4.3. We discuss our results and summarise our conclusions in section 5.

2 State of the art

2.1 Traditional, one-sided, contact discovery

Many messaging services implement the traditional one-sided version of contact discovery where you get immediately notified when anyone on your contact list is a member as well.

WhatsApp uses a particularly naive and privacy invasive protocol. Its optional contact upload feature uploads phone numbers in plaintext to the WhatsApp server to identify contacts that are members as well. It stores hashes of those phone numbers that are not a member yet to "to more efficiently connect you with these contacts if they join WhatsApp".[4]

A natural candidate to make one-sided contact discovery more privacy friendly is to use private set intersection [13] to compute the intersection of the contact list (held by the user) and the total set of users of the messaging service (held by the server). These academic approaches are not practical however, as the traditional protocols do not take the unbalanced nature of the sets involved in contact discovery into account: the user has at most several hundreds of contacts, whereas the service typically has billions of users [28]. Even private set intersection (PSI) protocols optimised for unbalanced sets [25] perform poorly even after extensive optimisation: matching a thousand contacts against a quarter billion registered users still takes three seconds (assuming a WiFi connection) [22]. Such protocols may optimise the computational load for the device with the smallest set, but typically at the expense of requiring complex operations to be performed by the server storing the larger set. Moreover, such a PSI protocol needs to be run with *each* of the billions of users, and e.g., all of the protocols discussed by Kiss *et al.* [25] require the server to send a significant amount of data (in the best case a Bloom filter with a constant size, chosen between 1.8-5.5 MB, representing the list of known users). Other efficient PSI protocols like [8, 7] assume a stronger honesty requirement than usual: they allow an adversarial sender (the server) to force the output of the receiver (the

[4]https://faq.whatsapp.com/general/contacts/about-contact-upload, accessed 2021-10-14.

mobile device of the user) to equal all the members of its set *without needing to guess* the members of the set held by the mobile device as is typically the case in the honest-but-curious setting for private set intersection. This makes such protocols not useful for our application, as we wish to develop protocols that withstand even such malicious behaviour.

As result, privacy friendly messaging services like Signal resort to using trusted hardware based approaches. The basic protocol (hash the entries in a user's contact list before sending it to the server) is kept verifiably honest (in the sense that it doesn't store any information after running the contact discovery protocol) by running the server code on trusted hardware [27]. Clearly this is the most efficient way (at least from the client perspective) to solve contact discovery. It does require the server to have trusted hardware, and crucially depends on this hardware to be indeed trusted for privacy to hold.

2.2 Mutual contact discovery and related protocols

A problem very similar to mutual contact discovery, called *private matchmaking*, was studied almost forty years ago by Baldwin and Gramlich [1]. In abstract terms, the problem there is to determine whether two *known or identified* users share the same secret value (which potentially is from a small set, and therefore could be easy to guess). Potential use cases are to determine whether two agents have the same level of clearance, or a high-level job referral service where an intermediary allows a potential candidate for a particular job to verify whether a particular company is actually offering it. Baldwin and Gramlich propose a protocol involving the help of a central matchmaking server to allow two known parties A and B to verify whether they share the same secret. Due to certain technical details their protocol cannot be turned into a mutual contact discovery protocol directly.[5] But their ideas have inspired the protocols discussed in this paper.

Mutual contact discovery is also related to the following problems studied in the past. There is an important distinction, however, as mutual contact discovery assumes the participants are far apart and thus involves the cooperation of central matching server, while the protocols discussed below allow participants that are in each other's vicinity to discover or authenticate each other directly.

Private handshaking protocols (that could be considered a variant of private matchmaking protocols where the secret to check is known to have large entropy and therefore cannot be guessed) are an example [30, 2]. Because these protocols assume an input from a large entropy domain, these cannot directly

[5] In essence the problem is that as it stands the Baldwin and Gramlich protocol assumes the parties can exchange messages directly, among each other, before being discovered. In other words, it assumes that the underlying source of identifiers used to establish contact is itself a kind of communication network that allows the exchange of messages. When using mobile phone numbers as identifiers this is indeed the case, but we want our protocols to be more generally applicable using different kinds of identifiers without making such a strong assumption.

be used to solve mutual contact discovery (where the input, for example phone numbers, has relatively low entropy). Moreover, these protocols assume that the parties wishing to “shake hands” do so directly, without the involvement of a third party. We do note that the underlying techniques used in protocols for private matching and private handshaking prove to be useful to implement mutual contact discovery as well.

Heinrich *et al.* [17] studied mutual contact discovery in the context of AirDrop, Apple’s peer-to-peer file exchange protocol. In AirDrop, users can choose to be only discoverable for known contacts. Apple’s protocol has some privacy weaknesses, that allow an attacker to recover the contact details of both senders and receivers. Heinreich *et al.* propose to fix this by essentially running two private set intersection protocols with the set of known contacts and the identity (or identities) of the other participant as input.[6] Note that the setting we study here is different from that of AirDrop (or private set intersection generally), as mutual contact discovery for messaging is asynchronous and mediated by a third party (offering the messaging service), whereas AirDrop runs directly and in real-time between the two participants. This difference allows us to implement mutual contact discovery more efficiently, at least for the users whose computation and communication complexity is linear in the length of their own contact list (and does not depend on the length of the contact lists held by the other users). More importantly, mutual contact discovery allows a user to discover its mutual contacts among *all* other users at once.

Finally, we wish to point out the resemblance of mutual contact discovery with contact tracing and exposure notification, recently proposed to fight the COVID-19 pandemic, see [32, 29]. Although they are different in certain fundamental aspects (there is no underlying social graph over pre-existing identifiers, for example), nevertheless techniques used in e.g., the DESIRE protocol[7] turned out to be applicable in our setting as well.

3 Model and notation

Mutual contact discovery allows a potentially very large, dynamic, set of *users* from an existing social graph, that is represented by locally maintained contact lists, to discover each other when joining a new service. Alice and Bob are mutual contacts if Alice is a contact of Bob and Bob is a contact of Alice. In the case of messaging, the underlying social graph is based on the existing mobile

[6]We note that a different approach requiring only one run of a private set intersection protocol (see e.g., [13]) is also possible. Assume identities can be globally ordered. Concatenate the entries within your contact list with your own identity, putting the one that comes first in the global identity order first. This way, if Alice has Bob in her contact list and vice versa, both will construct an entry “Alice || Bob” in the augmented contact list. Private set intersection will find a match if and only if both are member of the other person’s contact list.

[7]See https://github.com/3rd-ways-for-EU-exposure-notification/project-DESIRE.

phone network, where the identifiers are mobile phone numbers and the social graph is given by the contact lists users maintain on their mobile phones.

Members can communicate directly, securely, and anonymously with a separate *matching server*, and can communicate with each other only through this matching server.[8] Intuitively, the mutual contact discovery protocol should satisfy the following requirements.

Correctness When Alice and Bob are mutual contacts, use the messaging service, are honest, and concurrently run the mutual contact discovery protocol, they will discover each other.

Security If Bob is not a member then Alice cannot be made to believe he is.

Membership privacy If the adversary is not a contact of Alice, then it will not be able to learn whether Alice is a member.

Contact privacy An adversary is not able to tell whether Bob is a contact of Alice, provided Alice and Bob are both honest.

In the above, the adversary could be a user or the matching server itself. We make no assumptions about the behaviour of adversarial users (or the matching server). We do assume that the matching server runs independently of the messaging server, in particular we assume that the matching server has no information about membership. This formal separation of roles makes the analysis in this paper cleaner as it allows us to focus on what exactly a matching server can and cannot learn purely on the basis of the mutual contact discovery protocol alone. (Clearly, the messaging service is able to reconstruct the social graph for those members that actively exchange messages.)

Note that the problem can be made more abstract by considering arbitrary identifiers and arbitrary relationships: all that contact discovery requires is an existing social graph represented by locally maintained contact lists. We now proceed with this more abstract definition.

3.1 Formal definition

Assume a directed 'social graph' (V, E) exists with vertices V representing people, and directed edges E representing their relationships. $A, B, \ldots \in V$ represent the unique identifiers of its members. Vertices know their own identifier, and maintain a list of contacts $contacts_A = \{B \in V \mid (A,B) \in E\}$. (Note how we associate a user with its identifier, and how the list of contacts captures the full social graph.) As V may be small, we assume these identifiers have relatively low entropy, and are therefore easily guessed or enumerated.

[8]This star topology is inherent to (current) messaging services, for which contact discovery is developed. In practice members will have many different ways to find and connect to each other directly, using their unique identifiers, but in the context of this problem we disallow them to use this ability.

It turns out that it is hard to prevent a malicious user from evading detection as a mutual contact. We therefore define a slightly more general, weaker, form of mutual contact discovery as follows. Allow members M to mark certain contacts for whom it wants to hide membership (but for whom it wants to discover whether that contact is a member), and mark the other contacts as visible. That is, define two disjoint sets $\mathit{hiddenby}_M$ and $\mathit{visible}_M$ such that $\mathit{contacts}_M = \mathit{visible}_M \cup \mathit{hiddenby}_M$. For honest members M, $\mathit{hiddenby}_M = \emptyset$. Moreover, for *strong* mutual contact discovery, $\mathit{hiddenby}_M = \emptyset$ for *all* members, even malicious ones. (This should then be enforced by the protocol.)

Definition 3.1 (mutual contact discovery protocol) *A mutual contact discovery protocol among a set of members $\mathscr{M}$ over a social graph (V, E) is an interactive protocol run between*

- *a set of members $\mathscr{M} \subseteq V$, where each member $M \in \mathscr{M}$ has a set of contacts $\mathit{contacts}_M \subseteq V$, divided into two disjoint sets $\mathit{visible}_M$ and $\mathit{hiddenby}_M$, and*
- *a matching server $\mathscr{X}$.*

The set of members $\mathscr{M}$ is unknown to both the matching server and each of the members. The protocol consists of the following two sub-protocols.

- $\mathsf{Setup}(1^\sigma)$ *is a probabilistic polynomial-time algorithm with as input a security parameter 1^σ. It generates the necessary system parameters, and initialises all members $M \in \mathscr{M}$ as well as the matching server $\mathscr{X}$.* Setup *must be run first.*
- $\mathsf{Discover}(\mathscr{M})$ *is an interactive protocol run in parallel between each of the members $M \in \mathscr{M}$ and the matching server $\mathscr{X}$. If successful, each $M \in \mathscr{M}$ obtains a set out_M of discovered contacts.*

Define
$$A \rtimes B \triangleq (A \in \mathit{visible}_B) \land (B \in \mathit{contacts}_A) \ .$$
Note that this is only a symmetric relationship among honest users, or for *strong* mutual contact discovery.

The protocol must satisfy the following correctness condition.

Definition 3.2 (Correctness) *For all $A \in \mathscr{M}$ and all $B \in V$ that follow the protocol we have*

$$\begin{cases} B \in \mathit{out}_A & \text{if } (B \in \mathscr{M}) \land A \rtimes B, \\ B \notin \mathit{out}_A & \text{otherwise, with overwhelming probability.} \end{cases}$$

Observe how this abstract definition captures the essence of mutual contact discovery while simplifying from the practical setting of mobile phone users joining a messaging service and discovering their mutual contacts in the following ways.

- The underlying social graph is assumed to be static, while in practice people add and remove contacts from their contact list.
- The set of members $\mathscr{M}$ of the messaging service is assumed to be static as well. Again in practice this is not the case.
- The contact discovery protocol is run once at the same time for all current members in parallel. In practice, members will run the discovery protocol as soon as they join, and occasionally afterwards.

As we will discuss later in section 4.2, this does not fundamentally change the problem and the results.

3.2 System model

The system is a star network with member devices at the edges and the matching server as the central node. Members can authenticate the server[9] and can set up a sender-anonymous secure communication channel that preserves the integrity, confidentiality and freshness of all messages exchanged. In this setting the server is authenticated, while members remain anonymous.[10] In fact we assume anonymous communication channels are used throughout (a fresh one for *each message exchange*). Tor[11] could be used for this purpose [12], or a system like Private Relay recently announced by Apple.[12] We will not go into details here, but simply note that many privacy preserving protocols need to make a similar assumption about the communication layer offering some kind of sender anonymity (or assuming a benevolently amnesiac server if such anonymity is not offered). For context see [18].

3.3 Threat model and security assumptions

We assume an *active* adversary where all adversarial entities may behave arbitrarily. In particular an adversarial member may add extra identifiers to its input $contacts_X$, and may decide to hide certain contacts using $hiddenby_X$. Such an adversary can observe, eavesdrop, replay, modify or block messages on the network. Note however that by assumption (see earlier this section) all communication between members and the matching server are secure (confidential and authentic), so in our analysis we only really need to consider observing and blocking messages.

[9]In what follows we write server to denote the matching server.

[10]In the conclusions section we will return to the question of how to keep members anonymous in practice, given that they communicate with the messaging server directly and thus leak their possibly identifying IP address. Note this is a *general* issue that plagues *any* contact discovery protocol that aspires to protect privacy.

[11]https://www.torproject.org

[12]https://developer.apple.com/videos/play/wwdc2021/10096

3.4 Requirements

We formally define the requirements our protocol should satisfy (as informally defined at the start of this section) using the 'malicious model' [14, 15] (which is weaker than the universal composability framework [6]). In this setting, all security, privacy and functional requirements are captured by the description of an *ideal system*, where a trusted third party (communicating using secure links) receives all participant inputs, locally computes the results, and returns all participant outputs to each participant individually. This ideal system is defined here as follows.

Definition 3.3 (ideal system for mutual contact discovery) *Let social graph (V, E) be given. Let $\mathscr{M} \subseteq V$ be a set of members, and let $\mathscr{X}$ be the matching server. Let $\mathscr{T}$ be a trusted server connected to each user in V and to the matching server $\mathscr{X}$ over a secure (and for users anonymous) communication link.*

Let $hiddenby_M$ and $visible_M$ be disjoint subsets of V given as input to participant M, set $contacts_M = hiddenby_M \cup visible_M$, and let ϵ be the input to the matching server $\mathscr{X}$.[13] *The ideal system for mutual contact discovery proceeds in the following synchronous phases.*

1. *Each member $M \in \mathscr{M}$ sends $hiddenby_M$ and $visible_M$ to trusted server $\mathscr{T}$. The matching server $\mathscr{X}$ sends ϵ to trusted server $\mathscr{T}$.*

2. *$\mathscr{T}$ computes, for all $M \in \mathscr{M}$, the set $out_M = \{X \mid (X \in \mathscr{M}) \wedge M \rtimes X\}$. $\mathscr{T}$ selects, for all $M \in \mathscr{M}$, a set $out'_M \subseteq out_M$ and sets $\epsilon_M = |out_M \setminus out'_M|$.*[14] *The selection of all out'_M is free except that $\sum_{M \in \mathscr{M}} \epsilon_M = \epsilon$ must hold.*

3. *$\mathscr{T}$ sends out'_M to each $M \in \mathscr{M}$ as its results, which each M outputs. $\mathscr{T}$ sends $|S_\triangleleft|$ and $|S_\rtimes|$ to $\mathscr{X}$ as its result, which it outputs.*[15]

Here

$$S_\triangleleft = \{(A, B) \in \mathscr{M} \times \mathscr{M} \mid A \in contacts_B\}$$

is defined as the set of all contact pairs, and

$$S_\rtimes = \{(A, B) \in \mathscr{M} \times \mathscr{M} \mid A \rtimes B\}$$

is defined as the set of all mutual contact pairs.

[13]Malicious participants may set their input $contacts_M$ to an arbitrary set. For honest members, $hiddenby_M = \emptyset$. For honest server, $\epsilon = 0$. ϵ models how often the matching server withholds information to participants.

[14]Here $\setminus$ denotes subtraction among sets, and recall that $M \rtimes X$ denotes that M and X are mutual contacts.

[15]This output is of course superfluous, but captures what the matching server learns during a protocol run, and by making the output explicit we stay within the general SMC paradigm.

Observe how this ideal model also prevents the matching server form learning the individual number of contact held by each member, a privacy property currently not achieved by other contact discovery protocols.

For ease of exposition we do not consider aborts; those are modelled as participants not submitting their full set of contacts. However, the definition of the ideal system is somewhat tricky because the matching server is an active participant of the real system, that may also misbehave. In particular we have to allow it to be only partially responsive, omitting to relay certain information to certain participants. This is what the error input ϵ is for.

The behaviour of the *real protocol* must subsequently be proven to be indistinguishable from the behaviour of the ideal model when simulated by a simulator corrupting the same set of parties involved. This is proven using a simulation proof technique [26].

Definition 3.4 (secure mutual contact discovery protocol) *A protocol Π (satisfying the syntactic of definition 3.1 and correctness constraints of definition 3.2) is a secure mutual contact discovery protocol if for every adversary $\mathscr{A}_\Pi$ (not tapping any communication channels) there exists a simulator $\mathscr{S}_I$ for the ideal system I of definition 3.3, where $\mathscr{S}_I$ controls the same set of entities as $\mathscr{A}_\Pi$, such that the joint view of $(\Pi, \mathscr{A}_\Pi)$ is indistinguishable from the joint view of $(I, \mathscr{S}_I)$.*

We define strong mutual contact discovery as follows.

Definition 3.5 (secure strong mutual contact discovery protocol) *A protocol Π is a secure strong mutual contact discovery protocol if it is a secure mutual contact discovery protocol according to definition 3.4 while for* all *members M, $hiddenby_M = \emptyset$ in the ideal system.*

3.5 Pairings and the BDH assumption

Let q be a large prime. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of order q. Let $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ be a *bilinear map* (also known as *pairing*) satisfying the following properties:

- For all $a, b \in \mathbb{Z}$ and all $P, Q \in \mathbb{G}_1$ we have $\mathbf{e}(aP, bQ) = \mathbf{e}(P, Q)^{ab}$.
- $\mathbf{e}$ is non-degenerate: for some $P, Q \in \mathbb{G}_1$ the pairing $\mathbf{e}(P, Q)$ is unequal to the identity element of $\mathbb{G}_2$.
- $\mathbf{e}$ can be efficiently computed.

A pairing satisfying these requirements is called *admissible*. Such groups and an admissible pairing exist [5].

Definition 3.6 (Bilinear Diffie-Hellman (BDH) problem) *The Bilinear Diffie-Hellman (BDH) problem over groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, and an admissible pairing $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ is defined as follows: given a random generator P of $\mathbb{G}_1$ and $aP, bP, cP \in \mathbb{G}_1$ for some random $a, b, c \in \mathbb{Z}_q^*$, compute $\mathbf{e}(P, P)^{abc}$.*

It is believed that the Bilinear Diffie-Hellman problem is hard (i.e., any randomised polynomial time adversary has at most a negligible advantage solving BDH) for certain choices of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbf{e}$ [5].

4 A protocol using certified identifiers

Our protocol resembles Boneh and Franklin's system for Identity Based Encryption [5]). Let $H_1 : V \mapsto \mathbb{G}_1$ be a cryptographic hash function that maps identities to points in $\mathbb{G}_1$. We define

$$P_A = H_1(A) \ .$$

Define the certificate $C(A)$ of a member identity A to be

$$C(A) = sP_A$$

for some secret $s \in \mathbb{Z}_q^*$. Furthermore define a *token* for a pair of users $A, B \in V$ as

$$T_{AB} = \mathbf{e}(C(A), P_B) \ .$$

We have the following property

Property 4.1 *For all $A, B \in V$ we have $T_{AB} = T_{BA}$.*

Proof: As P is a generator of $\mathbb{G}_1$ there are $a, b \in \mathbb{Z}_q^*$ such that $P_A = aP$ and $P_B = bP$. Then $T_{AB} = \mathbf{e}(saP, bP) = \mathbf{e}(P, P)^{sap} = \mathbf{e}(sbP, aP) = T_{BA}$. $\triangleleft$

The intuition of the next protocol is that the above property allows mutual contacts to both compute the same token to discover each other, but as the computation of T_{AB} requires an unforgeable (and secret) identifier certificate as input, no one else can compute this token (this is formally proven later in theorem 4.6).

The protocol runs in two phases. In the *submission phase*, each member M essentially submits a different token T_{MA} to the matching server for each of its contacts A (using an additional hash function to shield the actual token from that server). The matching server stores all hashed tokens received. If A also participates and has M as contact, the server will have two copies of that hashed token (recall that $T_{MA} = T_{AM}$) in its database at the end of the submission phase. In the *query phase*, M asks the server how many copies of the token T_{MA} it holds. If the server response indicates there are two such copies, a mutual contact is found. Tokens are hashed before submitting them to the server to prevent the server from constructing a valid response to a query, and thus preventing it from lying about a mutual contact that does not in fact exist.

The full protocol is specified as follows.

Definition 4.2 (Mutual contact discovery protocol) *Let P_A, $C(A)$ and T_{AB} be as defined above. Define the following protocol.*

- *The setup phase* $\mathsf{Setup}(1^\sigma)$ *generates a large prime* q *(of length* σ*), groups* $\mathbb{G}_1$ *and* $\mathbb{G}_2$ *of order* q*, an admissible pairing* $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$*, a generator* P *of* $\mathbb{G}_1$*, a secret* $s \in \mathbb{Z}_q^*$*, a public key* $P_{\text{pub}} = sP$*, and cryptographic hash functions* $H_1 : V \mapsto \mathbb{G}_1 \setminus P$ *and* $H_2 : \mathbb{G}_2 \times \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \{0,1\}^n$ *(for some choice of* n*, also of length* σ*).*

  Let $<$ *be a global order over* $\mathbb{G}_1$*, and define*

  $$H_2^<(X,Y,Z) = \begin{cases} H_2(X,Y,Z) & \text{if } Y < Z \\ H_2(X,Z,Y) & \text{otherwise.} \end{cases}$$

  Note that $H_2^<(X,Y,Z) = H_2^<(X,Z,Y)$*.*

  The setup phase publishes $\{q, \mathbb{G}_1, \mathbb{G}_2, \mathbf{e}, P, P_{\text{pub}}, H_1, H_2\}$ *and provides each member* $M \in \mathscr{M}$ *its certificate* $C(M) = sP_M = sH_1(M)$*. Members keep this certificate secret. The setup phase instructs the matching server* $\mathscr{X}$ *to initialise a set* $S = \varnothing$*. (To be clear, the matching server does not learn about the secret values.)* s *is erased at the end of the setup phase.*[16]

- *The interactive discovery protocol* $\mathsf{Discover}(\mathscr{M})$ *between a member* $M \in \mathscr{M}$ *and the matching server* $\mathscr{X}$ *runs in two phases, as follows.*

  - *In the* submission phase *each member* M*, for each* $A \in contacts_M$*, computes token* T_{MA}*.* M *then computes* augmented *tokens* $H_2^<(T_{MA}, P_M, P_A)$ *and* $H_2^<(T_{MA}, P_A, P_A)$*. It sends* $\langle H_2^<(T_{MA}, P_M, P_A), H_2^<(T_{MA}, P_A, P_A)\rangle$ *to the matching server, which stores it in* S*.*

  - *In the* query phase *each member* M *does the same and again sends the tuple* $\langle H_2^<(T_{MA}, P_M, P_A), H_2^<(T_{MA}, P_A, P_A)\rangle$ *to the matching server. The matching server returns all tuples in* S *that match* $H_2^<(T_{MA}, P_M, P_A)$ *on the first component but are different on the second component.*[17] M *then computes* $H_2^<(T_{MA}, P_M, P_M)$ *and verifies whether at least one of the tuples returned has this value as the second component. If so,* M *records that* A *is discovered as a mutual contact, adding* A *to* out_M*.*

  - *After the query phase, each* $M \in \mathscr{M}$ *outputs* out_M*. The matching server outputs* $|\{(X,Y),(X',Y') \in S \mid X = X' \wedge Y \neq Y'\}|$ *as value for* $|S_\rtimes|$ *and* $|S|$ *as value for* $|S_\lhd|$*.*

Observation 4.3 *We have* $H_2^<(T_{MA}, P_M, P_A) = H_2^<(T_{AM}, P_A, P_M)$*, so if* A *and* M *are mutual contacts and are both honest, they will discover each other.*

[16]Recall that we are currently only considering the static, one shot, setting as defined in definition 3.1, where an abstract trusted entity runs the setup phase and provides members their certificates. We will return to this in section 4.3.

[17]If an external observer should not be allowed to notice whether a query returns a mutual contact or not, the response should be made to be exactly one (possibly random) tuple. To simplify the proof we assume that whatever the matching server learns is public knowledge.

This follows from property 4.1 and the definition of $H_2^<$.

Note that by forcing the server to respond with $H_2^<(T_{MA}, P_M, P_M)$ in the query phase ensures that it cannot trick M into believing A is a member when in fact it is not: only A or M could have created that message (because T_{MA} is never revealed in the clear).

Recall that we assume communication links to be sender anonymous, and observe that indeed the protocol in no way relies on the matching server to be able to link different messages from the same user.

Observation 4.4 *When all participants in a run of protocol 4.2 are honest, then* $|\{(X,Y),(X',Y') \in S \mid X = X' \wedge Y \neq Y'\}|$ *and* $|S|$ *as returned by the protocol for* $|S_{\rtimes}|$ *and* $|S_{\lhd}|$ *are indeed the correct values according to definition 3.3.*

This follows from the fact that every honest member submits exactly one token pair for every contact, and that participants that are mutual contacts submit a token pair with the same first component (by observation 4.3).

4.1 Analysis

Model the hash functions H_1, H_2 as random oracles [3, 24]. Assume the Bilinear Diffie-Hellman problem is hard for our choice of $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbf{e}$. Security of our protocol depends on the hardness of the following problem in this setting.

Definition 4.5 (Mutual Contact Discovery Token (MCDT) problem) *The Mutual Contact Discovery Token (MCDT) problem over groups* $\mathbb{G}_1, \mathbb{G}_2$ *of order* q*, an admissible pairing* $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$*, a generator* P *of* $\mathbb{G}_1$*, a public key* $P_{\text{pub}} = sP$ *(for some random secret* $s \in \mathbb{Z}_q^*$*), and cryptographic hash functions* $H_1 : V \mapsto \mathbb{G}_1$ *is defined as follows: given the certificates* $C(X) = sP_X$ *for some* $X \in \mathcal{V} \subset V$ *chosen by the adversary, and random* $A, B \in V$ *with* $A \neq B$ *and* $A, B \notin \mathcal{V}$*, compute* $T_{AB} = \mathbf{e}(C(A), P_B)$ *(where* $C(X) = sP_X$ *and* $P_X = H_1(X)$ *for* $X \in V$*).*

Theorem 4.6 *If the Bilinear Diffie-Hellman (BDH) problem over groups* $\mathbb{G}_1, \mathbb{G}_2$ *of order* q*, and an admissible pairing* $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$ *is hard, then so is the Mutual Contact Discovery Token (MCDT) problem over the same* $\mathbb{G}_1, \mathbb{G}_2, \mathbf{e}$*, a generator* P *of* $\mathbb{G}_1$*, a public key* $P_{\text{pub}} = sP$ *and cryptographic hash functions* $H_1 : V \mapsto \mathbb{G}_1$ *(in the programmable random oracle model).*

Proof: Suppose not.

Let $\mathscr{A}_{\text{MCDT}}$ be an adversary for the MCDT problem. We show how to convert it into an adversary $\mathscr{A}_{\text{BDH}}$ for the BDH problem as follows, where $\mathscr{A}_{\text{BDH}}$ functions as the challenger for $\mathscr{A}_{\text{MCDT}}$.

Let the setup of $\mathscr{A}_{\text{BDH}}$ be groups $\mathbb{G}_1, \mathbb{G}_2$ of order q, and an admissible pairing $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$.

Let P be a generator of $\mathbb{G}_1$, and let $P_1 = sP$, $P_2 = aP$, and $P_3 = bP$ all in $\mathbb{G}_1$ for some random $s, a, b \in \mathbb{Z}_q^*$ be the challenge given to $\mathscr{A}_{\text{BDH}}$.

$\mathscr{A}_{\text{BDH}}$ sets up $\mathscr{A}_{\text{MCDT}}$ as follows: $\mathbb{G}_1, \mathbb{G}_2$, q, and $\mathbf{e}$ as given, P as in the challenge, and $P_{\text{pub}} = P_1$, also from the challenge. H_1 is a random oracle controlled by $\mathscr{A}_{\text{BDH}}$ as described below, that can be queried by $\mathscr{A}_{\text{MCDT}}$.

$\mathscr{A}_{\text{BDH}}$ answers queries to H_1 as follows. It maintains a mapping L_1 of already made queries and their responses. L_1 is initially empty. When queried for a point $X \in \mathbb{G}_1$ not in L_1 it picks a random $b \in \mathbb{Z}_q^*$, returns bP, and stores $L_1[X] = (b, bP)$. If X is in L_1 it looks up $(b, Q) = L_1[X]$ and returns Q. Observe how this guarantees a random distribution for the output of the H_1 oracle.

Next $\mathscr{A}_{\text{BDH}}$ allows $\mathscr{A}_{\text{MCDT}}$ to query the certificates $C(X) = sP_X$ for some $X \in \mathscr{V} \subset V$. $\mathscr{A}_{\text{BDH}}$ responds by first (internally) querying the oracle for each X and subsequently looking up $L_1[X]$. This equals (b, bP) for some b by construction. It responds by returning $bP_{\text{pub}} = bsP = sbP = sH_1(X) = sP_X$ by construction.

Then $\mathscr{A}_{\text{MCDT}}$ starts and is allowed queries to the H_1 oracle. Let us assume it makes q_E such queries.

$\mathscr{A}_{\text{BDH}}$ then sets up the challenge A, B to $\mathscr{A}_{\text{MCDT}}$ as follows: it picks random A and B and looks up A and B in L_1. If any of these exists in L_1, the adversary fails. Observe that the adversary is successful with non-negligible probability at least $(1 - q_E/|V|)^2$. (The queries made by $\mathscr{A}_{\text{BDH}}$ to compute the certificates do not count as by assumption $A, B \notin \mathscr{V}$.) Otherwise it returns P_2 for $H_1(A) = P_A$ and P_3 for $H_1(B) = P_B$ and updates L_1 accordingly (the random b is no longer relevant and can be kept empty). It then sends A and B as the challenge to $\mathscr{A}_{\text{MCDT}}$.

$\mathscr{A}_{\text{MCDT}}$ runs and is allowed additional queries to the H_1 oracle.

After some time $\mathscr{A}_{\text{MCDT}}$ responds with some non-negligible probability p with the correct response $T_{AB} = \mathbf{e}(C(A), P_B)$. But then $T_{AB} = \mathbf{e}(sP_A, P_B) = \mathbf{e}(sP_2, P_3) = \mathbf{e}(saP, bP) = \mathbf{e}(P, P)^{sab}$. In other words T_{AB} is a correct response to the BDH challenge, which implies $\mathscr{A}_{\text{BDH}}$ wins with non-negligible probability $p(1 - q_E/|V|)^2$, contrary to assumption. $\triangleleft$

We now prove that any view of an adversarial entity in a real protocol execution can be simulated by a simulator in the ideal system, as required by definition 3.4.

Theorem 4.7 *The mutual contact discovery protocol 4.2 is secure according to definition 3.4; it's behaviour cannot be distinguished from the ideal system 3.3.*

Proof: Informally, the theorem is based on the following reasoning. Theorem 4.6 ensures that only A and M can compute a token T_{MA}. As T_{MA} is only ever sent encapsulated by H_2, which is modelled by a random oracle, no other party learns any information about T_{MA}. We conclude that only A and M can compute an augmented token $H_2^<(T_{MA}, P_X, P_Y)$ for any X and Y, but observe that the matching server $\mathscr{X}$ learns such values when they are sent (by either A or M as just argued). Also note that by the same reasoning a valid tuple $\langle H_2^<(T_{MA}, P_M, P_A), H_2^<(T_{MA}, P_A, P_A) \rangle$ (i.e., a tuple where the first and second part actually belong together) can only be constructed by A or M (you cannot even see from the outside whether $H_2^<(T_{MA}, P_X, P_Y)$ and $H_2^<(T_{A'M'}, P_{X'}, P_{Y'})$ match on $A = A'$ and $M = M'$). This means we only need to consider the originator of such tuples: a malicious server could share such tuples with other users but this would in the end only result in the server adding meaningful, valid, tuples to its set S that it already has.

We now proceed with a more formal, detailed, proof.

In the following, the simulator provides the adversary oracle access to the hash function H_2, maintaining a mapping of already returned responses L_2, initially empty. Whenever the adversary queries the oracle for an input (X, Y, Z) the simulator looks up $L_2[(X, Y, Z)]$ and returns that as the response if it exists, or creates a random element from $\{0, 1\}^n$, stores it in $L_2[(X, Y, Z)]$ and returns that. L_2 is maintained such that given an earlier response x the simulator can efficiently retrieve (X, Y, Z) such that $L_2[(X, Y, Z)] = x$

Simulating a participant M:

We start with the submission phase. The simulator is given $C(M)$ and starts adversarial M. M will return a list of tuples to submit to the matching server. The task of the simulator is to extract the input $\mathit{hiddenby}'_M$, $\mathit{visible}'_M$ and hence $\mathit{contacts}'_M$. It can do so as follows, considering the tuples one by one.

Let $\langle H', H'' \rangle$ be the next tuple. According to the protocol, this tuple should match $\langle H_2^<(T_{MA}, P_M, P_A), H_2^<(T_{MA}, P_A, P_A) \rangle$ for some A. The simulator looks up the preimage of H' and H'' in L_2.

If the preimage of H' does not exist, it skips this tuple and considers the next. (It can do so because then, with overwhelming probability, no other honest member A' will submit a valid token $H_2^<(T_{A'M}, P_{A'}, P_M) = H'$ that could result in a match at the server.)

Otherwise, let (T, X, Y) be this preimage, i.e., it is the result of calling H_2 for either $H_2^<(T, X, Y) = H'$ or $H_2^<(T, Y, X) = H'$. The simulator verifies that $X < Y$. If not, it skips this tuple and considers the next. (It can do so because no other honest member A' will query the oracle with $X \geq Y$.) The simulator then verifies that $X = M$ and $T = T_{XY} = \mathbf{e}(C(M), Y)$ and sets $A' = Y$ (or that $Y = M$ and $T = T_{YX} = \mathbf{e}(C(M), X)$ and sets $A' = X$). If this verification step fails, then it also skips this tuple and considers the next. (It can do so because then, with overwhelming probability, again no other honest member A' will submit a valid tuple with H' as the first field. This is because according to theorem 4.6, M can only construct a valid token T of the form $\mathbf{e}(C(M), Z)$ for some Z, and by the construction of a valid H', Z should equal either X or Y.)

If the preimage of H'' does not exist, then the simulator adds A' to $\mathit{hiddenby}'_M$ and proceeds with the next tuple. (Because then, with overwhelming probability, A' will not detect M as a mutual contact, while M will detect A' if it is a mutual contact and participating.)

Otherwise, let (T', X', Y') be this preimage, i.e., $H_2^<(T', X', Y') = H''$. The simulator then verifies that $T = T'$ and $X' = Y' = A'$. If any of these tests fail, it adds A' to $\mathit{hiddenby}'_M$ (again because in this case A' will not detect M as a contact). Otherwise it adds A' to $\mathit{visible}'_M$. It then proceeds with the next tuple.

When all tuples have been considered, the simulator removes any A' from $\mathit{hiddenby}'_M$ that are also a member of $\mathit{visible}'_M$.

We now consider the query phase.

1. The simulator receives out_M from the trusted third party.
2. For every $\langle X, Y \rangle$ it receives from the adversary the simulator returns all tuples stored locally during the submission phase for which X matches the first component and Y does not match the second component. (Observe how this is corresponds exactly to what the matching server would return if interacting with the adversarial M based on what it received from M itself. This ensures the adversary cannot distinguish interacting with the simulator or the real server.) The simulator also checks whether $X = H_2^{<}(T_{MA}, P_M, P_A)$ for some $A \in \mathit{contacts}_M$ (again it can do so). If this is not the case, nothing happens. Otherwise, if $A \in \mathit{out}_M$ it adds $(H_2^{<}(T_{MA}, P_M, P_A), H_2^{<}(T_{MA}, P_M, P_M))$ to its response to the adversary (if it hadn't added that tuple already).
3. The simulator outputs whatever the adversary outputs.

With the above observations in mind, it is easy to see that the view of the adversary in the real protocol execution is indistinguishable from its view provided by the simulator. Therefore its output is the same, and hence the output of the simulator in the ideal case equals the output in the real case.

Simulating the matching server:

We start with the submission phase. The simulator receives $\mathit{hiddenby}'_M$ and $\mathit{visible}'_M$ from all M and uses that to compute $|S_{\triangleleft}|$ and $|S_{\rtimes}|$. It interacts with the (adversarial) matching server as follows.

We use observation 4.4 to construct the necessary information that the simulator needs to send the right (number of) messages in the submission and the query phase. Recall these messages are pairs (X, Y) of augmented tags.[18] We know that $|S_{\triangleleft}|$ equals $|S|$ and hence denotes the number of submission messages the simulator needs to send. Of those messages, $|S_{\rtimes}|$ denotes the number of times a particular augmented tag occurs twice as X (the first element) in such pairs.

Randomly generate $|S_{\rtimes}|$ 'tokens' T_i and points P_i and P'_i and compute $X_i = H_2^{<}(T_i, P_i, P'_i)$, $Y_i = H_2^{<}(T_i, P_i, P_i)$, and $Y'_i = H_2^{<}(T_i, P'_i, P'_i)$. Send $\langle X_i, Y_i \rangle$ and $\langle X_i, Y'_i \rangle$ to the matching server. The random oracle for H_2 ensures that these are indistinguishable from the actual pairs of tuples the matching server receives for actual mutual contacts in the real execution of the protocol.

[18] And by assumption do not contain any information about its sender, even in the lower layer.

For the remaining $|S_{\triangleleft}| - 2|S_{\rtimes}|$ contacts, generate random 'tokens' T_i' and points P_i'', compute $X_i' = H_2^{<}(T_i', P_i, P_i'')$ and $Y_i'' = H_2^{<}(T_i, P_i'', P_i'')$, and send $\langle X_i', Y_i'' \rangle$ to the matching server. The random oracle for H_2 again ensures that these are indistinguishable from the tuples the matching server receives for actual non-mutual contacts in the real execution of the protocol.

We now turn to the query phase. For each of the $X_i = H_2^{<}(T_i, P_i, P_i')$ computed for mutual contacts in the submission phase, the simulator sends the queries $\langle X_i, Y_i \rangle$ and $\langle X_i, Y_i' \rangle$ to the server. For each of the $X_i' = H_2^{<}(T_i', P_i, P_i'')$ computed for non-mutual contacts in the submission phase, it sends the query $\langle X_i', Y_i'' \rangle$. These messages correspond exactly to the messages the real (honest) participants would send.

The simulator initialises ϵ to 0.

Given that the simulator sent all submissions to the server itself, it knows exactly what responses to expect. In particular, for each query $\langle X_i, Y_i \rangle$ corresponding to a *mutual* contact, it expects the response $\langle X_i, Y_i' \rangle$ (and vice versa), while for a query $\langle X_i', Y_i'' \rangle$ corresponding to a *non-mutual* contact it expects no response at all. If it doesn't receive a response, or receives and invalid response when querying for a (internally known) mutual contact, it increases ϵ by one.

At the end of the query phase, the simulator sends ϵ to the trusted server. ϵ exactly corresponds to the number of times a participant fails to discover a mutual contact because the matching server lies or remains silent when queried. The resulting output in the ideal system is therefore indistinguishable from the output that would have resulted if the adversarial server had interacted with the real world.

$\triangleleft$

4.2 The dynamic, real life setting

The protocol presented satisfies the static setting of a mutual contact discovery protocol according to definition 3.1. As already mentioned earlier, in practice the setting is much more dynamic: the underlying social graph is not static, and people join and leave at arbitrary times. Therefore, in practice the protocol should allow members to repeatedly check for new contacts. Also, what happens when people join and leave deserves more attention.

A simple modification allows the protocol to also cater for this dynamic, real life setting as follows. Instead of running synchronously and distinguishing a submission and query phase, the protocol now runs asynchronously forever and only supports querying. Initially the token database S of the server is empty. Members M can query for contacts A at arbitrary times, using the same tuple $\langle H_2^{<}(T_{MA}, P_M, P_A), H_2^{<}(T_{MA}, P_A, P_A) \rangle$ as in the original protocol. Whenever the matching server receives a new tuple, it adds it to its database S. Also, as a response to any such query, the matching server returns all tuples in S that match $H_2^{<}(T_{MA}, P_M, P_A)$ on the first component but are different on the second component.

Because the protocol is now asynchronous, if M queries for contact A before A queries for contact M, only A will be informed of a mutual contact. This means that in practice if a member adds a new contact to their contact list, they should repeatedly query the matching server for this new contact until a match is found. Contacts for which mutual contact has already been established do in principle not need to be queried again.

However, this approach only considers the case where contacts are added. What about the case where members delete contacts from their list? For this the protocol could be amended to allow users to send a delete message to the matching server. (Recall from the discussion in the introduction that users do not necessarily always want to inform a contact that they were deleted from their contact list.) To delete a contact A, member M sends the tuple $\langle H_2^<(T_{MA}, P_M, P_A), H_2^<(T_{MA}, P_A, P_A)\rangle$, asking the matching server to delete it from S. If we trust the server to honour such requests, this ensures that if A later joins as a member, M will not be discovered as a mutual contact by A (and M will also not be informed of A's presence). However, if A already was a member, it would have discovered M as a mutual contact earlier. To prune their mutual contact list, members will have to check the status of *all* their contacts once in a while, by querying the matching server.

We argue that these modifications do not alter the privacy properties in any significant way. The server will only receive $\langle H_2^<(T_{MA}, P_M, P_A), H_2^<(T_{MA}, P_A, P_A)\rangle$ messages for any A that was in $\mathit{contacts}_M$ at any point in time. Those are the same messages it would have received when running the static protocol in a state where *all* these A are a contact of M at the same time. Also, as far as the matching server is concerned all these messages are completely unrelated (because different hashed tokens cannot be linked) and could have come form many different members instead. The additional timing information about when particular messages are sent thus becomes essentially useless. We conclude the matching server does not learn anything more significant than it could have learned from running the static protocol once. Any member will similarly receive only tuples $\langle H_2^<(T_{MA}, P_M, P_A), H_2^<(T_{AM}, P_A, P_A)\rangle$ for any A that was a mutual contact at any point in time, which it would have received when running the static protocol once.

4.3 Where do the certificates come from?

Another fundamental question is of course where the certificates used come from. One way to look at this is by considering certificates being provided by the existing, underlying, social graph. One setup that could work in practice would be one where we leverage the existing infrastructure of mobile phones (messaging services typically use mobile phone numbers as identifiers) and let the mobile network operator (that hands out these mobile phone numbers to subscribers) issue the certificates.[19] This would have the additional advantage that this prevents users from getting certificates for phone numbers that are not their own, as the mobile network operator can strongly authenticate its users through the SIM card they issue to their users.

A separate certificate issuer is also an option, but note that s must be kept secret and should not be managed by the matching server. A natural approach would be to host it along the messaging server, as this is already assumed not to collude with the matching server and by definition already learns the set of members. The method used by the messaging server to authenticate the identity of new users could be used to prevent users from certificates for identities that do not belong to them. The best approach would be to implement this in trusted hardware (as Signal does now for the *full contact discovery process*), to prevent the secret s from being leaked.

4.4 Other considerations

A subtle issue is the separation between the matching server and the messaging server. In particular, the question is to what extent a malicious user can bypass *mutual* contact discovery. Modern end-to-end encrypted messaging services implement a public key directory containing the keys for their members, so a malicious user could in principle directly ask (through the messaging API) the messaging server for a key of a contact to detect membership. The UI of the messaging app typically prevents this, but the underlying APIs would still allow this.

A more secure implementation of mutual contact discovery therefore needs to prevent this at the API layer, by providing members an access token for each successfully discovered mutual contact. This access token is required to obtain the public key of a contact from the public key directory (which would otherwise refuse to respond, thus protecting knowledge of membership form non-mutual contacts).

A straightforward idea is to reuse the token already returned by the matching server when M queries it to detect whether A is a mutual contact or not. Recall that in case of A being a mutual contact, the matching server returns $H_2^<(T_{AM}, P_M, P_M)$. If A also submits $H_2^<(T_{AM}, P_M, P_M)$ to the messaging server as an access condition to access its public key when adding M to its contact list, then this token can be used by M to obtain A's public key, but only when A and M are mutual contacts. (Here we use the fact that the matching server and the messaging server do not collude.)

[19]Whether it is realistic to believe that mobile network operators will do so in order to cooperate with, e.g., Signal is debatable, but the point here is to show that alternative structures to make use of an existing social graph are possible.

5 Conclusions and discussion

Contact discovery is based on the observation that “access to an existing social graph makes building social apps much easier” [28]. We have shown that by levering such a social graph as the source for certified identifiers, mutual contact discovery can securely and privately be implemented in a malicious adversarial setting.

Our protocol only provides *weak* mutual contact discovery, and allows a malicious mutual contact to evade discovery. A protocol for strong mutual contact discovery would require the matching server to verify that every tuple submitted to the matching server is correct (without learning about its contents), in the sense that it is guaranteed that a potential mutual contact will also discover the member submitting this tuple. This would probably require zero-knowledge proofs of some form, which would be prohibitively expensive. This is left for further research.

We did not implement a prototype and therefore are unable to offer true benchmarks. But let us consider the protocol for the dynamic, real life setting explained in section 4.2, and argue why this could be practical.

First consider computations. Note how the server is not involved in any complicated computations: it simply stores and looks up values, and answers queries. From the server side this is as good as it could possibly get, in terms of computations. Clients need to compute a pairing though, which may be expensive, but they only need to do so *once* for each member they add to their contact list, if they store the resulting token for later re-use.

Now consider storage. Server storage is a concern. Given the real-life numbers quoted earlier in section 2 – a quarter billion registered users with a thousand contacts each – implies a database containing dozens of terabytes (instead of the dozens of gigabytes for registered phone numbers otherwise needed to implement straightforward contact discovery, like Signal does [27]). In other words, the database grows by a factor corresponding to the average size of the contact lists. Terabyte-size high-traffic databases are used in practice however,[20] and in this case only need to support a lookup using the primary key (the first hashed token in the tuple). The database can even be sharded, based on this primary key, to distribute load.

Finally consider network traffic and related to that server. Clients can essentially run mutual contact discovery as in the traditional one-sided case, querying the server when a contact is added and regularly refreshing the state for already discovered contacts. Deletion is a new operation, but will be at most as frequent as adding a contact. We see that the number of requests to the server (determining network traffic and server load) are essentially the same as in the traditional one-sided case currently being used on a global scale by apps like WhatsApp and Signal. And given the protocol, the matching server does not

[20]See e.g. https://www.adyen.com/blog/updating-a-50-terabyte-postgresql-database.

have to be implemented in trusted hardware in this case.

Subtle privacy issues remain. For example, if B knows A is using the service and $A \in contacts_B$, then if B doesn't get $A \in out_B$ then it learns A didn't put B on their contact list.

In our analysis we have somewhat artificially separated the matching server from the main messaging service, to clearly delineate what can be learned *from the matching protocol alone*. Clearly membership privacy against the server is not really something that can be required in any meaningful sense in practice: the service provider needs to know its members.

Mutual contact discovery restricts the notification of new contacts to those that are mutual. As explained in section 4.4 the messaging services themselves should (and can) technically prevent non-mutual contacts to obtain a key or send a message. This will prevent (malicious) users to discover members automatically using some form of scripting.

The full benefit of privacy friendly contact discovery (one-sided, or mutual) is of course only achieved when combined with truly anonymous messaging services [20, 10, 21] so that mutual contacts cannot be discovered even by the messaging service itself while delivering the messages. But these services typically do not rely on a small entropy identifier like a phone number to identify contacts, but on public keys instead. This makes the problem inherently easier to solve. It is often left out of scope to discuss how members obtain the public keys of other members. But if members need to meet anyway to exchange public keys, this of course establishes mutual contact by definition, and the problem evaporates all together.

References

[1] R. W. Baldwin and W. C. Gramlich. “Cryptographic Protocol for Trustable Match Making”. In: *1985 IEEE S&P* (Oakland, CA, USA, Apr. 22–24, 1985). IEEE Comp. Soc., 1985, pp. 92–100.

[2] D. Balfanz et al. “Secret Handshakes from Pairing-Based Key Agreements”. In: *2003 IEEE S&P* (Berkeley, CA, USA, May 11–14, 2003). IEEE Comp. Soc., 2003, pp. 180–196.

[3] M. Bellare and P. Rogaway. “Random oracles are practical: A paradigm for designing efficient protocols”. In: *CCS 1993* (Fairfax, VA, USA, Nov. 3–5, 1993). ACM, 1993, pp. 62–73.

[4] A. Biryukov, D. Dinu, D. Khovratovich, and S. Josefsson. *Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications*. RFC 9106. RFC Editor, Aug. 2016, pp. 1–21. URL: http://www.rfc-editor.org/rfc/rfc9106.txt.

[5] D. Boneh and M. Franklin. “Identity-based encryption from the Weil pairing”. In: *SIAM J. Comput.* 32.3 (2003), pp. 586–615.

[6] R. Canetti. “Universally Composable Security: A New Paradigm for Cryptographic Protocols”. In: *42nd FOCS* (Las Vegas, NV, USA, Oct. 14–17, 2001). IEEE Comp. Soc., 2001, pp. 136–145.

[7] H. Chen, Z. Huang, K. Laine, and P. Rindal. “Labeled PSI from Fully Homomorphic Encryption with Malicious Security”. In: *CCS 2018* (Toronto, Canada, Oct. 15–19, 2018). ACM, 2018, pp. 1223–1237.

[8] H. Chen, K. Laine, and P. Rindal. “Fast Private Set Intersection from Homomorphic Encryption”. In: *CCS 2017* (Dallas, TX, USA, Oct. 30–Nov. 3, 2017). ACM, 2017, pp. 1243–1255.

[9] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. “Private information retrieval”. In: *JACM* 45.6 (1998), pp. 965–981.

[10] H. Corrigan-Gibbs, D. Boneh, and D. Mazières. “Riposte: An Anonymous Messaging System Handling Millions of Users”. In: *2015 IEEE S&P* (San Jose, CA, USA, May 17–21, 2015). IEEE Comp. Soc., 2015, pp. 321–338.

[11] W. Diffie and M. E. Hellman. “New directions in cryptography”. In: *IEEE Trans. Inf. Theory* IT-11 (Nov. 1976), pp. 644–654.

[12] R. Dingledine, N. Mathewson, and P. F. Syverson. “Tor: The Second-Generation Onion Router”. In: *13th USENIX Sec. Symp.* (San Diego, CA, USA, Aug. 9–13, 2004). USENIX Association, 2004, pp. 303–320.

[13] M. Freedman, K. Nissim, and B. Pinkas. “Efficient private matching and set intersection”. In: *EUROCRYPT 2004* (Interlaken, Switzerland, May 2–6, 2004). LNCS 3027. Springer, 2004, pp. 1–19.

[14] O. Goldreich, S. Micali, and A. Wigderson. “How to Play ANY Mental Game”. In: *19th STOC* (New York City, NY, USA, May 25–27, 1987). ACM, 1987, pp. 218–229.

[15] O. Goldreich. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge, UK: Cambridge University Press, 2001.

[16] C. Hagen, C. Weinert, C. Sendner, A. Dmitrienko, and T. Schneider. “All the Numbers are US: Large-scale Abuse of Contact Discovery in Mobile Messengers”. In: *NDSS 2021* (virtual, Feb. 21–25, 2021). 2021.

[17] A. Heinrich, M. Hollick, T. Schneider, M. Stute, and C. Weinert. “PrivatEDrop: Practical Privacy-Preserving Authentication for Apple AirDrop”. In: *30th USENIX Sec. Symp.* (online, Aug. 11–13, 2021). USENIX Association, 2021, pp. 3577–3594.

[18] J.-H. Hoepman. “Privacy Friendly E-Ticketing For Public Transport”. CoRR abs/2101.09085. Jan. 2021. arXiv: 2101.09085 [cs.CR]. URL: https://arxiv.org/abs/2101.09085.

[19] J.-H. Hoepman. “Private Handshakes”. In: *4th ESAS* (Cambridge, UK, June 2–3, 2007). LNCS 4572. 2007, pp. 31–42.

[20] J.-H. Hoepman. “Privately (and Unlinkably) Exchanging Messages Using a Public Bulletin Board”. In: *WPES 2015* (Denver, CO, USA, Oct. 12, 2015). ACM, 2016, pp. 85–94.

[21] J. van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. “Vuvuzela: scalable private messaging resistant to traffic analysis”. In: *Proc. of the 25th Symp. on Operating Systems Principles, SOSP 2015* (Monterey, CA, USA, Oct. 4–7, 2015). ACM, 2015, pp. 137–152.

[22] D. Kales, C. Rechberger, T. Schneider, M. Senker, and C. Weinert. “Mobile Private Contact Discovery at Scale”. In: *28th USENIX Sec. Symp.* (Santa Clara, CA, USA, Aug. 14–16, 2018). USENIX Association, 2019, pp. 1447–1464.

[23] B. Kaliski. *PKCS #5: Password-Based Cryptography Specification. Version 2.0*. RFC 2898. RFC Editor, Sept. 2000. URL: http://www.rfc-editor.org/rfc/rfc2898.txt.

[24] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. Boca Raton: CRC Press, 2014. URL: https://www.crcpress.com/Introduction-to-Modern-Cryptography-Second-Edition/Katz-Lindell/p/book/9781466570269.

[25] Á. Kiss, J. Liu, T. Schneider, N. Asokan, and B. Pinkas. “Private Set Intersection for Unequal Set Sizes with Mobile Applications”. In: *PoPETs* 2017.4 (2017), pp. 177–197.

[26] Y. Lindell. “How to Simulate It - A Tutorial on the Simulation Proof Technique”. In: *Tutorials on the Foundations of Cryptography*. Ed. by Y. Lindell. Springer International Publishing, 2017, pp. 277–346. URL: https://doi.org/10.1007/978-3-319-57048-8_6.

[27] M. Marlinspike. “Technology Preview: Private Contact Discovery for Signal”. Sept. 26, 2017. URL: https://signal.org/blog/private-contact-discovery/.

[28] M. Marlinspike. “The Difficulty Of Private Contact Discovery”. Jan. 3, 2014. URL: https://signal.org/blog/contact-discovery.

[29] T. Martin, G. Karopoulos, J. L. Hernandez Ramos, G. Kampourakis, and I. Nai Fovino. “Demystifying COVID-19 digital contact tracing: A survey on frameworks and mobile apps”. In: *Wireless Communications and Mobile Computing* (2020), p. 8851429.

[30] C. Meadows. “A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party”. In: *1986 IEEE S&P* (Oakland, CA, USA, Apr. 7–9, 1986). IEEE Comp. Soc., 1986, pp. 134–137.

[31] C. Percival and S. Josefsson. *The scrypt Password-Based Key Derivation Function*. RFC 7914. RFC Editor, Aug. 2016, pp. 1–26. URL: http://www.rfc-editor.org/rfc/rfc7914.txt.

[32] World Health Organization. "Contact tracing in the context of COVID-19, Interim guidance". May 10, 2020. URL: www.who.int/publications/i/item/contact-tracing-in-the-context-of-covid-19.

A Introduction to some other approaches

The protocol presented in the main body of this paper has the best privacy and security properties of all protocols we considered for solving mutual contact discovery. However, the use of pairings may make this a less desirable protocol in practice. It is therefore interesting to also briefly discuss some of the other ideas we had, that are more efficient to implement in practice (albeit with worse security and privacy guarantees).

The first one (discussed in section B) operates without additional services in an actively adversarial model, but nevertheless protects user privacy better than plain contact discovery protocols: it prevents adversaries from recovering contacts if little prior knowledge about contacts is known, and in particular prevents the server from recovering the social graph. The second protocol (see section C) significantly improves the privacy protection offered, because it also prevents an active adversarial server or third party to test even a suspicion of two users being a contact. It is actually quite similar to the main protocol presented in section 4, requiring an online key server instead.

We briefly sketch the protocols and argue why they have the properties we claim they have, without a full formal model or proof.

A.1 Security and privacy definitions

The first protocol in this appendix operates in a weaker setting, where guessing a particular member's identifier or a potential contact matters. Therefore, apart from the general threat model discussed above, for the protocols[21] discussed in this appendix we need to assume that adversaries will use prior knowledge to maximise their chance of success. In particular, it is realistic to assume an adversary knows the list of phone numbers in use, and knows the phone number of several persons of interest. It may also have some knowledge regarding the likely contacts of a person of interest.

When defining the requirements below, we need to consider adversarial behaviour of both members an the server. In the definitions, X denotes such a set of colluding adversarial entities, which for all definitions given below is a subset of the active members and may include the server. We have the following security requirement.

[21] We could have proven the second protocol correct in the more strict setting used in the proof of correctness for the certified identifiers based protocol presented in the main body of this paper, but choose not to for easy of presentation.

Definition A.1 (Security) *Let A and B in V be arbitrary users such that $A \in \mathscr{M}$ follows the protocol. If $B \notin \mathscr{M}$ or $B \notin contacts_A$ then X cannot force $B \in out_A$.*

Note that this definition is a slight relaxation of the correctness requirement, as it does not take into account whether $A \in contacts_B$. The reason being that if this is not the case, B could easily pretend A is one of its contacts. Also note that the *matching* server does not know or control $\mathscr{M}$.

We have also have the following two privacy requirements (where we write "X does not learn P" as a shorthand for "entities in X cannot decide whether P holds by observing a protocol run or by engaging in a protocol run, restricted by the threat model and security assumptions discussed below").

We first define *membership privacy*.

Definition A.2 (Membership privacy) *If $X \cap contacts_A = \emptyset$, then X does not learn whether $A \in \mathscr{M}$, for any $A \notin X$ of its choosing.*

Note that the definition of membership privacy only restricts users that do *not* appear in the contact list of a user A to discover whether A is actually a member of the service. In other words, we do not consider it a violation of membership privacy if some $Z \in contacts_A$ discovers $A \in \mathscr{M}$ even if $A \notin contacts_Z$, as the fact that A included Z as a contact signals that A allows Z to discover them, and Z can always pretend that $A \in contacts_Z$. Also note that $A \notin X$ implies A is honest. The definition also applies to X including the server (which is not a member of $contacts_A$ by definition).

We next define *contact privacy*.

Definition A.3 (Contact privacy) *X does not learn whether $B \in contacts_A$ (or not), for any $A \notin X$ and $B \notin X$ of its choosing.*

Note that this definition sidesteps the problem of B itself being able to detect whether $B \in contacts_A$ as it can always pretend that $A \in contacts_B$ as discussed above. In other words, this is not considered a violation of contact privacy. Again A and B are honest by definition.

A.1.1 On the choice of definitions

As is often the case, there are different ways how one might define relevant requirements that capture all the essential aspects of a particular problem.[22] In our case we could have defined a weaker version of contact privacy as "X does not learn an $A, B \notin X$ such that $A \in contacts_B$" (and a similarly weaker version of membership privacy). Except for contact privacy in the first protocol, that turns out not to matter much (as we will see further on).

[22]Note that this is much less of a problem in the ideal versus real model used in the main body of the paper as the ideal model captures all the intuitive properties you would want the protocol to have 'for free'.

Although membership privacy and contact privacy appear to be two entirely different notions at first sight, they are in fact related. If an entity X would be able to break contact privacy, it can test whether $B \in contacts_A$, while by definition $X \cap contacts_A = \emptyset$. But if it discovers $B \in contacts_A$ then surely $A \in \mathscr{M}$: if A is not participating in a run of the mutual contact discovery protocol, it cannot leak information about its contacts. Of course, to test such a suspicion, X needs to guess the right B (if it guesses wrong, i.e., a $B \notin contacts_A$, then the negative answer of the contact privacy 'oracle' is useless as an indication of whether $A \in \mathscr{M}$). The success of the adversary therefore depends on its knowledge of possible contacts, and we see that membership privacy weakly implies contact privacy.

The other way around, suppose an entity X would be able to break membership privacy, and determine whether $A \in \mathscr{M}$ while $X \cap contacts_A = \emptyset$. No general statement about contact privacy can be made in this case. For example, a system can maintain contact privacy while every member $A \in \mathscr{M}$ broadcasts this fact to the world. On the other hand, this does appear to be a pathological case: the idea of mutual contact discovery is that the fact that a user is a member is hidden for anybody except their contacts. As a result one would expect that the only way to break membership privacy is by learning something about the contacts of A.

We conclude that even though membership privacy and contact privacy are related, they are sufficiently different to warrant separate study.

A.2 Notation

We use a concatenation operator $\|$ where we assume that the resulting binary string $x \| y$ can be unambiguously decomposed into its parts x and y. In other words, there do not exist different x' and y' such that $x \| y = x' \| y'$.

The protocols below uses a *key derivation function* as a primitive. A key derivation function $\mathsf{KDF}(\cdot) : \{0,1\}^* \mapsto \{0,1\}^\sigma$ [23] (where σ is the security parameter) is a cryptographic hash function that is not only hard to invert (and is pre-image and collision resistant) but also takes a 'significant' time to compute. This means the function cannot be inverted in practive over relatively small, low entropy, domains (because computing a dictionary is prohibitively expensive). A good choice would be scrypt [31] or Argon2 [4] which won the password hashing competition in 2015.[23] Indeed the problem we face in this paper is very similar to that of protecting passwords stored in password files, as user chosen passwords typically have a low entropy as well.

Usually, key derivations functions (KDFs) include a parameter to tune their time complexity (for example by setting the number of iterations). This can be tweaked to achieve a decent security level for a particular input domain.

[23] See https://www.password-hashing.net, accessed 2021-10-15.

B A simple protocol

Define a token between users $A, B \in V$ as

$$v_{AB} = \mathsf{KDF}(A \| B) \ .$$

Then the following is a protocol for mutual contact discovery between a member $M \in \mathscr{M}$ and the matching server $\mathscr{X}$.

- In the *submission phase* each member M, for each $A \in \mathit{contacts}_M$, computes v_{AM}. M sends this value to the matching server, which stores it in S.
- In the *query phase* each member M now sends v_{MA} (i.e., with A and M reversed) for all $A \in \mathit{contacts}_M$ to the server. The server returns whether $v_{MA} \in S$. If so, M records that A is discovered as a mutual contact, adding A to out_M.

B.1 Informal analysis

The input to the KDF is the concatenation of two identifiers (typically phone numbers). For a messaging service with a national reach the set of phone numbers is typically several million in size. For a global system like WhatsApp or Signal, this is more like several billion. As mentioned before, contact lists can contain several hundreds of entries.

Let us consider the smaller, national, setting (which is therefore is the most risky). Let's say there are $2^{24} \approx 16,7$ million possible phone numbers. Then (because of concatenation) the input domain to the KDF contains at least 2^{48} possible elements (we ignore the round number here). Let us assume that the average customer hardware is a million (2^{20}) times slower than the fastest hardware available to the attackers. And let us assume that a very connected member has a contact list of at most several hundred entries (say 2^{8}), which each need to be processed by the KDF. Then the effort for an adversary to create a complete dictionary,[24] is $2^{48}/(2^{20} \times 2^{8}) = 2^{20}$ times higher than the effort for a member to participate in the protocol.

Using a KDF thus (in practice) limits parties without any prior knowledge to try all possible inputs and discover relevant ones. Parties with (specific enough) prior knowledge are less constrained and may thus break the privacy properties of the scheme. In the informal analysis presented below we say protection is 'limited' in this case.

Correctness First of all observe that when $A \in \mathit{contacts}_B$ and $B \in \mathit{contacts}_A$, i.e., when A and B *are* mutual contacts, and when both engage in the protocol during an even and following odd slot, both will be notified of being a mutual

[24]Such a dictionary would actually be huge to begin with: it would be 2^{48} bits, which equals roughly 32 TB.

contact. In the submission phase A sends v_{BA} to the server, who stores it in S. In the query phase that follows B sends v_{BA} to the server, who detects that this value already is in S.

Security To what extent does the protocol satisfy definition A.1? Note that A only queries the server in the query phase for $B \in contacts_A$, so security is maintained if that condition doesn't hold. However, if $B \in contacts_A$ while $B \notin \mathscr{M}$, then an adversarial server X simply can reply yes and pretend there is a match to any of A's queries. An adversarial member X can compute and send v_{AB} in the first phase (even when $A \notin contacts_B$). In the second phase, given that $B \in contacts_A$, A will send v_{AB} to the server. The server replies that there is a match, and hence $B \in out_A$. Security is therefore only guaranteed when $B \notin contacts_A$.

Membership privacy For membership privacy (see definition A.2) we need to prove that any entity $X \notin contacts_A$ does not learn whether $A \in \mathscr{M}$, for any $A \neq X$ of its choosing. In fact, membership privacy only holds in a limited sense, when the adversarial entity X is a member or the server. X needs to guess a Y such that $Y \in contacts_A$, and then compute v_{YA}. If X is an ordinary member it needs to send this value to the server in the query phase, hoping that the server returns true because $v_{YA} \in S$ as indeed $Y \in contacts_A$. If X is the server it can simply test this directly. By the properties of the KDF used to compute v_{YA}, X has only a limited number of tries, making it infeasible in general.

Contact privacy The protocol does not satisfy contact privacy (definition A.3): any member (or the server) can construct v_{AB} and either inspect S (the server) or submit it in the query phase (any other member) to test whether $A \in contacts_B$.

It would satisfy contact privacy in a weaker sense where the adversary would be required to find A and B such that $A \in contacts_B$. As this involves testing several guesses, the success of the adversary would be limited depending on its knowledge of possible contacts and the amount of resources it is willing to spend.

B.2 Final comments

Observe that the server learns the total number of matches. Because a fresh anonymous circuit is used to submit or query a value, the server does *not* learn how many matches a particular member has.

Also observe the use of a key derivation function: it prevents the adversary from trying out all possible combinations of inputs (which would have been feasible when using an ordinary hash function instead), especially because the input to the key derivation function in this protocol is the concatenation of two identifiers, whereas traditionally only a single identifier is hashed. This increases the uncertainty for the adversary considerably.

All in all we stress that even given the obvious shortcomings of this protocol, it is more privacy friendly than the hash-based approach for normal, one-sided contact discovery, for several reasons.

1. To be discovered by another member A, B needs to have A as a contact, or A needs to guess a contact of B.

2. It therefore prevents adversaries from discovering members if little prior knowledge about contacts is known.

3. In particular it prevents the server from learning the identifiers of non-members or recovering the social graph from the information that it receives.[25]

On the other hand, this particular mutual contact discovery does have a significant privacy leak: any member can test whether $B \in contacts_A$. This is typically not possible (by definition) in one-sided contact discovery protocols, as information about $contacts_A$ has no bearing on the information exchanged between any other member and the server. In other words there is an inherent trade-off between one-sided contact discovery and mutual contact discovery: the first protects contact privacy better, while the latter protects membership privacy better.

Finally note that many of the weaknesses in this protocol are caused by the fact that any member can compute and submit/query v_{AB} for arbitrary A and B. The use of certified identifiers by the protocol in the main body of this paper makes this impossible. An alternative, more efficient, approach that prevents an arbitrary member to compute such a value is used in the next protocol section.

C Protocol using a key server

The crucial observation is that in order to prevent the server to verify a guess, the values used to discover a connection need to involve a secret that the server does not know. As it turns out, that same secret can be used to ensure that users cannot verify a guess for a connection between two arbitrary users. The same insight underlies the main certified identities protocol presented earlier. The following protocol is inspired by, but different from, older match making protocols [1, 30].

Assume there is a key server, independent from the matching server. This is not an unrealistic assumption, as most messaging services run such a key

[25]In all fairness it should be mentioned that even for one-sided, traditional, contact discovery the process can be made such that the identity of the entity starting the contact discovery process remains anonymous, therefore also preventing the reconstruction of the social graph by the server.

server anyway to allow users to exchange messages end-to-end encrypted. Alternatively, one could leverage the existing infrastructure of mobile phones (or any other existing infrastructure that provides the identifiers used to discover contacts) and let the mobile network operator maintain such a key database.

Let $K_A = g^{k_A}$ for some generator g over a group in which the Computational Diffie-Hellman (CDH) problem is hard [11]. Let user A generate such a key pair k_A, K_A when joining the system, and submit its public key K_A to this key server. We assume the key server can verify the identity of users submitting keys (to prevent the adversary from submitting keys for users). When asked for a key of a user that does not exist, the key server generates a random public key and stores it for that user.[26] Subsequent requests for the key of that same (unenrolled) user return the same key. All requests for keys should take the same time to serve. This way, users cannot use the key server to test whether users are enrolled or not.

Consider the protocol using certificates from section 4. Instead of certificates we use public keys to compute the necessary tokens. That is, we define

$$T_{AB} = {K_A}^{k_B} = g^{k_A k_B} \ .$$

and run the protocol as before, with the following modification: when each member M, for each $A \in contacts_M$, computes T_{AM} it first retrieves K_A from the key server.

C.1 Performance

Note how the server is not involved in any complicated computations: it simply stores and looks up values, and answers queries. From the server side this is as good as it could possibly get.

In each round, a user B needs to create a key pair (as explained this involves one exponentiation in a prime-order subgroup of $\mathbb{Z}_p^*$, where p itself is prime) and for each $A \in contacts_B$ retrieve K_A from the key server and compute $T_{AB} = {K_A}^{k_B}$ (again one exponentiation in a prime-order subgroup of $\mathbb{Z}_p^*$). This is some work, but not excessive: the private set intersection protocols discussed in section 2 each require the user to do at least the same amount of work.[27]

C.2 Informal analysis

We again argue the protocol satisfies all required properties in an informal fashion here.

As all network communication is assumed to be anonymous, the key server does not learn the identity of the user requesting a particular public key. As

[26]We assume the key server does not collude with the matching server, or else the matching server may be able to detect contacts of unenrolled users as it controls the private key.

[27]For example, the OPRF construction used in the efficient protocol of Chen *et al.* [7] also uses an exponentiation for each user input.

a result the key server cannot reconstruct the social graph. It does learn the number of people interested in contacting a particular someone, though. It is for this reason that we settled for the certified identities based protocol as the main protocol because the issuer is offline and therefore limits the risk of tracing users.

In what follows, we again only consider a single adversarial user or a single adversarial server. As will become clear during the analysis, collusion among several adversarial entities does not help them.

Note that no Diffie-Hellman instance (i.e., T_{AB} in the protocol) is ever sent in the clear. This in turn implies that the Decisional Diffie-Hellman problem is irrelevant here and that we only need to rely on the hardness of the Computational Diffie-Hellman (CDH) problem.

Correctness Correctness immediately follows from the fact that $T_{AB} = T_{BA}$ and the analysis of the original protocol in section 4.

Security The protocol satisfies definition A.1. For suppose $B \in contacts_A$ but $B \notin \mathscr{M}$. Then adversarial X (whether it is the server or a user different from B) cannot forge the value $H_2(T_{AB}, B)$ unless it knows T_{AB}, which is never revealed. Because of the CDH assumption it also cannot compute it.[28]

Membership privacy The protocol satisfies membership privacy (see definition A.2). Consider an adversarial $X \notin contacts_A$, for any $A \neq X$ chosen by X. To test that $A \in \mathscr{M}$, X needs to guess a non corrupted B such that $B \in contacts_A$ and then test this choice. It can only do so by sending $H_2(T_{AB}, P)$ to the server in the query phase. However, computing $H_2(T_{AB}, P)$ requires knowledge of eiter A's or B's secret, so this is impossible. We conclude that X does not learn whether $A \in \mathscr{M}$.

Contact privacy The protocol satisfies contact privacy (definition A.3). To test whether $A \in contacts_B$, the adversary needs to compute $H_2(T_{AB}, P)$ and either query the server, or (when the server itself is adversarial) test membership in S. However, computing T_{AB} requires knowledge of either A's or B's secret, so this is impossible.

C.3 Final comments

As in the first protocol the matching server learns the total number of matches. (When a fresh anonymous circuit is used to submit or query a value, the match-

[28]Note that this only holds if we assume that the matching server and the key server do not collude; if not, then for unenrolled user B the servers actually know the private key and *can* compute T_{AB}, and as a result convince $B \in out_A$ for those A that have $B \in contacts_A$. So if the key server and the matching server collude, protection is limited. The same caveat applies to membership and contact privacy.

ing server does not even learn how many matches a particular user has.)

The key server learns the number of members and their identity. As discussed before, we need to assume queries to the key server are anonymised (shielding the network address of the requestor) to prevent the key server from also reconstructing the full social graph. The key server does learn the number of users that have a particular user in their contact list, i.e., it learns how popular (or unpopular) a user is, by counting how many times a particular key is requested. To prevent the key server from learning this, an (inefficient) approach would be to use private information retrieval [9]. In any case, note that this is about all the key server learns.

Observe that if the main protocol using certified identifiers would rely on a separate certificate issuer, this issuer also learns the identities of all participating members.

D Discussion

Compared to the protocol used by Signal (that essentially matches hashes of phone numbers found in contact lists), the protocols discussed in the appendix should perform with similar performance. In both our protocols the matching server only performs simple store and lookup operations. User side we expect only a constant factor slowdown (depending on the time required to compute the key derivation function) as both our protocols also only hash (using the key derivation function) entries in the contact lists, and the second protocol requires only a single exponentiation for each entry in the contact list.

We wish to stress that the definition of membership privacy only restricts members that do *not* appear in the contact list of a member B to discover whether B is actually a member of the service. Even in the second protocol it is still possible that a member A guesses a B with $A \in contacts_B$ and then test this guess by sending $H_2^<(T_{AB}, P_A, P_B)$ to the server in the query phase. A's luck depends on its prior knowledge of such B. Note though that A has only a limited number of tries, making it infeasible in general. We do not consider this an attack as the fact that B included A as a contact signals that B allows A to discover them. (Modulo the discussion in section 1.)

Regarding contact privacy for the first protocol note that the definition of contact privacy is strong. A weaker version of contact privacy that requires the adversary to find a pair A, B such that $B \in contacts_A$ is actually satisfied by the first protocol in a limited sense (depending on its knowledge and the amount of times it is willing to evaluate the key derivation function).

The first protocol prevents adversaries from recovering contacts if little prior knowledge about contacts is known, and in particular limits the ability of the server from recovering part of the social graph from the information that it receives. The second protocol significantly improves the privacy protection offered, because it also prevents an active adversarial server or third party to test even a suspicion of two members being a contact. To do so, the use of a key server is required. Note that a distinct advantage of the first protocol discussed in this appendix is that it does not rely on such an infrastructural assumption.