David Chaum, Mario Yaksetig*, Alan T. Sherman, and Joeri de Ruiter

UDM: Private User Discovery with Minimal Information Disclosure

Abstract: We present and analyze UDM, a new protocol for user discovery in anonymous communication systems that minimizes the information disclosed to the system and users. Unlike existing systems, including those based on private set intersection, UDM learns nothing about the contact lists and social graphs of the users, is not vulnerable to off-line dictionary attacks that expose contact lists, does not reveal platform identifiers to users without the owner's explicit permission, and enjoys low computation and communication complexity.

UDM solves the following user-discovery problem. User Alice wishes to communicate with Bob over an anonymous communication system, such as cMix or Tor. Initially, each party knows each other's public contact identifier (e.g., email address or phone number), but neither knows the other's private platform identifier in the communication system. If both parties wish to communicate with each other, UDM enables them to establish a shared key and learn each other's private platform identifier.

UDM uses an untrusted user-discovery system, which processes and stores only public information, hashed values, or values encrypted with keys it does not know. Therefore, UDM cannot learn any information about the social graphs of its users. Using the anonymous communication system, each pair of users who wish to communicate with each other uploads to the user-discovery system their private platform identifier, encrypted with their shared key. Indexing their request by a truncated cryptographic hash of their shared key, each user can then download each other's encrypted private platform identifier.

Keywords: anonymous communication systems, cMix, contact discovery, contact lists, social graphs, user discovery, User Discovery with Minimal information disclosure (UDM).

DOI Editor to enter DOI
Received ..; revised ..; accepted ...

David Chaum: xx.network, USA, E-mail: david@chaum.com
***Corresponding Author: Mario Yaksetig:** University of Porto, E-mail: mario.yaksetig@fe.up.pt
Alan T. Sherman: Cyber Defense Lab, CSEE Department, University of Maryland, Baltimore County (UMBC), Baltimore, MD 21250 USA, E-mail: sherman@umbc.edu
Joeri de Ruiter: SIDN Labs, NL, E-mail: joeri.deruiter@sidn.nl

1 Introduction

To protect the confidentiality of messages, some communication systems—such as Signal [1] and WhatsApp [2]—offer end-to-end encryption services. End-to-end encryption, however, does not protect against traffic analysis, in which an adversary monitors observable characteristics of the messages (e.g., time, size, file type) and who is communicating with whom. Furthermore, anonymous communications systems—such as cMix [3] and Tor [4]—protect against traffic analysis but fail to provide services for private user discovery. For a user to learn the private platform identifiers of intended recipients—which are, for example, used when sending and receiving messages through the anonymous communication system—most existing messaging systems force users to disclose their contact lists to the system. Some systems are vulnerable to off-line dictionary attacks that enable the system or anyone to learn the contact lists or even social graphs of users. Other systems offer more privacy, but at the cost of high computation and communication complexity.

For example, exhibiting an egregious lack of concern for user privacy, Telegram [5] facilitates user discovery as follows. With permission from the user, Telegram uploads the user's entire contact list in plaintext and stores it. By contrast, Signal [1] is one of the most popular communication systems that purports to provide privacy. Their 2014 user-discovery protocol [6], however, has high computation and communication complexity, due to its use of an RSA-based [7] Bloom filter [8]. Signal's 2017 user-discovery protocol [9] uploads the user's contact list to the system and requires trust that secure hardware does not leak information. Silent Circle's [10] set intersection method is vulnerable to an off-line dictionary attack that exposes the contact list to the system.

Private user discovery is important to protect contact lists, social graphs, and who is communicating with whom. Such data are attractive targets for criminals, and protecting them is vital to democracy and the way of life many people desire.

Seeking greater privacy for communicants, we present a new private user-discovery protocol, *User Discovery with Minimal information disclosure* (UDM), with low computation and communication complexity. Invented by Chaum in

June 2016,[1] and first analyzed by Yaksetig [11], and unlike existing systems, UDM distinguishes between a user's *public contact identifier (public ID)*, such as an email address, and their *private platform identifier (private ID)*, for example, a 128-bit random string. UDM enables a registered user of a communication system to discover the private ID of someone from their public ID, provided this person grants permission for its disclosure to the user. In our protocol, no one—including users, external attackers, and the user-discovery system—can learn the private ID associated with any user's public ID, except when authorized by the user associated with the private ID. In addition, no one can link any two other distinct registered members, at least one of whom lists the other in their contact list. UDM works by encrypting and cryptographically hashing in novel ways.

The rest of this paper defines the user-discovery problem, gives an overview of UDM, reviews and compares user-discovery systems and approaches, explains our adversarial model, presents our UDM protocol, comments on its privacy, describes our prototype implementation, discusses various issues involving UDM, illustrates useful applications of UDM, and presents our conclusions.

The main contributions of this paper are: (1) a definition of the user-discovery problem, (2) a new protocol UDM for solving it, (3) a preliminary analysis of UDM, (4) a comparison of the characteristics of UDM with those of several existing user-discovery mechanisms, and (5) a discussion of selected applications for private user discovery.

2 Problem Specification

The user-discovery problem can be defined in terms of users and their contact identifiers, communication channels, inputs, outputs, usability, and computational, implementation, and privacy requirements.

There is a universe of *users*, each of whom has a *public ID* (e.g., email address or telephone number) and a means of communicating with each other over an insecure channel (e.g., email), addressing messages using these public IDs. All users know, or could learn, each other's public IDs. Each user has a list of the public IDs of the users they know; we call this list the user's *contact list*. The transitive closure of this list under the contact relation is the user's *social graph*.

A subset of these users are registered members of some *Communication System* in which they address messages using their *private IDs* (e.g., a random string). We assume this communication system provides anonymous communications, such as those provided by cMix [3] or Tor [4]. Each registered member has and knows their own private ID. They can also establish a connection with the communication system's *User-Discovery System*, in which connection the user can authenticate the system.

The *user-discovery problem* is for a registered user, say Alice, to learn the private ID of another registered user, say Bob, identified by their public ID, provided Bob consents to disclosing his private ID to Alice. In the basic form of the problem, we assume that Alice and Bob already know each other in the sense that they list each other's public ID in their contact list.

Importantly, there are two privacy requirements:

1. No one—including users, external attackers, and the User-Discovery System—should learn the private ID associated with any other user's public ID, except when authorized by the user associated with the private ID.
2. No one—including users, external attackers, and the User-Discovery System—should be able to link any two other distinct registered members, at least one of whom lists the other in their contact list. Here, "linking" means finding for any two users, say Alice and Bob, any *contact identifier pair* (a, b), where a is the public ID or private ID for Alice, and b is public ID or private ID for Bob.

Furthermore, we seek solutions that are easy to use, relatively easy to implement, and have low computation and communication complexity (transmit a small number of bits).

3 Overview of UDM

In UDM, the User-Discovery System comprises a Public-Key Manager and Encrypted ID Manager. Suppose that Alice and Bob are registered with the anonymous communication system and with UDM, and that Alice lists Bob's public ID in her contact list. Alice wishes to learn the private ID of Bob for the anonymous communication system. As summarized in Figure 1, the UDM protocol works in six steps:

0. If Bob does not list Alice's public ID in his contact list, or if Bob is not registered with the communication system and UDM, then Bob must register with the communication system and UDM, and Bob must add Alice as a contact. Alice could prompt Bob to do so through an out-of-band communication, for example, in person, by telephone, or through the insecure channel.

1 Private correspondence dated June 6, 2016.

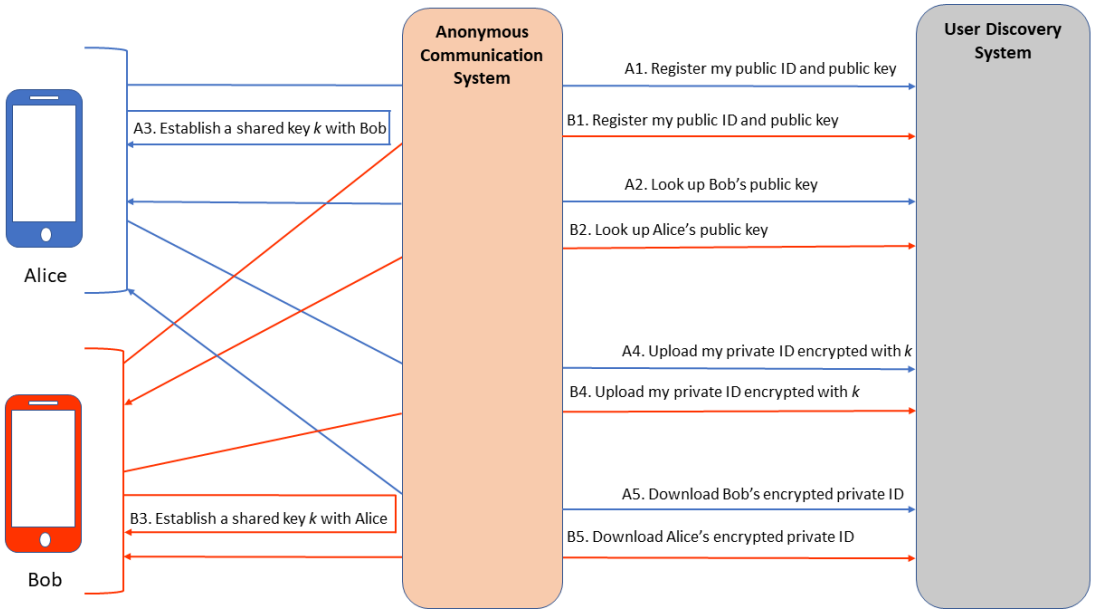

Fig. 1. Summary of the UDM process. For Alice and Bob to learn each other's private ID, Alice and Bob register with the User-Discovery System, establish a shared key with each other, and exchange their encrypted private IDs encrypted with the shared key. All communications take place over an anonymous communication system.

1. Using the anonymous communication system and with the help of UDM's User-Discovery System, Alice and Bob learn each other's UDM public keys associated with their public IDs.
2. Without disclosing any secrets to each other, and using their UDM public keys, Alice and Bob establish a shared symmetric encryption key. Each does so asynchronously by performing a calculation on the other's public key.
3. Alice sends to the User-Discovery System her private ID encrypted under the shared key, together with a truncated hash of the shared key. The User-Discovery System stores the ciphertext in a table indexed by the associated truncated hash value.
4. If Bob does not wish to proceed, he stops. Otherwise, Bob sends to the User-Discovery System his private ID encrypted under the shared key, together with a truncated hash of the shared key.
5. From the User-Discovery System, and using the truncated hash, each party can retrieve the other's encrypted private ID.

With the help of a UDM app running on each user's trusted device, Steps 2-3 happen automatically, provided Alice and Bob include each other in their contact list.

Throughout this process, the User-Discovery System never learns anyone's private ID or shared key. Furthermore, the system cannot link any two distinct registered members, at least one of whom lists the other in their contact list.

4 Previous User-Discovery Systems and Approaches

Existing communication systems perform user discovery in ways that disclose contact lists and social graphs or reveal social contacts (links between two users), to users, external attackers, or the system. Some are vulnerable to off-line dictionary attacks that learn such information, or carry out large computations or transmit large amounts of data. We briefly review selected previous user-discovery methods; we discuss other possible approaches to user discovery; and we compare these methods and approaches to UDM.

4.1 Previous User-Discovery Systems

Telegram [5] exhibits a disturbing lack of concern for user privacy. After a user installs the app on their personal mobile device, with permission from the user, Telegram uploads the user's entire contact list and stores it permanently on its server. Implementing a commonly used "set intersection" strategy, Telegram also returns those contacts (including their telephone

numbers) who are registered users. Whenever someone registers, Telegram notifies each user in the registrant's contact list that this friend has joined. Telegram also provides a search capability to find any user. Favoring functionality and simplicity over user privacy, Telegram can learn every user's contact list.

The extremely popular *WhatsApp* [2] messaging system also implements a set intersection strategy, which exposes each user's contact list to the system. With the user's permission, WhatsApp uploads the user's entire contact list and returns those contacts who are registered users.

WeChat [12] follows a strategy similar to Telegram's and WhatsApp's. Each user can upload their contact list and link it to their telephone number.

Skype [13] allows users to search the entire database of users who have not opted out. There is no mechanism to find users who have opted out. If Alice shared her contact details with Bob before opting out, Bob could still contact Alice. Skype also supports a set intersection strategy. A user may create a Skype contact list and upload it to learn which contacts are registered with the system. Curiously, and with some potentially adverse privacy implications, Alice can already send messages to Bob before Bob accepts Alice as a contact.

Zoom [14] enables any user, say Alice, to add a contact by submitting their public contact identifier (i.e., email address). Alice may also connect her Gmail or Facebook account to her Zoom account and thereby submit all of her contacts to Zoom. If the submitted user, say Bob, is not registered, then Zoom sends an email to Bob inviting him to join. If Bob is registered, then Zoom sends a "friend request" from Alice to Bob, inviting him to accept the request. The system informs Alice of Bob's decision. Even after receiving a rejection, Alice may repeat the friend request any number of times. Although not yet implemented, Zoom released specifications for a new way to detect misbehaving Zoom servers (see Section 7.7).

Keybase [15] permits users to search the database of registered users, or on mobile clients, to upload their contact list and learn which of those contacts are registered on the platform. Users may "follow" other users. For any user, say Alice, anyone may see the users whom Alice follows and the users who follow Alice.

Signal's [1] 2014 and 2017 user-discovery protocols protect user privacy the most in comparison with all existing user-discovery systems in use. They also represent the most popular of the systems that try to protect user privacy. Nevertheless, each has serious limitations. Using Bloom filters with RSA signatures, *Signal's 2014* user-discovery protocol fails to scale to a significant user base. Each user downloads an entire encrypted Bloom filter containing the server's RSA signature for each registered contact in the system. Users upload to the server a blinded query hiding the value of each telephone number in their contact list. The Signal server blindly signs [16] each uploaded value and returns a valid RSA signature on each. Users receive these signatures, unblind the values, and perform local queries to check whether or not the contacts are in the Bloom filter.

For a large user base (e.g., 10 million), Bloom filters are relatively large (40 MB). Therefore, this strategy does not scale well: it would require clients to perform a considerable download frequently on their mobile devices.

Instead of downloading a large Bloom filter, *Signal 2017* [9] allows users to query a secure enclave operated by Signal in the cloud. This approach relies on trusted hardware and the capability of users to verify that the server is executing the correct software. However, several attacks (e.g., Meltdown [17], Spectre [18], and Foreshadow [19]) threaten to leak confidential data from the enclave. Regardless, the user still uploads their contact list in the query stage to allow the secure enclave to intersect it with its local records and return which users are registered in the system.

Zimmermann's *Silent Circle* [10] introduces a protocol based on hashing the private platform identifiers (telephone numbers) before sending them to the server. Each user uploads a highly truncated (at least four bits) hash value of each platform identifier in their device. The server responds with less truncated hash values of those uploaded values that correspond to registered users. The user then compares the less truncated values with the full hash values. The user assumes that, if these values match, then the corresponding user is registered, which introduces a false-positive error rate in detecting membership. Because telephone numbers are relatively short, an adversary could precompute the truncated hash value of all possible telephone numbers. In doing so, anyone (including the system) who learns the transmitted lists of hashed identifiers learns the user's contact list and which list members are registered in the system.

4.2 Other Approaches to User Discovery

Other possible approaches to private user discovery include methods based on *Private Set Intersection (PSI)* and *Private Information Retrieval (PIR)*. When comparing systems, keep in mind that UDM leverages an anonymous communication system, which incurs its own setup and communication costs, whereas most other user-discovery systems do not take advantage of such a communication system.

PIR-PSI [20] (2018) combines PIR and PSI techniques with two servers that store their data sets in a Cuckoo table [21] to allow clients to calculate the set intersection. Building on an existing PIR method [22, 23], this design allows one server to learn the PIR output, blinded by masks known to the client. Consequently, the server and the client can perform a stan-

Table 1. Selected user-discover systems and their characteristics. Many of the systems use some type of "set intersection strategy," in which the user uploads their contact list, and the system responds with those contacts on the list who are registered with the system. Only in UDM is the platform identifier (private ID) different from the public contact identifier (public ID, such as an email address). Rating codes: ⊕ good, ⊙ fair, ⊖ poor.

System	Year	Method	System receives contact list	Reveals platform identifiers to users	Amount of computation	Number of bits transmitted
Skype	**2003**	**transparent search and set intersection**	⊖ **yes**	⊖ **yes**	⊕ **low**	⊕ **low**
WhatsApp	**2009**	**set intersection**	⊖ **yes**	⊖ **yes**	⊕ **low**	⊕ **low**
WeChat	**2011**	**set intersection**	⊖ **yes**	⊖ **yes**	⊕ **low**	⊕ **low**
Zoom	**2011**	**private search and set intersection**	⊖ **yes**	⊖ **yes**	⊕ **low**	⊕ **low**
Telegram	**2013**	**set intersection**	⊖ **yes**	⊖ **yes**	⊕ **low**	⊕ **low**
Signal	**2014**	**RSA-based encrypted Bloom filter**	⊕ **no**	⊙ **yes (b)**	⊖ **high**	⊖ **high**
Keybase	**2014**	**transparent search and set intersection**	⊖ **yes**	⊖ **yes**	⊕ **low**	⊕ **low**
Silent Circle	**2015**	**set intersection with truncated hashes**	⊙ **yes (a)**	⊙ **yes (b)**	⊕ **low**	⊕ **low**
Signal	**2017**	**user searches database in secure enclave in cloud**	⊖ **yes**	⊖ **yes**	⊕ **low**	⊕ **low**
PIR-PSI	**2018**	**private information retrieval & private set intersection**	⊙ **yes (a)**	⊙ **yes (b)**	⊖ **high**	⊖ **high**
MPCD	**2019**	**unbalanced private set intersection**	⊕ **no**	⊙ **yes (b)**	⊖ **high**	⊖ **high**
UDM	**2016**	**platform identifier encrypted by shared key**	⊕ **no**	⊕ **no**	⊕ **low**	⊕ **low**

	Comments
a	**Dictionary attack reveals contact lists.**
b	**Dictionary attack reveals platform identifiers.**

dard two-party PSI on masked values. This approach requires the assumption of multiple non-colluding servers and results in a relatively high communication complexity. By comparison, leveraging an anonymous communication system, UDM tolerates colluding managers, while still providing low communication complexity.

Mobile Private Contact Discovery at Scale (MPCD) [24] (2019) optimizes existing off-line/on-line unbalanced PSI techniques by applying new forms of correlated random *oblivious transfer (OT)* [25] precomputation, and introducing a compression method for Cuckoo filters [21] that allows the protocol to reduce the required network communication. In the off-line stage, the server performs symmetric cryptographic operations for every element in the corresponding set. The result of this phase is a compact Cuckoo filter of the *pseudorandom function (PRF)* [26] evaluation of all users in the system, which MPCD sends to the user. At this point, the client and the server can run an oblivious PRF evaluation protocol, where one input is the client's search string, and the other input is the PRF key the server used in the off-line step. This process ensures that the client learns only the PRF output value and that the server learns nothing about the search string. Thus, the client checks only if a specific entry is present in the server's set. The client can then check the received PRF output against the Cuckoo filter obtained in the setup phase. If the element is in the Cuckoo filter, then the contact associated with that search string is a registered user.

MPCD requires a setup phase, in which the server must perform a large amount of computation to be able to provide users with the required information for local searches. Therefore, MPCD incurs a heavy computational cost in comparison to UDM, which has no setup phase.

4.3 Comparison of User-Discovery Systems

Table 1 lists and compares the characteristics of several selected systems.

5 System Architecture

We describe the UDM architecture in terms of its components and roles. As shown in Figure 2, the UDM system involves registered users and two untrusted managers—a Public-Key Manager and an Encrypted ID Manager, which compose the user-discovery system. All communications among these roles take place via an underlying *anonymous communication system*, such as cMix [3] or Tor [4].

Each *registered UDM user* U has a *private ID* I_U, which they use in the anonymous communication system. In addition, each user U has a *public ID* J_U (e.g., email address, legal name, pseudonym), which they might use in an available insecure communication system (e.g., email). Using the anonymous communication system, each user can establish a session with either manager, in which the user can authenticate the manager.

Together, the Public-Key and Encrypted ID Managers implement a *user-discovery system*. Conceptually, it is convenient to view this system with two separate roles; in practice one might choose to implement them together or separately. The *Public-Key Manager* keeps track of the public key associated with each public ID. The *Encrypted ID Manager* keeps track of encrypted private IDs, encrypted with shared keys unknown to the managers.

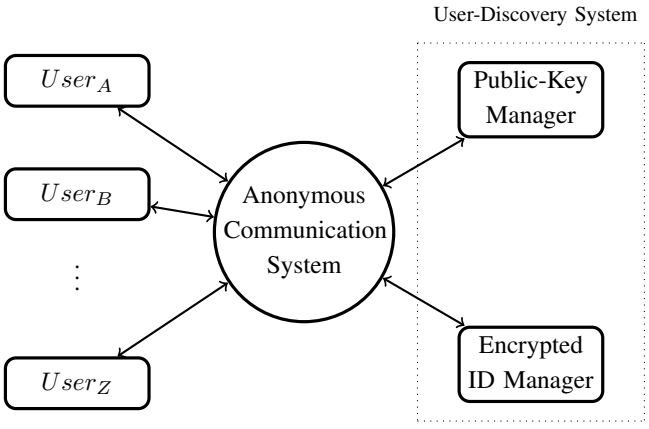

Fig. 2. UDM Architecture. Users interact with a Public-Key Manager and an Encrypted ID Manager through an anonymous communication system.

6 Adversarial Model

The adversary could be anyone, including users, the Public-Key Manager, the Encrypted ID Manager, or an external attacker. The adversary's goals are: (1) to learn the private ID corresponding to the public ID of any user, and (2) to link any two users who list each other in their contact list.

Between all communicants in the underlying anonymous communication system, we assume channels are protected for authentication, confidentiality, and integrity. Also, inputs and outputs of this system cannot be linked. The adversary cannot defeat standard cryptographic functions, including encryption, hashing, and signatures.

We assume that users have some way to authenticate themselves during the UDM registration process to ensure that their UDM public key is correctly associated with their public ID. We also assume that users can authenticate the managers. To detect misbehaving managers and dishonest users who enter incorrect registration information, one option is periodically for users to verify that the user-discovery system correctly returns the UDM public key associated with their public ID. To obtain some assurance that the user is in control of the given public ID, another option, albeit weak, is for the system during registration to send a verification message to the user's public ID via the insecure channel.

We make no security assumptions about communications that take place outside of the anonymous communication system. In particular, for any such communications that are not protected, the adversary can eavesdrop, modify, or inject messages. Furthermore, the adversary can delay or stop messages and observe the traffic flows. That is, we assume a Dolev-Yao attacker [27] in the insecure channel.

The managers are untrusted. The managers, however, are *cautious* in the sense that they do not wish to be caught engaging in any improper activity. We do not consider denial-of-service attacks and assume that suitable protections are in place to mitigate at least flooding attacks.

We assume that the user operates from a trusted device, perhaps a smartphone. We assume further that a trusted app running on the trusted device supports the UDM system. The trusted device holds the user's keys and contact list.

7 System Design

We explain how UDM works by describing its cryptographic elements, the two tables maintained by the User-Discovery System, how users register with the system, how any pair of users can establish a shared key, how users can upload and retrieve encrypted private IDs, how users can check that the untrusted managers are acting correctly, and how the UDM app supports these processes. To protect user privacy, each table includes only public values, hashed public values, or hashed or encrypted private values. Figure 3 illustrates the process.

7.1 Cryptographic Elements

UDM uses a *symmetric encryption function* E and a *cryptographic hash function* H. The notation $E[k, I]$ denotes the symmetric encryption of private ID I under key k. The encrypted ID manager also uses a *truncated hash function* $\hat{H}$ (e.g., the first 128 bits of H's output).

In addition, users engage in an *asynchronous key-establishment process* over the anonymous communication system. Each user U has a *UDM public-private key pair* (p_U, s_U) that is used only in the key-establishment process. Only the user knows their UDM private key.

7.2 Public-Key Manager

The Public-Key Manager maintains a table of public keys for the UDM registered users (Table 2). Known as the "blind directory," this table is indexed by a cryptographic hash (fingerprint) of the corresponding public ID.

Table 2. The blind directory of public keys of registered users stored by the Public-Key Manager, indexed by a hash of the public ID.

Fingerprint of Public ID	Public Key
$H(J_B)$	p_B
$H(J_A)$	p_A
$H(J_C)$	p_C
...	...

7.3 Encrypted ID Manager

The Encrypted ID Manager stores a table of encrypted private IDs for pairs of communicants who have established a shared key (Table 3). In this table, which is known as the "encrypted address database," each private ID is encrypted by the corresponding shared key. Indexed by a truncated cryptographic hash $\hat{H}$ of the shared key, the second column of the table stores the encrypted ID of the communicant who *first* uploaded the given truncated hash value. The third column, if present, stores the encrypted ID of the communicant who uploaded the given truncated hash value *second*. The truncated hash provides a way for the two communicants to upload and download their encrypted private IDs, without the manager learning who they are.

Table 3. The encrypted address database of encrypted private IDs stored by the Encrypted ID Manager, encrypted by a shared key, and indexed by a truncated cryptographic hash of the shared key. Only the two communicants know their shared key.

Fingerprint of Shared Key	Encrypted First Uploaded Private ID	Encrypted Second Uploaded Private ID
$t_{A,B} = \hat{H}(k_{A,B})$	$E[k_{A,B}, I_A]$	$E[k_{A,B}, I_B]$
$t_{C,A} = \hat{H}(k_{C,A})$	$E[k_{C,A}, I_C]$	$E[k_{C,A}, I_A]$
$t_{A,D} = \hat{H}(k_{A,D})$	$E[k_{A,D}, I_A]$	-
...	...	...

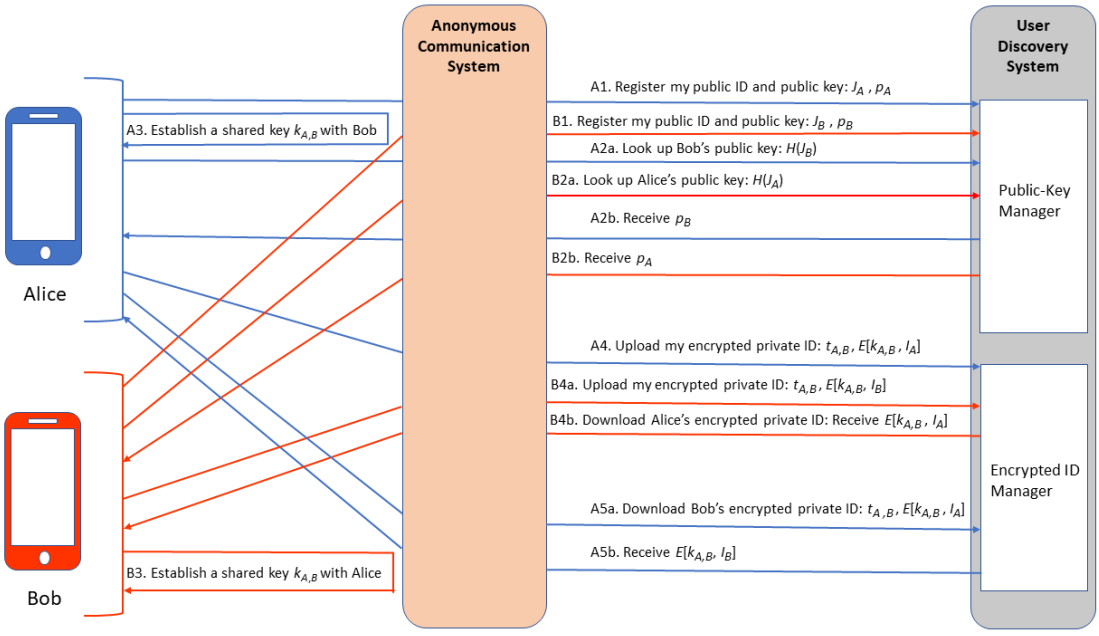

Fig. 3. The UDM process. Alice and Bob learn each other's private ID by interacting with an untrusted Public-Key Manager and an untrusted Encrypted ID Manager. Initially, each user registers with UDM. Then, they establish a shared key and exchange their private IDs encrypted with this shared key. All communications take place over an anonymous communication system. In Steps 4–5, $t_{A,B}$ is a truncated hash of the shared key. H is a cryptographic hash function, and E is symmetric encryption. In this figure, we assume that Alice uploads her encrypted private ID before Bob uploads his.

7.4 UDM Registration

To register with UDM, Alice generates a UDM public-private key pair (p_A, s_A). Using the anonymous communication system, Alice establishes a session with the Public-Key Manager, in which Alice authenticates the manager. Alice sends to the manager (J_A, p_A), where J_A is her public ID. The manager enters $(H(J_A), p_A)$ into the table of public keys. Behold that the manager does not learn anything about Alice other than (J_A, p_A); in particular, the manager does not necessarily learn Alice's name or IP address. Alice uses her UDM public key p_A only to establish shared keys with other UDM users, using a Diffie-Hellman key-exchange [28] process.

Separately, Alice registers with the anonymous communication system by establishing a private ID I_A, which she does not disclose to UDM. We assume that the only other information Alice provides to the anonymous communication system is her public key for that system, which is needed for setup and certain functionality in that system, and which is separate from her UDM public key.

7.5 Establishing a Shared Key

Assume Alice and Bob are registered with the anonymous communication system and with UDM. Assume also that Alice and Bob know each other in the sense that they list each other's public ID in their contact list. Suppose that Alice wishes to communicate with Bob over the anonymous communication system. We now describe how they establish a shared symmetric key $k_{A,B}$ known only to them.

If Bob is not registered with the anonymous communication system and UDM, then Alice could prompt Bob to register by sending him an out-of-band message, for example, over the insecure channel, from J_A to J_B including her public ID J_A.

With the support of the UDM app running on each user's trusted device, automatically, the following two steps are carried out between each pair of registered UDM users who know each other, including Alice and Bob. In particular, if Bob was not registered, Alice's app periodically checks with the manager if Bob completed his registration.

First, Alice and Bob learn each other's UDM public keys from their public IDs as follows. Using the anonymous communication system, Alice sends the request $H(J_B)$ to the Public-Key Manager. If Bob is registered with UDM, the manager responds by sending Bob's public key p_B to Alice. If Bob is not registered with UDM, the manager generates, sends, and remembers a randomly generated public key to Alice. Similarly, Bob learns Alice's public key from the Public-Key Manager.

Second, Alice and Bob compute a shared key asynchronously using the other's UDM public key and their own UDM private key. For example, using a Diffie-Hellman key exchange, Alice can compute $k_{A,B} = k_A = (p_B)^{s_A}$, where s_A is Alice's UDM private key. Similarly, Bob can compute $k_{A,B} = k_B = (p_A)^{s_B}$, where s_B is Bob's UDM private key. For key exchange based on discrete logarithms, all computations are performed modulo q for some public prime q. Only Alice and Bob know their shared key.

7.6 Uploading and Retrieving Encrypted Private Identifiers

Assume that Alice and Bob have established a shared key $k_{A,B}$ with each other, which they associate with their public IDs J_A and J_B. Supported by a UDM app running on their trusted devices, Alice and Bob proceed as follows.

First, using the anonymous communication system, say, Alice sends $(t_{A,B}, E[k_{A,B}, I_A])$ to the Encrypted ID Manager, where I_A is Alice's private ID and $t_{A,B}$ is a truncated hash of $k_{A,B}$. If there is no entry in the manager's table for $t_{A,B}$, the manager enters $t_{A,B}$ and $E[k_{A,B}, I_A]$ into the first and second columns of the table, respectively.

Second, if Bob wishes to share his private ID I_B with Alice, Bob sends $(t_{A,B}, E[k_{A,B}, I_B])$ to the Encrypted ID Manager. If there is already exactly one entry in the table at row $t_{A,B}$, and if this entry is not $E[k_{A,B}, I_B]$, the Manager sends the contents of the second column at row $t_{A,B}$ to Bob. In this case, the contents of the second column are $E[k_{A,B}, I_A]$. The Manager enters $E[k_{A,B}, I_B]$ into the third column of the table at row $t_{A,B}$.

Third, Alice periodically sends $(t_{A,B}, E[k_{A,B}, I_A])$ to the Encrypted ID Manager to determine if the table at row $t_{A,B}$ has two entries. If it does, and if the second column has the value $E[k_{A,B}, I_A]$, the Encrypted ID Manager sends the contents of the third column at row $t_{A,B}$ to Alice. In this case, the contents of the third column are $E[k_{A,B}, I_B]$. With knowledge of $k_{A,B}$, Alice and Bob each can decrypt the private ID of the other.

If Bob happens to start the process before Alice does, then the protocol proceeds with the roles of Alice and Bob reversed.

7.7 Checking the Managers

Periodically, users can check if the user-discovery system is operating correctly by interacting with the Managers through the anonymous communication system and verifying their responses. For example, anyone who knows the correspondence (J_A, p_A)—including Alice—could verify if the Public-Key Manager correctly returns Alice's UDM public key p_A when given the request $H(J_A)$ of Alice's public ID. In addition, periodically any user can request the public key of any other user, to create dummy requests that hide the user's potential interest in new recipients. Another way to enhance assurance is to have multiple independent managers.

Recently, Zoom released specifications [29] for their end-to-end encryption services, which include a useful idea for their server to commit cryptographically to the (public ID, public key)-pairs it provides by cryptographically signing each pair. This technique could be easily added to UDM to facilitate checking the correct operation of the UDM Public-Key Manager.

7.8 UDM App

A trusted UDM app [30] running on the user's trusted device supports the UDM Protocol. In particular, the app allows users to register in the UDM service and requests the user's permission for the app to access the contact list locally. This list is not uploaded directly to the User Discovery System. Over the anonymous communication system, the app separately performs individual contact queries and establishes a shared key with each registered user on this contact list.

8 Privacy Notes

We briefly and informally discuss the privacy properties of UDM (see privacy requirements in Section 2), considering information learned by the untrusted user-discovery system and possible attacks on the protocol. Such information includes information stored by the system, information that can be inferred from the history of messages sent to the system, and information observed from communications over the insecure channel. See also open problems in Section 10.

Information Stored by User-Discovery System. Because the Public-Key Manager and Encrypted ID Manager are untrusted, these managers can collude with each other and exfiltrate any information they possess. However, because they are cautious, they must operate correctly, at least for any functionality that can be checked (with non-negligible probability) by users. Because any communication between any user and manager takes place over the anonymous communication system, the manager does not know with whom they are communicating.

The Public-Key Manager maintains a table that associates, for each registered user U of UDM, the hash $H(J_U)$ of their public ID with their UDM public key p_U. The manager also learns the public ID J_U. Thus, the manager learns

who is registered with UDM (in terms of the associated public IDs). All of this information is public.

Whenever the Public-Key Manager is asked for the UDM public key corresponding to the public ID of an unregistered person, the manager returns a randomly generated public key (not belonging to any user). Although we consider membership in UDM public information, returning a random public key makes it harder for users to determine UDM membership, without the help of the untrusted manager. To accomplish this goal, it is important that UDM remember the randomly generated value; otherwise, receiving two different public keys in subsequent queries would reveal non-membership. Nevertheless, an observer could detect a difference in the responses from the Public-Key Manager between those given before and after a user registers. Also, care should be taken not to reveal through differences in the timing of responses whether or not a random key was selected.

Similarly, indexing the public keys by the hash of the corresponding public ID, complicates membership determination for anyone who gains access to the table, without the help of the untrusted manager (see Section 10).

The Encrypted ID Manager maintains a three-column table in which, for each row, the first column holds a truncated hash $t_{A,B} = \hat{H}(k_{A,B})$ of a shared key known by some registered users A and B. The remaining two columns of this row, respectively, hold the encrypted private IDs $E[k_{A,B}, I_A]$ and $E[k_{A,B}, I_B]$ of these two users, ordered by who first contacted the Encrypted ID Manager. Because we assume H and E are perfectly secure, these data are indistinguishable from random bits. UDM maintains the two stated privacy requirements, even if the adversary gains access to the tables.

Information Derived from Communication History. Although the user-discovery system does not know with whom it is communicating, during registration, the Public-Key Manager learns the public ID of each registrant. Moreover, it might be able to infer some information based on the pattern and timing of messages, at least for some situations. The untrusted managers can record their entire communication history and exfiltrate this information. Because all communications take place over the anonymous communication system, external attackers can not observe such patterns.

For example, suppose Alice and Bob are registered with UDM, and Alice adds Bob to her contact list. Suppose also that Bob wishes to communicate with Alice. In relative close proximity in time, Alice and Bob will contact the Public-Key Manager. Pairs of users who match this usage pattern are more likely to be contacts with each other than are arbitrary pairs of users. We assume that the volume of such requests is sufficiently high that no useful inferences can be made. To avoid creating a detectable pattern, when the app initiates requests concerning members of the user's contact list, the app should make those requests in random order and spread out the requests over moderate time periods.

The situation is slightly worse if one or both of the communicants is not yet registered with UDM. In this case, one of both of the parties will first register with UDM, and then both parties will communicate with the Public-Key Manager.

An additional leakage of information can be observed if one of the parties, say Alice, prompts the other, say Bob, to register over the insecure channel. Anyone observing this communication can infer that Alice probably wishes to communicate with Bob. The managers might be able to strengthen such inference if then, in relatively close proximity in time, Bob registers with UDM, and Alice and Bob contact the Public-Key Manager. For this reason, privacy is best maintained if the parties never use the insecure channel.

One possible mitigation for these leaks of information might be for users (via their app) to generate additionalrequests to the user-discovery system to obtain public keys for randomly selected users.

Man-in-the-Middle During Key Establishment. It is important that users check that the Public-Key Manager operates correctly, returning the correct public keys corresponding to the given public IDs. Similarly, any pair of users who were able to verify their shared key and private platform IDs, could additionally check the correct behavior of the Encrypted ID Manager. Without checking the Public-Key Manager, the following *Man-in-the-Middle (MitM)* attack by the user discovery system would be possible.

For any pair of users—say, Alice and Bob—whom the system would like to attack, the system mounts a classic MitM attack. Instead of giving p_B to anyone who requests Bob's public key, the system returns a key p_β of its creation. Similarly, to anyone who requests Alice's public key, the system returns a key p_α of its creation. Because the system does not know with whom it is communicating, it does not know when Alice and Bob are making requests. In the resulting actions, Alice establishes a shared key $k_{A,S}$ with the system (falsely thinking the key is shared with Bob), and Bob establishes a shared key $k_{S,B}$ with the system (falsely thinking the key is shared with Alice). To avoid exposing its malicious activity, the system maintains a six-column table of encrypted IDs, with separate columns for Alice's and Bob's points of view. Variations of this attack are also possible. For example, instead of providing, say, Alice's UDM public key, the system provides the public key of some other user.

A consequence of this attack is that the system learns the shared keys $k_{A,S}$ and $k_{S,B}$, and therefore Alice's and Bob's private IDs. In addition, the malicious system can store incorrect private IDs in the table (e.g., ones that it controls).

In addition to checking the correct operation of the Public-Key Manager, a possible second mitigation of this attack is for Alice and Bob to verify their shared key, as is commonly done at the end of key-establishment protocols. If Alice and Bob met in person, each could scan the other's QR code of their shared key. Such verification cannot be performed securely through the anonymous communication system until Alice and Bob have authenticated each other's private IDs. Furthermore, our assumptions presuppose only one additional channel, which is insecure. If Alice and Bob attempt to verify their shared key over the insecure channel, they risk intrusion of the malicious managers in the middle of that channel, foiling their verification protocol. All communication systems face this challenge of detecting MitM, which stems from the difficulty of authenticating users as they register into the system.

9 Implementation

We implemented a prototype of UDM for the cMix system in the Elixxir platform (see Section 11) supporting over $1,000$ active users. Written in Golang running on a simple Amazon Web Services (AWS) instance with 1 CPU core and 2GB RAM, the prototype has an available bandwidth of 125 Mbits/s. For symmetric encryption we use AES-GCM [31]; for hashing we use Blake2 [32]; and for key establishment we use Diffie-Hellman with a 2048-bit group, as defined in [33]. To truncate any hash value, we take the first 128 bits.

We have not yet implemented a memory-intensive hash function (see Section 10), pending a study of tradeoffs between the running time on the trusted device to process discovery requests and increased work required by an off-line adversary to learn what users are registered.

Our simple and modular implementation runs well on light-weight machines. By contrast, PIR-PSI [20] runs on a single benchmark machine with 2x 18-core Intel Xeon E5-2699 2.30 GHz CPU and 256 GB RAM, and MPCD [24] runs on a server equipped with an Intel Core i7-4600U 2.6 GHz CPU and 16GB of RAM. Each of these approaches simulates a setting where a user with a contact list of length $N_c \in \{1, 2^8, 2^{10}\}$ performs a lookup against a server with a user population of $N_s \in \{2^{20}, 2^{24}, 2^{26}, 2^{28}\}$.

For implementation simplicity, the Public-Key Manager and Encrypted ID Manager run on the same AWS instance and share a PostgreSQL database comprising two tables. Each entry in the encrypted address database includes a Time-To-Live field set to two weeks. This mechanism helps prevent filling the database with older data entries; the anonymous communication system provides the primary defense against flooding attacks. The implementation works as expected and is limited mostly by the speed of the anonymous communication network. The current implementation can handle $N_c = 2^{10}$ with $N_s = 2^{20}$.

Automatically, with support of the UDM app, each user establishes a shared key with every other UDM user she knows (whose public ID appears on her contact list). UDM carries out this key establishment individually, one user pair at a time. Each new user first registers with UDM. The app periodically checks for newly registered members.

10 Discussion

We briefly discuss selected issues, including major design decisions, registration design, authenticating users, encrypted database design, and open problems.

Major Design Decisions. In pursuit of our goal to effect user discovery with minimal disclosure to the system and other users, our major design decisions were: (1) to leverage the capabilities of the underlying anonymous communication system, (2) to separate the public ID from the private ID, and (3) to establish a shared key between each pair of communicants, and to use this key to hide the communicants' relationship with each other from UDM.

The main difficult cryptographic work is for pairs of communicants to establish shared keys. Thus, at some additional complexity, UDM provides greater contact privacy to its users. Our UDM implementation hides much of this additional complexity from the user through an app running on the user's device.

UDM is intended for use with an anonymous communication system. There would be limited value in using UDM on a non-anonymous system, which exposes all traffic flows.

Registration Design. To register with UDM, a user U sends the Public-Key Manager (J_U, p_U), where J_U is U's public ID, and p_U is U's UDM public key. An alternative design choice would have been instead to send $(H(J_U), p_U)$. A disadvantage of this alternative is that anyone could register as someone else. With our choice, the Public-Key Manager can verify that the registrant has control of their J_U, assuming some authentication mechanism were available.

A consequence of our design choice is that the Public-Key Manager learns for each registered user U in UDM their (J_U, p_U) pair. Because the Manager is untrusted, the Manager can exfiltrate the entire list of registered members and their (J_U, p_U) pairs. Although we consider this information public, we prefer not to facilitate its dissemination.

Even without a malicious Manager, for anyone who gains access to the blind directory of UDM public keys, an off-line attack on the public IDs reveals the (J_U, p_U) pairs, assuming the universe of public IDs is known and tractable (e.g., all telephone numbers). Using a memory-intensive hash function [34–37] would mitigate this off-line attack.

Authenticating Users. The difficulties discussed involving registration design and MitM attacks during key establishment stem largely from a fundamental challenge faced by all communication systems: how to authenticate users, including during registration with the system? Some suitable assumptions are necessary to solve this challenge. For a discussion of techniques for doing so with minimal assumptions, see Sherman et al. [38].

Encrypted Address Database Design. Assuming a secure hash function, our design of the encrypted address database (Section 7.3) associates on each line the encrypted private ID with a tag that is the truncated hash of the shared key used to encrypt the private ID. A limitation of this design is that, if the adversary could mount a preimage attack (possibly off-line) of the hash function, then the adversary could decrypt the private ID. Also, it is wise practice not to re-use key material for different algorithms.

A more cautious design can unlink the encryption key k^* used to encrypt the private ID from the tag t as follows: compute $t||k^* = f(k)$, where k is the shared key; f is a cryptographically-secure pseudorandom key-derivation function [39]; and $||$ denotes concatenation. This construction also mitigates the potential danger that the key-exchange process might not generate uniformly distributed key values.

Open Problems. Open problems include:

1. Formally state and prove the privacy properties of UDM.
2. Design a decentralized version of UDM by replacing the centralized user-discovery system with a multi-party computation [40, 41].
3. Expand the capability of UDM to handle groups of users and group communications.
4. Incorporate a Diffie-Hellman key-exchange mechanism that provides *forward secrecy* so that compromise of a user's UDM private key will not compromise the private IDs of other users who previously established a shared key with the user [42, 43].
5. Design a variation of UDM that uses *ephemeral* private IDs that change with every connection. For example, the shared key could act as a seed for cryptographically-secure pseudorandom number generator. This variation would limit loss caused by disclosure of a private ID.

11 Selected Applications of UDM

To illustrate some of the many potential useful capabilities made possible by private user discovery, we describe three selected applications of UDM: thwarting spam, remailers, and in-person private user discovery. We do so in the context of how we integrated our UDM implementation with cMix in the Elixxir platform [44].

The cMix system, as implemented by Elixxir, allows senders to publish a pair of values in an output batch—while hiding which specific one of many senders provided the corresponding pair of values to the input batch. One value of the pair, called the "*payload*," is an encrypted message of a standard size. The other value of the pair is called the "*address*." The basic idea, at a high level, is that the recipient's smartphone looks for its addresses in published output batches and decrypts the paired payloads to recover the messages sent to it. The implementation, for greater efficiency, carries out this high-level idea with multiple gateways from which a client can request messages sent to a given address. Building on cMix, Elixxir provides a variety of services with strong privacy properties.

Thwarting Spam. Private user discovery with a large address space of private IDs can help prevent and deal with spam. Nobody can spam users by simply guessing valid addresses, and there is no public list of valid addresses. Each address in Elixxir is 256 bits long. They are generated by a cryptographic hash function, and therefore appear random and without structure. One of several advantages, of this admittedly huge address space, is that addresses are in some sense *unlisted*. Besides the inconvenience, spam can also in extreme cases undermine privacy of recipients: for example, downloading a huge amount of spam might be detectable.

Remailers. Private user discovery combined with "remailers" can effect powerful functionality to enhance privacy. For example, remailers can forward messages and hide forwarding addresses from untrusted users, and their actions can be updatable and based on conditional logic. Their utility is enhanced by the fact that UDM allows users to change their private IDs and to create a different private ID for use with each contact.

Rather than supplying, say, Charlie's main address (private ID) to UDM, Charlie's smartphone could instead supply the addresses of a remailer. A *remailer* is a small independent service that watches for messages sent to certain addresses that its customer has supplied to it. If it finds a message sent to one of these addresses, in the simplest case, then it just encrypts the payload using the key that customer supplied and

then sends the result over cMix to the forward-to address that the customer also supplied. When the recipient's smartphone finds messages at one of the forward-to addresses it has set up with remailers, it can decrypt the payloads with the keys shared with the remailer before decrypting with the keys from UDM.

Example use cases illustrating the system can be divided roughly into two broad expected types: the first type is where parties know that there is or will be an ongoing trusted relationship. The second type is exemplified by brief chance meetings, where potentially either person may wish to be contactable and/or potentially to be able to contact. For the ongoing trusted type, smartphones may simply exchange their main addresses and keying. Examples of this trusted type include such scenarios as two or more friends meeting in person or enrolling with a medical or professional services office.

The second type of use case is typically where, say, two people, who did not previously know each other, have met unpredictably by chance—potentially, either person may wish to be contactable or be able to make contact later, but with some restrictions enforced at least initially. For instance, one person may: know the whereabouts of an item left accidentally by the other person, remember how or where they had previously seen or known the other person, or a related person, recall the answer to a question or have a helpful suggestion or a question such as about sourcing a fashion item, or have later seen that person from afar with a mutual friend or on some media.

One power of a remailer is that it can decide whether or not to forward, thereby providing a sophisticated filtering service. It can, for instance, be used to set up one-time-use addresses. Doing so lets anyone contact the user once, with a single message, but that is the only message that will be forwarded through that one-time-use address (subsequent messages will be dropped). Further exchange of messages would typically be through the user's main address, which the user's phone could automatically supply if the user replies.

Another power of a remailer is that it can hide the main address from those who may not yet be trusted with it, including for entities who have ongoing communications with the user, reducing the risk of spam. Yet another power is that a remailer can be updatable and conditional. For example, a remailer can allow someone to contact the user only if they present it with a digital signature certifying some condition. In our era of contract-tracing for exposure to the COVID-19 virus, it would be useful to set the condition to be that the sender has been diagnosed with a certain disease (the user could even change the list of diseases later).

Most or all of the functionality of a remailer could be implemented within UDM, especially considering the option of frequently changing private IDs. It is convenient, however, to organize this functionality in remailers external to UDM.

There can be a permissive remailer without filtering whose address is directly linked to a particular legacy address, such as an old email address of the user. For example, the user might allow anyone with the user's old phone number a chance to reach the user, perhaps only once, by virtue of their system enrollment or perhaps just by answering so-called "captchas."

Someone, perhaps if their smartphone were compromised, who is trusted with a user's main address, might unwittingly allow a massive spam attack on that user, perhaps aimed at revealing the user's identity. The target recipient could stop this attack and recover by publicly announcing, through a cMix message, what might be called a "*change of address*." Such an announcement would be authenticated by the preimage of the current address under the one-way function used to form it. The announcement would not include a new address, but rather would alert anyone using the old address to go through user discovery. The user might maintain the previous main address as a legacy address. All traffic could initially be sent through remailers before the user shares a new main address.

A centralized service could hypothetically provide all the user discovery and remailing, but a distributed solution is preferable and has several advantages. UDM's blind directory and encrypted address database could each be provided separately and even by multiple independent entities. One way to structure this architecture, so as not to require multiple inquiries, would be to divide the services by legacy identifier types, or ranges, of fingerprint values. Some redundancy of these services, however, can be advantageous in terms of reliability and verifiability of correctness. Unlike user-discovery services, remailer services can be provided independently by many parties and can be strung together in chains or even graphs, with some information even split between remailers. Some power users might, for example, have permissive remailers for a few legacy addresses and different remailer chains for public keys for which they desire restricted access.

In-Person Private User Discovery. We describe a variation of UDM that can be used when two people, possibly strangers, meet in close physical proximity, each carrying a trusted device. In this scenario, public IDs are not needed because the physical proximity plays the role of public IDs. Each trusted device broadcasts a randomly chosen ephemeral public key, used only for this one encounter. The broadcast might happen over Bluetooth. The two people then establish a shared key using a Diffie-Hellman key exchange.

To protect against MitM, taking advantage of their physical proximity, the parties then verify their key using one of the following optical or aural options. (1) Each party's device scans a QR code of the other's shared key and compares it with their key. (2) Using a truncated hash of the key, each party selects a piece of music from a standard list of recognizable

tunes. The trusted devices then play the music simultaneously, strictly alternating which device plays each beat. The device with the larger public key starts. Each person verifies that the music sounds correct. If the music sounds correct, the parties have high assurance that their keys match.

At this point, there are two variations. (1) Encrypting their messages with the shared key, the parties can directly exchange IDs. (2) The parties can engage in the UDM protocol, starting at the step after establishing a shared key. In either variation, the ID could be the ID of a remailer. By using remailers, the parties could limit the privacy risk created by this encounter, for example, by allowing for only one subsequent communication and only under certain conditions.

12 Conclusion

We propose a new user-discovery protocol, UMD, for anonymous communication systems that minimally discloses information to the system, external attackers, and other users. With Bob's permission, any user Alice can discover Bob's private platform identifier without disclosing it to the system, even under an off-line dictionary attack. Furthermore, UDM does not reveal to Alice any other private platform identifier, or any links between two other users, and UDM enjoys low computation and communication complexity. Existing user-discover systems, such as those for Telegram, Signal, and Skype do not achieve these objectives.

An increasing number of communication systems are providing confidentiality, integrity, and authentication services. A few systems also protect meta-data of traffic flows. For any communication system to be useful, it must provide a mechanism through which users can discover the platform identifiers of intended recipients. We should expect more and demand that any communication and user-discovery system work with minimal information disclosure, protecting not only message contents and traffic flows, but also the contact lists and social graphs of its users.

We hope that our protocol will provide a useful tool, and set a new higher level of privacy and performance expectations, for all communication systems.

Acknowledgments

We thank Akshita Gorti for helpful comments and drawing Figures 1 and 3. Yaksetig was supported in part by the MOBILE+ project, coordinated by the University of Porto, with support from the European Commission through a fellowship, during which he visited UMBC for the spring 2017 semester. Sherman was supported in part by the National Science Foundation under SFS grants DGE-1241576, 1753681, and 1819521, and by the U.S. Department of Defense under CySP grants H98230-17-1-0387, H98230-18-1-0321, and H98230-19-1-0308.

References

[1] Signal. https://signal.org, 2013.

[2] WhatsApp. https://whatsapp.com, 2009.

[3] David Chaum, Debajyoti Das, Farid Javani, Aniket Kate, Anna Krasnova, Joeri de Ruiter, and Alan T. Sherman. cMix: Mixing with minimal real-time asymmetric cryptographic operations. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *Applied Cryptography and Network Security - 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings*, volume 10355 of *Lecture Notes in Computer Science*, pages 557–578. Springer, 2017.

[4] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, pages 21–21, Berkeley, CA, USA, 2004. USENIX Association.

[5] Telegram. https://telegram.org, 2013.

[6] Moxie Marlinspike. The Difficulty of Private Contact Discovery. https://whispersystems.org/blog/contact-discovery, 2014.

[7] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[8] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[9] Moxie Marlinspike. Technology preview: Private contact discovery for Signal. https://signal.org/blog/private-contact-discovery, 2017.

[10] Phil Zimmermann. A privacy-preserving contact discovery server. https://github.com/SilentCircle/contact-discovery, 2015.

[11] Mario Yaksetig. Implementation and analysis of privategrity user discovery: Learning contact identifiers with minimal information disclosure. Master's thesis, Faculdade de Engenharia, Universidade do Porto, Porto, Portugal, July 2017.

[12] WeChat - Free Messaging and calling app. https://wechat.com, 2011.

[13] Skype | Communication tool for free calls and chat. https://skype.com, 2009.

[14] Zoom. https://zoom.us, 2011.

[15] Keybase. https://keybase.io, 2014.

[16] David Chaum. Blind signatures for untraceable payments. In David Chaum, Ronald L. Rivest, and Alan T. Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US.

[17] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown. *CoRR*, abs/1801.01207, 2018.

[18] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. *CoRR*, abs/1801.01203, 2018.

[19] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, page 991–1008, USA, 2018. USENIX Association.

[20] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. PIR-PSI: Scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies*, 2018(4):159 – 178, 2018.

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

[22] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 337–367, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[23] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing: Improvements and extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 1292–1303, New York, NY, USA, 2016. Association for Computing Machinery.

[24] Danie Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *Proceedings of the 28th USENIX Conference on Security Symposium*, SEC'19, page 1447–1464, USA, 2019. USENIX Association.

[25] Michael O. Rabin. How to exchange secrets with oblivious transfer. *Technical Report TR-81*, 1981, 1981.

[26] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2): 231–262, March 2004.

[27] D. Dolev and A. C. Yao. On the Security of Public Key Protocols. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, SFCS '81, pages 350–357, Washington, DC, USA, 1981. IEEE Computer Society.

[28] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6): 644–654, 1976.

[29] Josh Blum, Simon Booth, Oded Gal, Maxwell Krohn, Karan Lyons, Antonio Marcedone, Mike Maxim, Merry Ember Mou, Jack O'Connor, Miles Steele, Matthew Green, Lea Kissner, and Alex Stamos. E2E Encryption for Zoom Meetings. https://github.com/zoom/zoom-e2e-whitepaper/blob/master/zoom_e2e.pdf, 2020.

[30] xx network. xx messenger app. https://elixxir.io/xx-messenger, 2019.

[31] David Mcgrew and John Viega. The Galois/counter mode of operation (GCM). 02 2004.

[32] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. BLAKE2: simpler, smaller, fast as MD5. https://blake2.net/blake2.pdf, 2013.

[33] T. Kivinen and M. Kojo. More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). https://tools.ietf.org/html/rfc3526, 2003.

[34] A. Biryukov, D. Dinu, and D. Khovratovich. Argon2: New generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 292–302, 2016.

[35] Niels Provos and David Mazières. A future-adaptable password scheme. In *USENIX Annual Technical Conference, FREENIX Track*, 1999. URL http://www.usenix.org/events/usenix99/provos.html.

[36] C. Percival and S. Josefsson. The scrypt Password-Based Key Derivation Function. https://tools.ietf.org/html/rfc7914, 2016.

[37] Burt Kaliski. PKCS #5: Password-Based Cryptography Specification Version 2.0. https://tools.ietf.org/html/rfc2898#section-5.2, 2000.

[38] Alan T. Sherman, John Seymour, Akshayraj Kore, and William Newton. Chaum's protocol for detecting man-in-the-middle: Explanation, demonstration, and timing studies for a text-messaging scenario. *Cryptologia*, 41(1):29–54, February 2017.

[39] H. Krawczyk and P. Eronen. MAC-based extract-and-expand key derivation function (HKDF). Internet Engineering Task Force (IETF), Request for Comments: 5869, May 2010. https://tools.ietf.org/html/rfc5869.

[40] David Chaum, Ivan B. Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, pages 87–119, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg. ISBN 978-3-540-48184-3.

[41] David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In *Proceedings on Advances in Cryptology*, CRYPTO '89, page 591–602, Berlin, Heidelberg, 1989. Springer-Verlag.

[42] Christoph G. Günther. An identity-based key-exchange protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 29–37, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg.

[43] Trevor Perrin and Moxie Marlinspike. The Double Ratchet Algorithm. https://signal.org/docs/specifications/doubleratchet/doubleratchet.pdf, 2016.

[44] David Chaum. Elixxir. https://elixxir.io, 2018.

Submitted to PETS 2021 (May 29, 2020).