Where's Wally? How to Privately Discover your Friends on the Internet

Panagiotis Papadopoulos
FORTH-ICS, Greece
panpap@ics.forth.gr

Antonios A. Chariton
University of Crete, Greece
csd3235@csd.uoc.gr

Elias Athanasopoulos
University of Cyprus, Cyprus
athanasopoulos.elias@cs.ucy.ac.cy

Evangelos P. Markatos
FORTH-ICS, Greece
markatos@ics.forth.gr

ABSTRACT

Internet friends who would like to connect with each other (e.g., VoIP, chat) use point-to-point communication applications such as Skype or WhatsApp. Apart from providing the necessary communication channel, these applications also facilitate contact discovery, where users upload their address-book and learn the network address of their friends. Although handy, this discovery process comes with a significant privacy cost: users are forced to reveal to the service provider *every person they are socially connected with*, even if they do not ever communicate with them through the app.

In this paper, we show that it is possible to implement a scalable User Discovery service, without requiring *any* centralized entity that users have to blindly trust. Specifically, we distribute the maintenance of the users' contact information, and allow their friends to query for it, just as they normally query the network for machine services. We implement our approach in *PROUD*: a distributed privacy-preserving User Discovery service, which capitalizes on DNS. The prevalence of DNS makes *PROUD* immediately applicable, able to scale to millions of users. Preliminary evaluation shows that *PROUD* provides competitive performance for all practical purposes, imposing an overhead of less than 0.3 sec per operation.

CCS CONCEPTS

• **Security and privacy → Pseudonymity, anonymity and untraceability**; **Privacy-preserving protocols**; *Privacy protections*;

KEYWORDS

Mobile User Discovery, Privacy of Social Graph, Adress-book, DNS

ACM Reference Format:
Panagiotis Papadopoulos, Antonios A. Chariton, Elias Athanasopoulos, and Evangelos P. Markatos. 2018. Where's Wally? How to Privately Discover your Friends on the Internet. In *ASIA CCS '18: 2018 ACM Asia Conference on Computer and Communications Security, June 4–8, 2018, Incheon, Republic of Korea.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3196494.3196496

ASIA CCS '18, June 4–8, 2018, Incheon, Republic of Korea

ACM ISBN 978-1-4503-5576-6/18/06…$15.00
https://doi.org/10.1145/3196494.3196496

1 INTRODUCTION

More and more users turn to Internet-based, real-time applications (e.g., Messenger, WhatsApp, Viber, Skype, WeChat, Snapchat) for their daily communication needs (VoIP, chat, file/image/video sharing). Recent studies [33] predict that instant messaging (IM) alone will account for 75% of the overall mobile communication traffic by 2018. Other studies [3] show that the combined user base of the top four mobile communication apps is larger than the combined user base of the top four social networks, proving that these apps have managed to transcend traditional social networks.

Today, each time a user opens her mobile app, the application conducts an extensive *User Discovery* across all friends in her address-book. Instantly, the user exposes her entire set of contacts to the application provider, relinquishing the privacy of her social graph. Hence, the provider learns *which* user is socially connected with *whom*, even if they haven't ever sent a single byte to each other. What is more, there are cases where the revealed social graph does not even belong to the mobile application. For example, in applications like Whatsapp (or Viber), the address-book uploaded during User Discovery is not the one that contains the Whatsapp friends of the users; it is her mobile phone address-book, which contains people that she is socially connected with even if they have never installed Whatsapp!

Building services which can provide in practice scalable User Discovery, while respecting the user's privacy, is challenging. Existing services are currently based on a centralized architecture for the User Discovery process: a directory server (or group of servers) operated by the application provider is responsible to store, maintain, and respond to queries regarding its users' network addresses. The centralization of this information to a single provider, inevitably, transforms it into a powerful "Big Brother", who maintains, and owns data, that include both user network addresses, and parts of the global social graph. To make matters worse, there have been incidents where this sensitive data were used beyond the control of the user (i.e., sold to advertisers [25, 27, 29, 31, 34], handed over to government agencies[39], or included, as an asset, in the company's future buy-out [23, 32]). Even in the case of trusted services, there exist bold examples, where the application's bad design allowed sensitive data to be leaked [15], enabling anyone interested to find out, at any time, the current IP address, and consequently the approximate geolocation of any user.

To address these drawbacks, the research community explored *if it is possible to build a User Discovery service, without revealing the entire or part of the user's social graph.* Vuvuzela [36], for instance,

is a full-featured system for private messaging, resisting traffic analysis attacks. However, it requires users to have their devices *constantly online*, for sending no-op messages, even during idle times. Vuvuzela is an all-or-nothing approach: you cannot use it just for discovering your buddies. In this paper, we are primarily interested in privacy-preserving User Discovery, which should be neutral to the actual application. Apres [14], on the other hand, attempts to solve the problem over Tor, which can cause additional overhead, especially for mobile clients, and IoT. Other systems are based on Private Information Retrieval (PIR) [6], requiring a large amount of traffic (linear in some cases to the size of the database), or a large amount of computation, or a combination of both, which may limit the scalability of the system [2, 4].

In this work, we propose *PROUD* (PRivacy-preservation Of User Discovery): a scalable system, which (i) enables users to control their current network address without relying on any centralized infrastructure, and additionally, (ii) allows them to find the network addresses of their friends, without revealing their social associations. *PROUD* is based on a core Internet service, available for everyone: the Domain Name System (DNS). In *PROUD*, users who would like to be discovered by their friends, place their network address in (specially crafted and encrypted dead-drops) in the DNS. Users, who would like to find the network address of their friends, retrieve the dead drops from DNS, and, after appropriate decryption, they (and only they) are able to find the current network address of their friends. To summarize, we make the following contributions:

- We design *PROUD*: a system to enable users to discover their contacts in a privacy-preserving fashion, without revealing *who is socially connected with whom*. Following similar distributed approaches [1, 12], *PROUD* does not require the user to blindly trust *any* centralized entity.
- We implement *PROUD* as a standalone service by leveraging the publicly accessible datastore of DNS, and we provide[1] an API library to support any type of app. Our service is immediately applicable, without needing maintenance by *any* single entity.
- To quantify the effectiveness of our approach, we develop a simple IM application, which *outsources* its user discovery to *PROUD*. The performance evaluation of our system shows that it has minimal bandwidth requirements, and adds negligible latency to the user experience (0.35 sec on average).

2 THREAT MODEL

Apart from sensitive data that a User Discovery service delivers (i.e., users' current network address), there are also metadata produced by its operations (i.e., Set/Get address), which reveal who queries for whom. With such metadata, an application provider can reconstruct the users' social graph (i.e., who is socially associated with whom). This graph can be analyzed, together with other leaked information [24, 26, 28], to infer interests or preferences (political or religious beliefs, sexual preferences, etc.) [10], and be sold to advertisers, even when they have not shared *any* related information [21].

In this work, we assume an Internet service, which operates on a community-based interaction. In such services, users are clustered in communities. A typical example is a chat application, where users form communities that exchange text messages. We additionally assume an attacker who aims at discovering the community structure (i.e., the social topology) by attacking the service. The attacker can learn the community structure either by being in cahoots with the centralized service, or by compromising the service and leaking their data. PROUD addresses that by distributing the community information in two sets of non-colluding nodes: (i) the ones responsible for setting the user discovery information (registration servers), and (ii) the ones responsible for responding to queries for this information (resolvers). Of course, the non-colluding nodes assumption is not new in information-theoretic PIR (IT-PIR) [6], and has been widely used in the area [4]. PROUD can preserve the privacy of users as long as *only* one of the two above types of nodes is compromised by the attacker: either registration servers or resolvers. Finally, we must note that the way a user will connect with their friends after querying PROUD (e.g., through TLS connection, Tor or physically by car), is beyond the scope of our paper.

3 SYSTEM OVERVIEW

3.1 Our approach: *PROUD*

The objective of *PROUD* is to enable users to advertise their current network address to their friends only, without revealing their social topology to any third party. In a nutshell, we design a system where a user (which we call by convention Alice), creates a dead-drop (namely friendship record) with her current contact point (e.g., IP address), which her friend (Bob) can query.

Our system needs two separate types of servers: (a) a registration server, responsible for storing and maintaining the friendship records of Alice, and (b) a recursive resolver, that handles the record querying part. In order for our system to be immediately applicable, we have to capitalize our system on an existing datastore. This datastore must fulfill specific requirements that are necessary for *PROUD*. These requirements include: (i) distribution of data and infrastructure (i.e., servers), (ii) *distribution of management* by allowing anyone interested to maintain their own servers, thus participating in a wider coalition of nodes, and finally, (iii) provide a high level of scalability.

Such a system could be a structured peer-to-peer system based on DHT, e.g., Chord [35] or Kademlia [20]. The example of GNUnet's Name System (GNS) [37] proves that DHT-based systems can successfully resolve keys to values. Similar translation of keys to values, is already provided by the traditional DNS for years, making this system an essential part of the Internet.

The key-value datastore of DNS. Although DNS was first designed to map domain names to IP addresses, its success as a highly scalable, lightweight, key-to-value mapping system, made it indispensable to many Internet-related applications, and there are several projects leveraging it to distribute their mappings (e.g., SSL certificates [5, 9], server discovery [13], blacklist querying [17]). The split design of DNS allows PROUD to perform over 2 separate types of nodes, since each node may take *one of the two roles* below:

(1) Authoritative nameserver: a server responsible for a DNS zone that provides responses to queries about records in this zone. In PROUD, such nameservers are responsible of user registration, and maintaining the friendship records.

[1] Source Code: https://github.com/panpap/PROUD

(2) Recursive resolver: receives queries from the users' devices, contacts one or more authoritative nameservers inside the zone hierarchy to locate a DNS record, before finally responding back to the querying user. In PROUD, Bob can query *any* existing DNS resolver to retrieve friendship records.

3.2 The PROUD protocol

For presentation purposes, throughout this section, we use a mobile instant messenger (MIM) as our application example, which supports two types of directional friendships: followers and followees[2]. In our scenario, both Alice and Bob have a user unique identifier (UID) such as phone number, username, email. For simplicity's sake, they will be the only participants in our example. Bob is wishing to follow his friend Alice, so he only queries the system for the address of Alice[3]. Our protocol, supports semantics that can be simplified to `put(key, value)` and `get(key)` commands and provides 3 main operations: (i) create a new friendship, (ii) update with the device's current IP address, and (iii) discover the address of a friend in the network. Beginning with a straw-man design, below, we describe the basic functionalities of PROUD.

To bootstrap PROUD, we assume that (i) each user is able to create and store locally her personal asymmetric key-pair and that (ii) there is an out-of-band channel, through which (similar to other systems in the literature [2, 4, 14, 36]) users can initially exchange information and public keys with their friends. The above asymmetric key-pair is stored encrypted on the user's device using symmetric cryptography and a password as key that the user can memorize.

A. User registration. Every user of our system is able to follow other users without any further requirements. In order to have followers of her own, a user, through PROUD, acquires a DNS zone in a domain[4,5]. This zone is maintained by the authoritative nameserver, which is responsible for adding, updating, and publishing information about this particular zone. Every zone includes multiple records. Each record has an *index token* (a subdomain) and a payload. In PROUD, every pair of friends (i.e., followee and follower) has its own record (namely *friendship record*), which includes as payload the current address of the followee.

B. New friendship. In every real-time MIM application (e.g., Viber, WhatsApp), whenever Bob wants to follow Alice, he just chooses her from his contact-list, and the application sends her a follow request. In PROUD, as soon as Alice accepts the follow request, she creates a new friendship record in her authoritative server (step 1 in Figure 1). This new friendship record represents their directional friendship and contains the concatenated UIDs of both Alice and Bob as an index token they both know, along with her current network address. Hence, assuming that Alice's DNS zone is in the example.com domain and her current IP address is IP_A =>`10.1.0.52`, the mapping of the created friendship record will be the following:

$$\underline{key} : \langle UID_B | UID_A \rangle.example.com$$
$$\underline{value} : 10.1.0.52$$

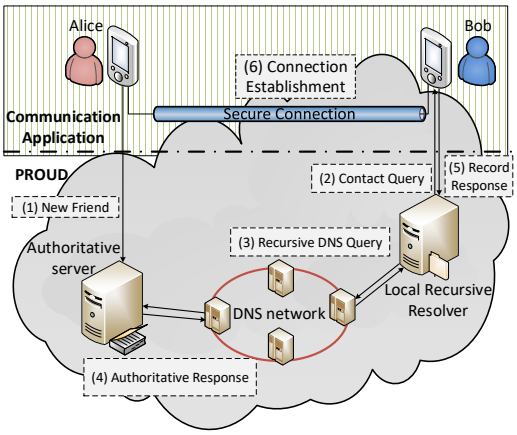

Figure 1: High level overview of *PROUD*. Alice place her friendship records to an authoritative server, while Bob assigns his local resolver to retrieve her friendship record from the DNS network.

As we can see, the friendship record, in this first straw-man proposal, consists of two fields: (i) the index token, which contains the concatenated UIDs of both follower and followee, and (ii) the current IP address of the followee.

Of course Alice needs to ensure Bob that this particular friendship record is created by her, therefore before uploading her new friendship record to the authoritative server, she signs her friendship record's payload using her private key. In this way, she prevents adversaries from impersonating her by creating rogue friendship records, tricking Bob to miss-follow a malicious user instead of her.

Apart from providing authenticity to Bob, Alice also needs to ensure that the information destined for Bob will be accessible only by him. Thus, whenever she creates or updates a friendship record, apart from signing the payload, she also encrypts it with a fresh symmetric key that only Bob can access. This way, she can guarantee the confidentiality of the query response, allowing only Bob to decrypt the friendship record's payload, and learn her current network address. As a consequence, up to this point, the friendship record of Alice and Bob is the following:

$$\underline{key} : \langle UID_B | UID_A \rangle.example.com$$
$$\underline{value} : E_{Pub_B}(K), E_K(IP_A, T), S_{Priv_A}(H(E_K(IP_A, T)))$$

where K is a fresh symmetric cryptographic key[6], the functions E and S are encryption and signing respectively, while H is the SHA secure hash algorithm to ensure the integrity of the cyphertext. As we see, the payload now is three-fold, since it includes: (i) the IP address of Alice (IP_A) together with a timestamp T encrypted with the *unique to this record version* symmetric key K, (ii) the key K, encrypted with Bob's public key Pub_B, and (iii) the SHA digest of the ciphertext signed by Alice, so an adversary will not be able to tamper, replay or alter the encrypted IP address. Note at this point, that symmetric keys K are used only once. This way, they are not needed to be stored in the user device. Symmetric encryption, at this point, allows us to easily encrypt messages of arbitrary size.

Using UIDs in plaintext as an index token, obviously causes a serious privacy leak. To avoid that, Alice could hash the concatenation of both UIDs: $\langle h(UID_B|UID_A) \rangle$. Since Bob and Alice know the UID

[2]Bidirectional friendships are implemented in a similar way.

[3]Of course, in practice, all parties participate in both registration and query functions, while at the same time they have multiple followees and followers.

[4]The user does not need to register a new domain: we utilize free services (e.g., https://pointhq.com/), which provide DNS hosting and DNS zones.

[5]User's domain is transmitted together with the public key during bootstrapping.

[6]Depending on the cryptographic algorithm, K can be the key and the Initialization Vector (IV), or just the key.

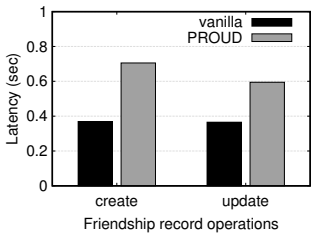

Figure 2: Execution time per set-friendship operation. There is an additional delay of less than 0.35 sec on average per operation due to cryptographic computations.

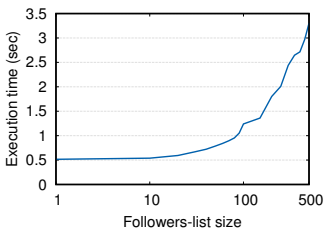

Figure 3: Followers-list update time as a function of its size. It takes less than 1.8 sec on average to update as many as 200 friendship records with the new IP.

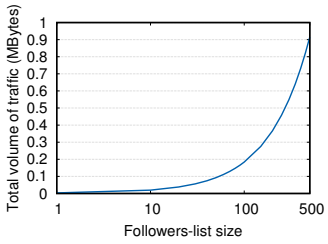

Figure 4: Total traffic volume during followers-list update. Even for users with large list of followers, updating all friendship records takes less than 1 MByte.

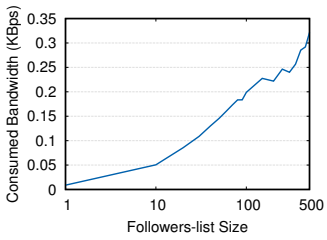

Figure 5: Total generated bandwidth during followers-list update. It takes less than 0.22 KBps on average to simultaneously update 200 friendship records.

of each other, they can easily produce this index token and query a DNS resolver for the record representing their particular friendship. But again, using an unsalted hash means that anyone able to enumerate both UIDs may reconstruct this index token, and hence query for it. Obviously even by performing such query they will not manage to learn Alice's network address, since they could not decrypt the response. Yet, they will be able to detect the existence of an association between Alice and Bob. To remedy this, and provide Perfect Forward Secrecy (PFS), in PROUD we periodically generate a new random index token through a pseudorandom number generator (PRNG). To achieve this, upon friendship bootstrapping, Alice and Bob feed their pseudorandom number generator with the same seed SE_{AB}[7] and change this index token periodically (e.g., once per day). This way, it is guaranteed that (i) Alice and Bob are the only parties able to *reproduce* and *query* this index token, and (ii) that the record queries that Bob performed in the past cannot be linked with the ones he performs now. The final format of the friendship record that Alice creates in PROUD is the following:

$\underline{type}$: TXT
$\underline{key}$: $R(SE_{AB}).example.com$
$\underline{value}$: $E_{Pub_B}(K)$, $E_K(IP_A, T)$, $S_{Priv_A}(H(E_K(IP_A, T)))$

where R is the output of the PRNG function that uses SE_{AB} as input. As one can observe, we use the TXT type of DNS records, which allows us to add arbitrary data in our friendship records.

C. Friendship update. When Alice connects to the network, she may roam across Access Points, from WiFi to a Mobile Network. This makes her device pass to new network state, which from now on we will call a new *epoch*. Each *epoch* of a user includes her move to a new network address. At the beginning of each new *epoch*, the user has to publish her new IP address immediately, in order for her followers to remain updated. Hence, the device must monitor for IP address changes and then promptly update all friendship records in PROUD. To perform such network state update, Alice pushes the updated friendship records to her authoritative nameserver. All new recursive queries after that will fetch the updated friendship records. Note that as described above, this record update may not take place only every epoch but also periodically, as frequently as the participants have decided to change the index token of their friendship record.

[7]This shared seed is transmitted as a tuple during friendship bootstrapping together with public keys and the followee's domain.

D. Friendship revocation. There may be cases, where Alice needs to remove Bob from her followers. In such case, Alice just removes from the authoritative nameserver the friendship record that represents her friendship with Bob. In case she remains in the same *epoch*, she also has to deny any future connections from Bob, until she moves to a new *epoch*. After this point, Bob cannot obtain the new IP address and thus he cannot relocate her in the network.

E. User discovery. In order for Bob to remain connected with Alice, he needs to be up-to-date with her latest *epoch* before attempting to establish any connection. Therefore, he needs to periodically query PROUD for Alice's current network address. Hence, every N seconds, Bob makes a simple query to his local resolver (steps 2-5 in Figure 1), asking for the record with the index token that represents his friendship with Alice. As soon as he retrieves the response, he uses (i) his private key to recover the key and decrypt the payload, and (ii) Alice's public key to verify that this payload was indeed created by her. In case the timestamp is obsolete he immediately discards the record, otherwise he can recover the IP address and open a communication channel with Alice.

4 SYSTEM EVALUATION

We implemented PROUD as a library to facilitate easy integration in any application. The library is written in Java, the cryptographic module uses the Bouncy Castle's Crypto APIs [16]. For authentication, we use RSA asymmetric cryptography with PKCS1Padding and 1024-bit keys. For encryption, we use AES symmetric cryptography with PKCS5Padding and 256-bit keys. AES in CBC mode is also used for encrypting the local stored lists and keypairs. Each IV is also encrypted with AES-ECB mode in one block. Additionally, we use SHA256 for integrity verification and SHA2-PRNG for our cryptographically secure pseudorandom number generation (CSPRNG). For the communication with the authoritative server, we used the `dnsjava` library [38]. For our experiments, we implemented a simple IM application which outsources its User Discovery functionality to PROUD. For our measurements, we used a PC equipped with an Intel Core i7-4790 Processor, 3.60GHz with 8MB L2 cache, 8GB RAM, and an Intel 82567 1GbE network interface.

Experimental Evaluation: First, we study the execution time of set-friendship operations. Specifically, we measure the execution time of friendship record create and update requests in PROUD. In addition, we build a simple DDNS client (vanilla case) in Java to compare our results with the vanilla DNS record operations.

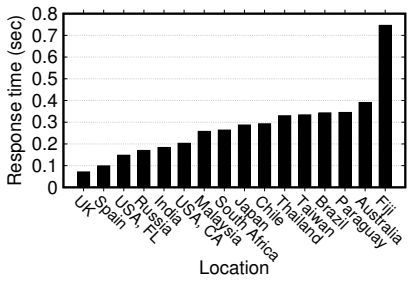

Figure 6: Querying latency from different geolocations when the authoritative server is located in Europe. In most of the cases the response time is less than 1 sec.

We repeat each operation 100 times and in Figure 2 we report the average values. As we can see, the average latency of our service reaches 0.7 sec for a *new friendship operation* and 0.59 sec for an *update friendship* operation. On the other hand, in the vanilla case, the same record operations cost 0.37 sec and 0.36 sec respectively. This practically insignificant 0.3 sec latency imposed in each operation of our system is caused by the cryptographic computations. A user, to allow their friends to continue following them uninterruptedly when they move inside a network, has to update all of their friendship records as soon as they change network address. In our prototype, we use threads to perform such multiple update operations in parallel and achieve the lowest possible latency. In Figure 3, we present the time it takes for a user to update all of their friendship records as a function of the number of followers in the user's followers-list. As we see, even for a large number of followers (i.e., 500 followers), the update time for the entire list is reasonably low, less than 3.5 sec on average.

Next, we quantify the amount of traffic the most heavy-load operation of PROUD: the update operation, generates as a factor of the number of followers. To do so, we trigger a new epoch on the user's device and we measure the bytes uploaded to the authoritative server. As we can see in Figure 4, there is a linear increase generating a traffic of up to 0.36 MBytes when a list of 200 followers gets updated. This volume overhead is reasonably low even for cellular networks. In addtion, Figure 5 shows the average bandwidth consumption as a function of the followers-list size. We notice that even in the case of 500 followers the bandwidth consumption needed for updating all friendship records is less than 0.32 KBps on average. Evidently, the total bandwidth consumed during the update operation is not really an issue, even for users with lots of followers using cellular network connections.

In Figure 6, we measure the response time of queries in PROUD from different geographic locations, when the authoritative server is located in Europe. We notice that even for the most remote location, the Fiji islands, the response time was as low as 0.7 sec. According to Ratatype [30], users can produce on average 41.4 words (207 keystrokes) per minute when chatting, and according to [18, 40] an IM transmission contains approximately 29 characters on average. This means that a user needs more than 2 sec to type even a short message of 10 characters. This window is more than enough for PROUD to retrieve the friend's new network address.

TTL reduces the load of servers but in PROUD, it may also create service hiccups: very low TTL values can cause relatively high load on the authoritative server, but very high TTL values may let the resolver respond with an obsolete friendship record of Alice causing a service hiccup to Bob's application. In PROUD, we follow an adaptive approach tuning the TTL according to Alice's moving frequency patterns. Particularly, we vary the TTL of her records based on the frequency of new epoch production. Hence, the more frequently Alice moves to new epochs the lower the TTL will be assigned in her next record updates. In a similar way, the TTL of a sporadically moving Alice are increased reducing the number of queries at the authoritative server, letting the resolvers respond to followers' periodic queries from their cache.

5 RELATED WORK

Ben Laurie in [14] proposes Apres: a system to allow Alice know when Bob is online along with his network address without letting anyone else know about it. Unlike PROUD, Apres assumes a dedicated server working as a hub to store and forward messages and requires the use of Tor in order to access the server anonymously. DP5 [4] aims to provide privacy of presence to real-time communications. Unlike PROUD, it requires special infrastructure consisting of a registration server and a set of PIR servers. DP5 imposes high network overhead: users, during their communication with PIR servers, need to download or upload their full friends-list for each operation. Moreover, DP5 is less scalable than PROUD: it requires an increasing management complexity for the additional PIR servers, considering the ever-increasing number of users.

Vuvuzela [36] is a system for private communication under heavy surveillance. It uses a dedicated infrastructure containing a Tor-like chain of servers which utilize onions, along with cover traffic to conceal the users message exchanges. Although Vuvuzela is able to adequately preserve the privacy of the user's social graph, its a heavyweight approach that requires the users to have their devices constantly online to send no-op messages even during idle times of the user. This results to a large bandwidth requirement per user (around 12 KB/sec per round or 12 GB/sec in total).

Pung [2] provides private communication based on a computational PIR model. Similar to Vuvuzela, Pung, assumes an untrusted ISP, however it further assumes all of its servers as untrusted. To improve its performance, Pung uses a probabilistic multi-retrieval scheme, which allows its server to efficiently process multiple retrievals from the same user. Although this scheme allows Pung to reduce computational costs by up to $11x$, its applicability is still questionable since the network costs are significantly increased.

Signal [19] is a commercial approach, which protects the privacy of the user's address-book during contact discovery by leveraging architectural support, and specifically Intel's Software Guard Extensions (SGX). Technically, the user performs *remote attestation* to query a directory service which runs in a trusted enclave at the remote server. The mayor drawback of the proposed approach is the current limited support of SGX enclaves in mobile devices.

6 DISCUSSION AND CONCLUSION

Colluding nodes. There is a case, where the infrastructure PROUD leverages may be hostile with an adversary compromising *both* parts of our system (authoritative servers and DNS resolvers). Then, these nodes may collude and link (a) record updates of authoritative

server and (b) queries received by the resolver, thus inferring *who queries whose record*. An adversary able to compromise such large parts of the global DNS network is considered very powerful (one may also say unrealistic). However, even in this case, the powerful adversary will be able to correlate only individual user associations and not the entire social graph of the querying user. In particular, he will be able to reveal only the user's associations with friends using the specific compromised authoritative server.

Users behind NAT and IPv6. Most distributed applications nowadays suffer when end-nodes are not in the same address realm. This situation affects all users behind NAT. However, in the near future, with the rise of IPv6, every device will have its own network address without needing any such layer. Current metrics put IPv6 at a minimum of 20% global adoption [8], and local uses (within a country) to over 50%. Additionally, IPv6 provides Privacy Extensions [22], which enables devices to frequently change network addresses within a day to preserve their privacy. It is apparent, that with such a functionality in place, the necessity for efficient privacy-preserving User Discovery becomes crucial. Of course, currently there are several practices for NAT penetration including the use of a trusted rendezvous point [11] or hole punching [7]. Given the short remaining life of NAT, this issue is beyond the User Discovery procedure and thus beyond the scope of this paper.

Conclusion: This paper proposes *PROUD*: a scalable privacy-preserving user discovery service able to protect both (i) data (user's current IP address) and (ii) metadata (who queries for whom). In *PROUD*, we decouple the user discovery from the application and we build a standalone service, which leverages the existing network DNS. This way, our approach is scalable and immediately applicable. Furthermore, in *PROUD*, we allow the users delegate trust among each other without relying on any centralized infrastructure that has to be blindly trusted. We implemented a prototype by building an IM on top of *PROUD*. Results of our evaluation show that even for very large contact lists, it has minimal bandwidth requirements and imposes practically negligible latency to the user experience.

Acknowledgments: The research leading to these results has received funding from European Union's Marie Sklodowska-Curie grant agreement No 690972. The paper reflects only the authors' view and the Agency and the Commission are not responsible for any use that may be made of the information it contains.

REFERENCES

[1] M. Ali, R. Shea, J. Nelson, and M. J. Freedman. Blockstack: A new internet for decentralized applications. Technical Report, 2017.

[2] S. Angel and S. Setty. Unobservable communication over fully untrusted infrastructure. In *OSDI'16*.

[3] BI Intelligence. Messaging apps are now bigger than social networks. http://www.businessinsider.com/the-messaging-app-report-2015-11, 2016.

[4] N. Borisov, G. Danezis, and I. Goldberg. Dp5: A private presence service. In *PET'15*.

[5] A. A. Chariton, E. Degkleri, P. Papadopoulos, P. Ilia, and E. P. Markatos. Dcsp: Performant certificate revocation a dns-based approach. In *EuroSec '16*.

[6] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *FOCS'95*, 1995.

[7] B. Ford, P. Srisuresh, and D. Kegel. Peer-to-peer communication across network address translators. In *ATC '05*.

[8] Google. Google ipv6 statistics. https://www.google.com/intl/en/ipv6/statistics.html.

[9] P. Hoffman and J. Schlyter. The DNS-based authentication of named entities (DANE) transport layer security (TLS) protocol: TLSA. https://tools.ietf.org/html/rfc6698, 2012.

[10] C. Jernigan and B. F. Mistree. Gaydar: Facebook friendships expose sexual orientation. *First Monday*, 14(10), 2009.

[11] T. Kato, N. Ishikawa, H. Sumino, J. Hjelm, Y. Yu, and S. Murakami. A platform and applications for mobile peer-to-peer communications. In *MobEA '03*, 2003.

[12] J. Kopstein. The mission to decentralize the internet. http://www.newyorker.com/tech/elements/the-mission-to-decentralize-the-internet, 2013.

[13] M. Krochmal and S. Cheshire. Dns-based service discovery. 2013.

[14] B. Laurie. Apres-a system for anonymous presence. Technical Report.

[15] S. Le Blond, C. Zhang, A. Legout, K. Ross, and W. Dabbous. I know where you are and what you are sharing: Exploiting p2p communications to invade users' privacy. In *IMC '11*.

[16] Legion of the Bouncy Castle Inc. Bouncy castle crypto apis. http://www.bouncycastle.org/.

[17] J. R. Levine. Dns based blacklists and whitelists. 2010.

[18] R. Ling and N. S. Baron. Text messaging and im linguistic comparison of american college data. *Journal of Language and Social Psychology*, 2007.

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

[20] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS'01*.

[21] A. Mislove, B. Viswanath, K. P. Gummadi, and P. Druschel. You are who you know: Inferring user profiles in online social networks. In *WSDM '10*, 2010.

[22] T. Narten, R. Draves, and S. Krishnan. Privacy extensions for stateless address autoconfiguration in ipv6. 2007.

[23] P. Olson. Facebook closes 19 billion whatsapp deal. www.forbes.com/sites/parmyolson/2014/10/06/facebook-closes-19-billion-whatsapp-deal.

[24] E. P. Papadopoulos, M. Diamantaris, P. Papadopoulos, T. Petsas, S. Ioannidis, and E. P. Markatos. The long-standing privacy debate: Mobile websites vs mobile apps. In *WWW'17*.

[25] P. Papadopoulos, N. Kourtellis, and E. P. Markatos. The cost of digital advertisement: Comparing user and advertiser views. In *WWW'18*.

[26] P. Papadopoulos, N. Kourtellis, and E. P. Markatos. Exclusive: How the (synced) cookie monster breached my encrypted vpn session. In *Eurosec'18*.

[27] P. Papadopoulos, N. Kourtellis, P. R. Rodriguez, and N. Laoutaris. If you are not paying for it, you are the product: How much do advertisers pay to reach you? In *IMC '17*.

[28] P. Papadopoulos, A. Papadogiannakis, M. Polychronakis, A. Zarras, T. Holz, and E. P. Markatos. K-subscription: Privacy-preserving microblogging browsing through obfuscation. In *ACSAC '13*.

[29] A. Peterson. Bankrupt radioshack wants to sell off user data. but the bigger risk is if a facebook or google goes bust. https://www.washingtonpost.com/news/the-switch/wp/2015/03/26/bankrupt-radioshack-wants-to-sell-off-user-data-but-the-bigger-risk-is-if-a-facebook-or-google-goes-bust/, 2015.

[30] Ratatype. Average typing speed infographic. http://www.ratatype.com/learn/average-typing-speed/.

[31] M. Richtel. F.t.c. moves to halt sale of database at toysmart. http://www.nytimes.com/2000/07/11/business/ftc-moves-to-halt-sale-of-database-at-toysmart.html, 2000.

[32] G. M. Robert-Jan Bartunek, Philip Blenkinsop. Eu fines facebook 110 million euros over whatsapp deal. http://www.reuters.com/article/us-eu-facebook-antitrust-idUSKCN18E0LA, 2017.

[33] S. Rowlands. Mobile messaging: War of the words. Whitepaper, 2014.

[34] D. Solove. Going bankrupt with your personal data. https://www.teachprivacy.com/going-bankrupt-with-your-personal-data/, 2015.

[35] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01*.

[36] J. Van Den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *SOSP'15*.

[37] M. Wachs, M. Schanzenbach, and C. Grothoff. A censorship-resistant, privacy-enhancing and fully decentralized name system. In *CANS'14*.

[38] B. Wellington. The dnsjava project. http://www.xbill.org/dnsjava/, 2002.

[39] J. I. Wong. Here's how often apple, google, and others handed over data when the us government asked for it. https://qz.com/620423/heres-how-often-apple-google-and-others-handed-over-data-when-the-us-government-asked-for-it/, 2016.

[40] Z. Xiao, L. Guo, and J. Tracey. Understanding instant messaging traffic characteristics. In *ICDCS '07*.