DP5: A Private Presence Service

Nikita Borisov
University of Illinois at Urbana-Champaign, United States
nikita@illinois.edu

George Danezis
University College London, United Kingdom
g.danezis@ucl.ac.uk

Ian Goldberg
University of Waterloo, Canada
iang@cs.uwaterloo.ca

ABSTRACT

The recent NSA revelations have shown that "address book" and "buddy list" information are routinely targeted for mass interception. As a response to this threat, we present DP5, a cryptographic service that provides privacy-friendly indication of presence to support real-time communications. DP5 allows clients to register and query the online presence of their list of friends while keeping this list secret. Besides presence, high-integrity status updates are supported, to facilitate key update and rendezvous protocols. While infrastructure services are required for DP5 to operate, they are designed to not require any long-term secrets and provide perfect forward secrecy in case of compromise. We provide security arguments for the indistinguishability properties of the protocol, as well as an evaluation of its performance.

1. INTRODUCTION

> *"We kill people based on metadata."*
> – General Michael Hayden [11]

Many organizations, from hobbyist clubs to activist groups to social media giants, provide a mechanism for their members to engage in real-time online communication with their friends. This is nowadays predominantly done using the federated XMPP [26] protocol with either web-based or standalone clients to access services.

A crucial part of a messaging service is to provide indicators of *presence*: when a person connects to the network, she would like to be informed of which of her friends are currently online. Depending on the exact details of the communication service, she may also wish to be informed of some extra data associated with each of her online friends, such as the friend's current IP address, preferred device, encryption public key, or other metadata useful for establishing communication. Note that the communication itself may then occur in a direct peer-to-peer manner, outside the scope or view of the organization providing presence service.

A typical presence mechanism works by having each user inform the server of who her friends are. Then, whenever those friends log in, they are informed of the user's state (offline or online), and if online, the associated data. Note that the "friend" relation is not necessarily symmetric: if Alice lists Bob as a friend, then Bob will see Alice's presence information, but not necessarily vice versa.

This ubiquitous and straightforward presence mechanism, however, has a significant privacy problem: the server learns the complete list of who is friends with whom, and when each user is online. However, we have recently observed that governments use legal compulsion on service providers to disclose their private data, as was the case for the Lavabit service [25]. On January 2014 the New York Times also revealed documents, leaked by Edward Snowden, demonstrating that online *address books* and *buddy lists* are prime targets for extra-legal surveillance by the United States' and United Kingdom's signal intelligence agencies [19]. As a result, organizations providing presence service may be reluctant to even *hold* this privacy-sensitive metadata.

In this work, we present DP5—the Dagstuhl Privacy Preserving Presence Protocol P.[1] DP5 allows organizations to provide a service offering presence information (and associated data) to their users, while using strong cryptographic means to prevent the organization itself from learning private information about its users, such as their lists of friends.

The key contributions of this paper relate to the design and analysis of DP5, a private presence system. More specifically, we:

- Present a set of security properties, functional requirements, and a desirable threat model for private presence. (§3)
- Describe a design, DP5, that fulfills the security requirements, based on private information retrieval and unlinkable pseudonyms in consecutive epochs. (§4)
- Show that the DP5 security mechanism provides unlinkability, and argue that it also provides forward secrecy even when all infrastructure components are compromised. (§5)
- Evaluate the system performance of all DP5 sub-protocols. (§6)
- Discuss design and implementation options, to strengthen the security of DP5 notably against client compromise. (§7)

2. RELATED WORK

Presence is a fundamental component of a number of real-time communication systems including Voice over IP and chat protocols, such as IRC, XMPP, and MSN.

One obvious design choice to achieve private presence consists of simply running a conventional presence system, or even a full chat server, behind a mechanism providing client and server anonymity, such as a Tor hidden service [16]. While this mechanism may hide the identities of users behind pseudonyms, it does not hide the *relationships* among their pseudonyms, and so can lead to re-identification nonetheless [24].

Laurie's Apres [22] was the first to suggest a privacy-friendly protocol to achieve presence. Apres introduces the notion of epochs (and calls them *ID du jour*) and the basic scheme by which presence information is unlinkable between epochs (through hashing) to prevent traffic analysis. Apres also considers how presence is an essential mechanism to enable further efficient communication, a feature that DP5 aims to preserve. A specific system making use of an Apres-like presence mechanism to facilitate real-time communication is Drac [13], which proposes a simplified presence mech-

[1]The extra 'P' is for extra privacy.

anism based on hashing.

DP5 provides an important additional security property compared with Apres (and Drac that builds on it): it hides the topology of the "friend" graph within each epoch. Since Apres was proposed, a body of work has demonstrated that merely providing unlikability of identifiers between epochs does not prevent de-anonymization of social network graphs if their topology remains the same [24]. This is true even if graphs between epochs are not completely isomorphic due to missing edges or vertices. The DP5 protocol eliminates this de-anonymization possibility by splitting presence into registration and lookups—whereas Apres was confounding the two—and ensuring no topology information leaks.

The Tor Project is in the process of redesigning the Hidden Services mechanism [16], which includes a few mechanisms related to the goals of DP5. Current thinking around hidden services allows for services with secret addresses. To preserve this secrecy, queries for the hidden service to hidden directory services are obscured through blinding their secret "public key" with a key derived from itself and an epoch. The core of this rendezvous mechanism is similar to the goals of the DP5 protocol, and has influenced our ideas around forward secrecy.

Presence is related to naming security. DNSSec [1] and DNSCurve [3] provide reliable mapping between names and low-level Internet protocol addresses. DNSSec has been engineered to facilitate offline signatures, and is therefore not appropriate to translate names to very dynamic information like presence and status. On the other had DNSCurve does support dynamic binding of names to addresses, through stronger channel security. While this provides privacy against network adversaries, and limited traffic analysis protection due to potential local caching, it does not provide the strong forms of traffic analysis protection DP5 was designed for. The GNU Name System [28] and a recent proposal by Tan & Sherr [27] use a Distributed Hash Table that all users maintain to mirror all peer's name records to rendezvous. As with DNSSec, this mechanism is ill-suited to dynamic information due to the slow rate of DHT updates, and does not provide clear privacy guarantees.

3. DESIGN AND SECURITY GOALS

The DP5 service aims to provide a private alternative to presence systems that support real-time communications such as instant messaging or Voice over IP (VoIP). In a nutshell, users are able to register and revoke "friends", and query the service to retrieve the online status of those that listed them as friends, as well as receive a small amount of extra information useful for bootstrapping other security protocols. From a security perspective, subject to some typical cryptographic assumptions, the service does not learn who is friends with whom, the topology of the social network remains secret, and no one is in a position to fake the status of any honest user. This section provides details about the properties and threat model of the DP5 design.

3.1 Presence features

DP5 acts as a presence mechanism, but is also enriched with features that allow it to compose well with, and provide a solid foundation for, other secure protocols.

It is assumed that through an "out-of-band" private channel users have acquired a public key corresponding to each of their friends. This can be done in practice through downloading all public keys, using a physical anonymous mechanism for transferring the key (such as a USB drive or smart-phone), or using a privacy-friendly record retrieval mechanism, such as private information retrieval (PIR).

Once users have the list of public keys of their friends they can use DP5 to perform a number of operations:

Friend Registration. A user Alice is able to use the public key of a user Bob to register Bob as her friend. As a result Bob is authorized to receive Alice's online status and other associated data.

Presence Registration. Alice can register her online status at a particular time period (epoch), along with a small amount of associated data for that time period. The system should eventually detect Alice going offline at a later period and change her status automatically.

Presence Status Query. A user Bob should be able to query the system and retrieve the online status of those users that have registered him as a friend at a particular time period (epoch). In particular we note that both Alice must have registered Bob as a friend, and Bob must issue a query for Alice's status, in order for Alice's status to be provided to Bob. As part of the response to the query, the presence-associated data of Alice is provided to Bob if she is online.

Friend Suspension or Revocation. Finally, Alice or Bob may decide that they wish to not be friends any more. Alice can thus chose to remove Bob from her friends and not advertise to him her presence, and Bob may chose to not query for Alice's presence or associated data. If they only do this temporarily we call the action a presence "suspension", and in the long term call this a presence "revocation".

3.2 Threat model and security assumptions

The DP5 design ensures some security properties for presence subject to some system and cryptographic security assumptions, as well as some limitations on the parties an adversary can control or corrupt. However, the DP5 protocol is extremely robust against passive or active network adversaries. More precisely the security of DP5 rests on the following *threat model*:

Secure end-user hosts. Throughout this work it is assumed that honest users' end systems are secure. In particular DP5 makes use of public-key encryption, for which the long-term private keys must remain confidential. Furthermore, the long-term public keys of a user's friends identify the social network that DP5 aims to protect, and thus, must be stored securely on a user device. The security of end hosts is an orthogonal problem to the one DP5 aims to solve. However, we discuss in Section 7.3 how to best partition an implementation of the DP5 protocol to store any long term keys into secure hardware to protect against some software attacks. Similarly it is assumed that honest services run on secure end systems that can maintain secrecy and integrity as necessary. Servers are engineered to not require long-term secrets, and provide forward secrecy, to mitigate any compromises.

Computational cryptography assumptions. DP5 makes use of a number of cryptographic techniques, and thus assumes that the adversary has not made cryptographic breakthroughs allowing him to bypass them. In particular we assume that the secure channels between honest users and honest infrastructure services provide the necessary authenticity, integrity and confidentiality. We also assume an adversary is not able to violate the properties of a secure pseudo-random function (PRF-IND), secure encryption (IND-CPA) or violate the Decisional Diffie-Hellman (DDH) assumptions or the co-DHP assumption for bilinear groups. (We elaborate on a variant that does not rely on pairing-friendly elliptic curves in Appendix A, at the cost of some extra server-side computation and storage.)

Ubiquitous passive network observer and dishonest users. We assume an adversary can observe all the information that is in tran-

sit between all honest and dishonest participants in the protocols. All security properties should hold even for an adversary with a full record of all network communications between all parties. An adversary can also make use of the presence system both by registering the presence of malicious users, as well as by querying it in any manner.

Threshold of honest infrastructure servers. The DP5 protocol uses a coalition of infrastructure servers to achieve its goals. It is assumed that at least one of those servers does not collude with the others to violate any security properties and executes the protocol correctly. Other servers may be passively dishonest: in such a case they follow the protocol, but share their internal state and secrets with the adversary. The DP5 protocol is designed to maintain all its security properties against such adversaries. It may be the case that some other servers are actively malicious, and do not follow the DP5 protocol. In such a case the DP5 protocol maintains its confidentiality and integrity properties, but may not provide some of its functionality—namely, it may suffer from denial of service. We discuss how to ameliorate this issue in Section 7.4.

Security in the covert model. Finally, some availability aspects of the protocol rely on the "covert security" model, namely that adversaries follow the protocol if deviations would be detected with some non-negligible probability. Specifically, we rely on this model to argue that registration servers would not remove presence entries without due authority.

3.3 Security goals

In this section we present the security goals of the DP5 service. It is worth noting that the security properties described are in relation to the additional information that could be leaked by the presence protocol and not the communication channels used.

- **Privacy of presence.** Only friends of Alice are able to detect whether Alice is or is not online. More formally, an adversary with a transcript of the contents of DP5 protocol interactions cannot distinguish whether Alice was one of the honest participants or not.
- **Integrity of presence.** Only Alice can convince one of her friends that she is online. More formally, if an honest friend of Alice becomes convinced that Alice is online at a particular epoch, it must be the case that Alice has performed the presence registration protocol for that epoch.
- **Privacy for social network.** Either Alice registering friends or her presence, or Bob querying for the presence of his friends, should reveal no information about who their friends are. Given any two lists of friends (up to a public maximum length) for any honest participant in the DP5 protocol, it is indistinguishable to the adversary which of the two lists was used. This holds for all parts of the protocol, including friend registration, presence registration, presence querying, the storage or retrieval of associated data.
- **Unlinkablity between epochs.** User actions are not linkable across epochs to an adversary that is not their friend. Specifically, given a transcript of the DP5 protocol for a specific user at an epoch, and a transcript at a subsequent epoch, an adversary cannot distinguish if the transcripts originated from the same user or different users.
- **Privacy of associated data.** Only friends can recover the plaintext of a user's associated presence data. If the adversary submits to the user two candidate plaintexts, and the user chooses one as their associated data for a specific epoch, the adversary cannot efficiently distinguish which of the two was chosen.
- **Integrity of associated data.** If an honest friend of Alice recovers a plaintext of associated data it must be the case that Alice ran the associated data registration protocol at that epoch, with that plaintext as input.
- **Indistinguishably of offline status, suspension and revocation.** A user Bob—even if Alice had registered him as a friend in the past—cannot distinguish whether Alice is offline, has suspended him, or has revoked him as a friend.
- **Auditability of infrastructure.** All actions that the centralized registration services perform should be publically verifiable. In particular a public append-only log of all actions of registration servers should not violate any security properties.
- **Forward and backward secrecy of infrastructure.** An adversary with the power to extract cryptographic keys from infrastructure servers at some point in time, cannot compromise the security of any past epochs. Once fresh authentication keys are generated future uses of DP5 are also safe.
- **Optional support for anonymous channels.** The DP5 protocol does not leak any additional information about the identity of clients than do the underlying communications channels. In particular, if the communication channels leak no identity, neither does DP5.

We note that the integrity of presence property is enforced by an auditing mechanism, and that the integrity of associated data requires either the mechanism described in Sect. A or the use of digital signatures.

4. THE DP5 PRESENCE PROTOCOL

4.1 Protocol description

The objective of the DP5 protocol is, broadly, for users to advertise their presence status to their friends only, without revealing their social network to any single third party. The protocol assumes a number of participants collaborate to achieve this: users, one of which we call by convention Alice, register their presence in the system to a registration service; users, such as one called Bob, can then query the service to retrieve the status of users for whom they are friends. The service is composed of a registration server, handling the user registration side of the protocol, and a number of private information retrieval (PIR) authorities handling the query side of the protocol.

For clarity of presentation we will pin Alice's role as wishing to advertise her presence to her friend Bob, while Bob only queries the system for Alice's presence. Of course, in practice, all parties partake in both the registration and query protocols, and have multiple friends.

4.2 DP5 setup

The DP5 protocol assumes that Alice and Bob share a cryptographically strong symmetric secret keys K_{ab} (we note that these keys have a "direction", thus the key K_{ba} is also shared but different from K_{ab}). This key can be computed through Alice and Bob generating a public-private key pair using a group G, generated by element g, in which the Decisional Diffie-Hellman problem is believed to be hard (such as the Elliptic curve group of Curve25519 [2]), and securely exchanging the corresponding public keys. An appropriate key derivation function can be used to extract K_{ab} and a different K_{ba}. The DP5 protocol does not require this shared key to be stable in the long term; thus, it is also possible for Alice and Bob to use a mechanism offering perfect forward secrecy to derive the shared key periodically.

The DP5 protocol divides time into short-term epochs, meant to last on the order of a few minutes, and long-term epochs, on

the order of a day. Clients and infrastructure are assumed to have loosely synchronized clocks.

All parties to the DP5 protocol share a common set of cryptographic primitive: three families of keyed pseudo-random functions ($\text{PRF}_K^\ell(m), \ell \in \{1,2,3\}$, implemented using SHA-256 [17]); an authenticated encryption primitive ($\text{AEAD}_K^{IV}(h;m)$)[2] (such as AES [12] in GCM mode [23]); and access to secure channels between clients and infrastructure (using TLS [15]).

Furthermore, DP5 makes use of three generators g_1, g_2 and g_T of groups G_1, G_2 and G_T respectively for which an efficiently computable asymmetric pairing function $e(G_1, G_2) \to G_T$ is known, such that $e(g_1^a, g_2^b) = g_T^{a \cdot b}$. The Decisional Diffie-Hellman problem is assumed to be hard in each of these groups (so that a "type 3" pairing [18], without an efficiently computable isomorphism from G_2 to G_1 or the reverse, is in use), as well as the Co-DHP (aka Co-CDH) [5] problem for G_1 and G_2. An efficiently computable hash function $H_0 : G_T \to \{0,1\}^\eta$ from elements of G_T to η-bit strings is known by all (η is the length of an identifier). Everyone also knows two efficiently computable hash functions $H_1 : \mathcal{T} \to G_2$ (where $\mathcal{T}$ is the set of valid epoch timestamps) and $H_3 : G_1 \to \{0,1\}^\nu$ (where ν is the key size of the PRF[3] function).

Finally, all users share some global parameters, such as a maximum number of friends N_{fmax}, the number N_{pirmax} of PIR servers and their IP addresses, the sequence number and duration of short-term (t_i) and long-term (T_j) epochs, and the byte size of all inputs and outputs of the cryptographic primitives.

4.3 PIR sub-protocol

DP5 uses private information retrieval (PIR) in order to allow clients to retrieve presence information from DP5 servers without revealing to the servers what information is being requested.

A basic PIR primitive is to consider a database of r blocks, each b bits in size, where the client knows the exact index of the block she wishes to retrieve. In DP5, we use *information-theoretic* PIR, in which multiple (non-colluding) PIR servers are employed. The client information-theoretically splits her query across the set of N_{pirmax} servers, and combines their responses in order to reconstruct the data in question. A *non-triviality* requirement is that the amount of data transferred in the protocol is sublinear in the total size of the database (rb bits).

Probably the simplest such scheme is due to Chor et al. [10]. This simple scheme sends r bits to, and receives b bits from, each server, for a total communication cost of $N_{\text{pirmax}} \cdot (r + b)$ bits. However, this scheme is not *robust*: if one of the servers fails to respond, or responds incorrectly, the client will fail to reconstruct her data, and indeed will be unable to identify the server(s) that responded incorrectly.

Goldberg [20] demonstrated a PIR protocol with only marginally larger communication costs: $N_{\text{pirmax}} \cdot (rw + b)$ bits, where w is the bitlength of the underlying finite field (typically $w = 8$). This protocol, however, is able to handle offline and malicious servers. Devet et al. [14] recently further extended the robustness of this protocol, enabling reconstruction of the requested data, and identification of the misbehaving servers, when only $t + 2$ servers are behaving honestly. Here, t is the privacy level: any collusion of up to t servers learns no information about the query. This protocol is implemented in the open-source Percy++ library [21], which we use in our implementation of DP5.

Our situation is not quite as simple as the above protocols provide for, however. Our databases consist of a collection of (key,value) pairs, and our goal is to retrieve the value corresponding to a given database key, rather than a particular block index. To do this, we build upon the block-retrieval PIR primitive above, using an extension to Chor et al.'s hash-based $\mathcal{PERKY}$ protocol [9, §5.3]. This extension works as follows: Let s be the (fixed) size (in *bytes*, as we use $w = 8$) of the (key,value) pair, and let there be n such pairs ready to be inserted into a database at the start of a short-term or long-term epoch. The high-level idea is that we will create $r = \lceil \sqrt{ns} \rceil$ buckets, and use a hash function on the keys to hash the (key,value) pairs into the buckets. The expected size of each bucket is then $\frac{n}{r}$ data items, or $\frac{n}{r}s$ bytes, and using Chernoff bounds, it is easy to see that the probability of one bucket containing more than $\frac{n}{r} + \sqrt{\frac{n}{r}}$ data items is negligible. In practice, we select the hash function by picking ten random PRF keys for a PRF with codomain $\{1, \ldots, r\}$, using each to hash all n keys in the database, and find the largest number of records hashed into any one bucket. We then keep the PRF key that yielded the smallest such largest bucket. That PRF key is made available to DP5 clients when they contact the PIR database servers.

This hash variant is more suitable for our purposes than the perfect hash in $\mathcal{PERKY}$, as our (key,value) records are small in comparison to the number of such records, so we want to have many records in a single PIR database block in order to balance the sending and receiving communication cost. $\mathcal{PERKY}$, on the other hand, uses perfect hashing to put zero or one keys (they do not consider associated values, but this is a trivial extension) into each PIR block, and uses n^2 blocks of size s bytes to accomplish this, while we use $r \approx \sqrt{ns}$ blocks of size about $\left(\frac{n}{r} + \sqrt{\frac{n}{r}}\right) \cdot s \approx \sqrt{ns} + \sqrt[4]{ns^3}$ bytes. As the underlying PIR protocol we use transmits a number of bytes equal to the number of records plus the size of each record, and the computation cost is proportional to the number of records times the size of each record, our hash-based protocol is preferable to that of $\mathcal{PERKY}$ in our environment.

4.4 DP5 registration

Alice registers her presence and associated data for each epoch, by updating a number of databases at epoch t_{i-1} and T_{j-1}, that are made available for all to query at epoch t_i and T_j.

Long-term epoch friendship database. Once per long-term epoch T_{j-1}, Alice updates the *long-term epoch friendship database* for the next long-term epoch T_j with a record for each of the friends she wishes to advertise her presence. The database is an oblivious repository of records for each directed *friend link* in the system; however, note carefully that this database does *not* leak information about the actual friendships to those without the appropriate secret keys. To perform this update, Alice picks a random private key $x \in_R |G_1|$, and derives a fresh public key $P_a^j = g_1^x$. Then for each of her friends she derives the shared key for the long-term epoch, and encodes a database entry comprising an identifier, and a ciphertext of her fresh public key.

For instance, Alice encodes an entry for Bob using their shared key K_{ab} for long-term epoch T_j as follows. She first derives an epoch key using a pseudo-random function and the identifier for the long-term epoch: $K_{ab}^j = \text{PRF}_{K_{ab}}^1(T_j)$. She then creates a public identifier for the key as $\text{ID}_{ab}^j = \text{PRF}_{K_{ab}}^2(T_j)$, and encrypts her public key as $C_{ab}^j = \text{AEAD}_{K_{ab}^j}^0(\text{ID}_{ab}^j; P_a^j)$. The resulting entry is $(\text{ID}_{ab}^j, C_{ab}^j)$.

Alice encodes an entry for each of her friends, and then pads the list of entries with random entries up to a maximum number of friends N_{fmax}. Those random entries are generated by Alice performing the encoding process above using a randomly chosen fresh

[2]We omit h when the AEAD mode is not used to provide integrity over any cleartext data (what the AEAD calls "associated data"—not to be confused with the DP5 associated data).

shared key. She then sends the fixed-size list of entries to the registration server, which stores it. Alice stores the fresh private-public key pair (x, P_a^j) until a new one is generated. We call the public component P_a^j the *ephemeral epoch key* of Alice.

Short-term epoch user and signature database. Once per short-term epoch t_{i-1}, Alice updates the *short-term epoch user database* for epoch t_i with a single entry, denoting she is online, and some associated data m_a. Alice first derives $s_a^i = H_1(t_i)^x$, which represents an unforgeable signature that Alice is online. Furthermore, Alice encrypts her associated data as $c_a^i = \text{AEAD}_{K_a^i}^i(m_a)$. where $K_a^i = \text{PRF}_{H_3(P_a^j)}^3(t_i)$. She then sends the record (s_a^i, c_a^i) to the registration server.

The registration server, upon receiving an entry (s_a^i, c_a^i) from Alice, first derives an identifier $\text{ID}_a^i = H_0(e(g_1, s_a^i))$. Then the server updates two parallel databases: the entry (ID_a^i, c_a^i) is added to the short-term epoch user database for t_i, and the entry (ID_a^i, s_a^i) is added to the short-term epoch signature database.

4.5 DP5 query

At the beginning of epoch T_j the registration service makes public the full long-term epoch friendship database with all entries received during epoch T_{j-1}. Similarly, at the beginning of each short-term epoch t_i, the registration server makes available the separate short term user and signature databases with collected during epoch t_{i-1}. All PIR servers download all databases as soon as they become available.

Furthermore, each PIR server audits at the start of t_i the user database using the entries in the signature database: each entry (ID_a^i, c_a^i) in the user database corresponds to an entry (ID_a^i, s_a^i) in the signature database, such that $\text{ID}_a^i = H_0(e(g_1, s_a^i))$. If the audit succeeds the PIR server proceeds to answer requests for entries in the databases.

Once per long-term epoch T_j Bob queries the long-term friendship database for entries corresponding to each of his friends. First, he reconstructs for all his friends a shared identifier; e.g., for Alice he computes the identifier $\text{ID}_{ab}^j = \text{PRF}_{K_{ab}}^2(T_j)$. He then pads this list of friend identifiers with a number of random identifiers, up to a maximum number of friends N_{fmax}. Finally, using a PIR protocol, he queries the long-term friendship database for the fixed-length list of identifiers. As a result, he receives a list of identifier and ciphertext entries (ID_{fb}^j, C_{fb}^j), one for each of his friends f who registered in epoch T_{j-1}. For each of those entries, for example Alice's, he decrypts ciphertext C_{ab}^j using key $K_{ab}^j = \text{PRF}_{K_{ab}}^1(T_j)$ to yield plaintext P_a^j. Upon valid decryption he updates his knowledge of the public key of Alice with the ephemeral epoch key decrypted, namely P_a^j.

Up to once per short-term epoch t_i, Bob may wish to refresh the status information of his friends including Alice. As a first step, Bob reconstructs the identifier of Alice using the latest public key P_a^j available for her. To this end he computes $\text{ID}_a^i = H_0(e(P_a^j, H_1(t_i)))$ for Alice, and similarly an ID for each of his other friends. He then privately queries the PIR servers for a list of those identifiers, padded with random string to a maximal length of N_{fmax}. Upon completion of the retrieval protocol, Bob decrypts each returned ciphertext entry with a symmetric key derived as $K_a^i = \text{PRF}_{H_3(P_a^j)}^3(t_i)$. If the decryption succeeds the status of the friend is set as "online", and the associated data m_a is returned. Otherwise the friend's status is set as "offline", and no associated data is returned.

4.6 Usage notes

Epoch lengths. The presence mechanism is divided into long-term epochs and short-term epochss, with different performance characteristics. In the long-term epoch the registration phase has a space complexity of $\mathcal{O}(\text{fmax})$, where fmax is the maximal number of friends. However, in the short-term epoch the registration space complexity is merely $\mathcal{O}(1)$, making short-term updates extremely efficient. Both mechanisms require $\mathcal{O}(\text{fmax})$ queries to the database, through the PIR mechanism. However, the size of the database is different in each case: the long term database is larger and contains $\mathcal{O}(N \cdot \text{fmax})$ entries, where N is the number of distinct registered clients, whereas the short-term database only contains $\mathcal{O}(N)$ entries making queries cheaper.

Skipping short-term epochs. A deployment can leverage this asymmetry to provide extremely timely updates. The long-term epoch can be set at the granularity of a day, while the short-term epoch can in the order of magnitude of minutes. Clients register their presence in the long-term epoch, and also at regular intervals in the short-term database. To detect presence it is imperative that all clients have an up-to-date view of their friends' entries from the long-term database, since this enables them to use the short-term update mechanism. This is relatively infrequent, and therefore cheap in terms of processing and bandwidth costs.

Clients can choose, according to available resources, how often they wish to query the short-term database. Short-term queries can be scheduled either for each short-term epoch, periodically but less frequently than the short-term epoch interval, or on-demand when a high-level (observable) action that requires presence information is undertaken. The frequency of updates may depend on load, network availability, or any other non-sensitive information but must not be dependent or adapt to the actual presence information retrieved in previous epochs. Such adaptive strategies create a timing side-channel that the adversary could use to infer the friends of a client—which is exactly what DP5 attempts to obscure.

Suspending presence and revocation. The cost of longer long-term epoch is in terms of inflexibility of suspension or revocation of presence. In order for Alice to selectively revoke a friend Bob, she must not use his Diffie-Hellman public-key when she registers with the long-term friendship database, and also change her short-term public key. Any updates to the long-term friendship database can only take place once the long-term epoch changes, and thus the immediacy of revocation depends on this period being short. Similarly, adding a new friend only fully takes place at the next long-term epoch. However, adding someone can be very fast when an out-of-band channel is available: Alice can exchange with them not only her Diffie-Hellman key but also her public key P_a^j for the current long-term epoch, so they can immediately start querying the short-term user database.

Missing the long-term epoch update. While long-term epochs should be long enough to provide a good window of opportunity for Alice to register, she might occasionally not be able to. In those cases, by convention, her friend may still seek her presence in the short-term database by using her last known ephemeral epoch key. Alice, aware that she missed a long-term update, may reuse her previous fresh public key for one or more additional long-term epochs until she is able to register again in the long-term friendship database. Note that both Alice and her friends can determine which was the last registered public key, through querying past epochs of the friendship database.

Users of the system may be tempted to skip looking up past long-term updates for specific friends, and use the above mechanism as long as there is no change in Alice's list of friends. While this does not lead to any immediate security breach, it is discouraged.

Filling in long-term updates. Alice's friends may also be offline

long enough to miss a particular long-term epoch update. The DP5 protocol assumes that all clients check for their friends' presence each long-term epoch. Therefore it is necessary to request, using the full PIR mechanism, all epochs that have been missed sequentially. Clients may be tempted to only query a subset of recent epochs, until they have identified the latest long-term update for all their friends. While this may be cheaper than requesting all updates, the stopping rule depends on the secret list of friends of a client, resulting in an observer receiving information about the list of friends by observing the number of requests. Consequently we require all clients to sequentially query all long-term epochs, even though this may leak to the adversary how long they have been offline.

Self-checking our own entries. A partial auditing mechanism is included in DP5 through PIR servers checking signatures on the database of entries. However, this only guarantees that malicious misleading entires are not included in the databases, but not that the registration server does not drop valid entries. Each client can perform some limited checks to reduce the likelihood of a malicious registration server not serving the full database. A client can query the database for keys that it registered corresponding to some of its friends, to check they are included and served correctly. The DP5 protocol may be extended with the registration servers providing a signed receipt upon each registration, to allow non-inclusion of records to be verified by third parties.

This auditing mechanism is quite robust: once the databases are provided to the lookup servers selective denial of service by the registration server is no longer possible. Furthermore, the privacy properties of PIR ensure that the queries to the known entries are perfectly private and do not leak any information about the identity or any other secret of the client. Since DP5 requires clients to query a maximum number of entries this audit process can consume unused slots and does not add any extra cost, as long as the clients have fewer friends than the maximum allowed.

5. SECURITY ARGUMENT

We present an analysis of the security properties of DP5.

- **Privacy of presence.** The long-term protocol uses shared keys that are derived via Diffie-Hellman key agreement; an adversary without access to the private keys cannot associate the shared key with a user. In the short-term protocol, Alice's so-called public key, g_1^x, is revealed only to her friends (via the long-term protocol), and thus a registration using the corresponding private key cannot be tied to Alice.
- **Integrity of presence.** In the long-term protocol, presence information is authenticated using AEAD with a shared key. In the short-term protocol, a presence registration requires $H(t_j)^x$, which is a BLS signature [6] on the epoch number t_j. The security of BLS for Type-3 curves was proven by Chatterjee et al. under an assumption they call Co-DHP*, which they show to be equivalent to Co-DHP [5] under a uniform generator assumption [8]. Because this signature is not included in the PIR database, a corrupt registration server collaborating with one of Alice's friends could convince her other friends that she is online, but this will be caught by the PIR servers using the partial auditing mechanism.
- **Privacy for social network.** Just as the long-term registration protocol does not reveal Alice's identity, it likewise hides the identities of her contacts. On the query side, the privacy of the social network is preserved through the PIR subprotocol.
- **Unlinkability between epochs.** The proof of this property is given in Appendix B.
- **Privacy and integrity of associated data.** This is ensured by the AEAD algorithm. Note that in the short-term protocol, the AEAD key is known to all of Alice's friends, and so one friend, collaborating with the registration server, could supply incorrect associated data for the registration. Unlike presence integrity, this will not be caught by the PIR servers' audit. This could be remedied by generating an additional signature on the associated data, at the cost of significantly increasing the size of the short-term database.
- **Indistinguishability of offline status, suspension, and revocation.** To revoke or suspend Bob's access, Alice chooses a new public key P_a^j and does not share it to Bob. To maintain indistinguishability, Alice may generate a separate key $\hat{P_a^j}$ and share it with Bob through the long-term database. Alice can use P_a^j in the short-term registration protocol, but from Bob's point of view, Alice will always appear as offline, as a consequence of the privacy of presence property.
- **Auditability of infrastructure.** The above privacy properties do not rely on the registration server maintaining any secrets and so a public audit log of registration messages will not violate any security properties.
- **Forward secrecy of infrastructure.** The long-term secrets of the PIR servers are used only for establishing TLS connections. Assuming that TLS is used in forward-secure mode, their compromise does not reveal any information about past registrations. (The servers should take care, however, not to store the plaintext PIR requests or responses after their use.) A compromise of the long-term secrets of the registration server can affect integrity only, and thus do not enable the compromise of past information.
- **Optional support for anonymous channels.** Alice's identity is not revealed in the registration protocol, as guaranteed by the privacy of presence and unlinkability between epochs properties. The use of PIR in the query protocol, in turn, fully hides the identity of the querier.

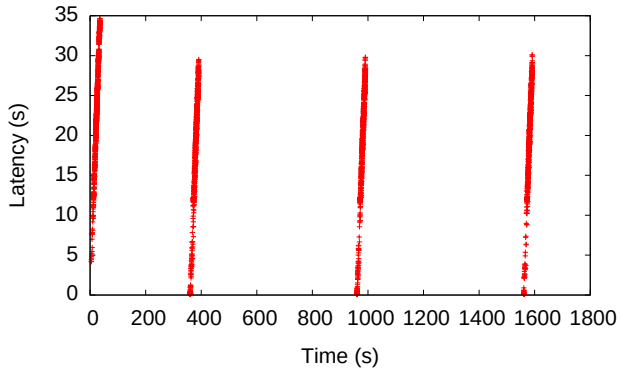
6. EVALUATION

6.1 Implementation

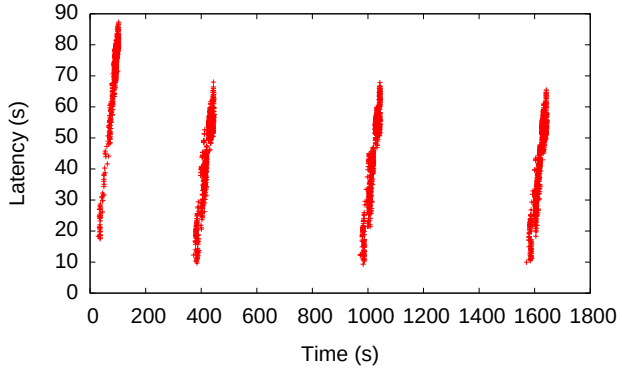
We implemented the DP5 protocol as a set of libraries, as well as clients and servers using those libraries. The cryptographic core relies on OpenSSL for AES and hash operations, as well as the TLS channels between clients and servers (using self-signed certificates). The AEAD function is instantiated with 128-bit AES in Galois Counter Mode (GCM) and an all-zero IV (since all keys used are fresh). Pairing-friendly curves are provided by the RELIC library, and we use the Percy++ library [21] for the robust PIR functions. The DP5 library is implemented in C++ (1000 lines of *.h* files, and 4800 lines of *.cpp* files), and the network code in Python 2.7 (2700 lines of *.py* files). These include unit test code, functional test code, integration tests, and experimental setup code. All client-server interactions are encoded as web requests using the lightweight Cherrypy framework, and both clients and servers are build around the Twisted non-blocking network libraries. The core library interfaces with the high-level network code using both native bindings and the CFFI foreign call interface for Python. We will be releasing our code under an open-source license.

6.2 Performance

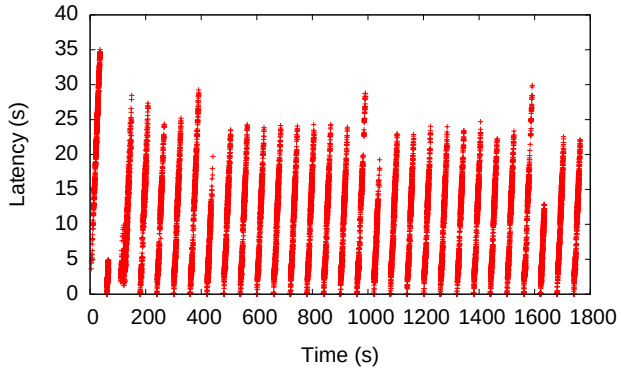
To evaluate the performance of DP5, we ran 960 simultaneous clients accessing the DP5 infrastructure. The clients were running on an 80-core Xeon 2.4 GHz server with 1 TB of RAM. For



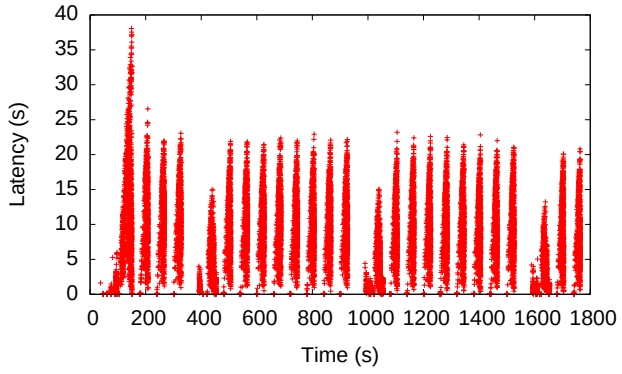
(a) Registration (long-term)



(b) Lookup (long-term)



(c) Registration (short-term)



(d) Lookup (short-term)

Figure 1: An execution trace of 960 clients.

Table 1: Data sizes

	Long-term		Short-term	
	Req	Resp	Req	Resp
DB size	12 MiB		80 KiB	
Registration	9 004 B	5 B	164 B	5 B
Lookup	300 KiB	500 KiB	5 B	80 KiB[3]

each of the short-term and long-term protocols, we used one server for registration and three servers supporting PIR (8 servers total). Each server was running on a 16-core Xeon 2.0 GHz machine with 256 GB of RAM. The machines were interconnected using 1 Gbps Ethernet.

Figure 1 shows the trace for 30 minutes of execution. Each point corresponds to a completed operation, with the x-coordinate representing the time of completion and the y-coordinate the latency from the time the operation was started. To better understand the impact of both short- and long-term epoch changes, we set their length to be 1 and 10 minutes, respectively. Clients are roughly synchronized and begin registration and lookup requests at the beginning of each epoch, which results in a roughly linear pattern of responses observed in the graph. The performance is somewhat worse in the first epoch as new TLS connections must be established to the servers; the connections are reused for future queries. Note that when a long-term epoch changes, clients are unable to issue meaningful PIR requests to the short-term database until they have received the (potentially) updated keys from the long-term database; this can be seen in the smaller number of short-term lookup requests every 10 minutes (Figure 1d). A more realistic value for the long-term epoch would be 24 hours, minimizing the impact of this problem. Further, clients could optimistically use previous keys in lookups, failing only in cases of key change.

Figure 2 summarizes the user-facing latency of the operations over a 5-hour execution. The box plot depicts the interquartile range, with a line representing the median, and the range of 99% of the points, with outliers plotted as individual points. We emphasize that these delays are a consequence of the synchronized behavior of the clients. For the short-term epoch, we expect this to represent real-world behavior, but for the long-term epoch, clients will come online during different parts of an epoch and thus experience lower delays. Monitoring the behavior of the servers, the PIR servers for the long-term database had high CPU utilization for the approximately 90 seconds following an epoch change; other servers were minimally utilized and thus appear to be able to support significantly larger numbers of clients.

Table 1 lists the sizes of the requests and responses in our protocol, excluding the overhead from the HTTP and TLS protocols.

6.3 Scaling

The bottleneck server-side lookup operations involved in DP5 are easily parallelizable and thus more resources can be deployed to support larger user populations. As the number of users grows, the database size will grow linearly with the number of users. Our PIR implementation uses bandwidth that grows as $\Theta(n^{1/2})$ with respect to the size of the database and thus remains practical for significantly larger user populations. The *per-user* server-side computation for PIR, however, is linear in the database size.

The long-term PIR server is the most resource-intensive compo-

[3]Our implementation automatically selected the trivial PIR protocol (i.e., downloaded the entire database) for the short-term protocol as this was more bandwidth efficient than PIR for this size of database.

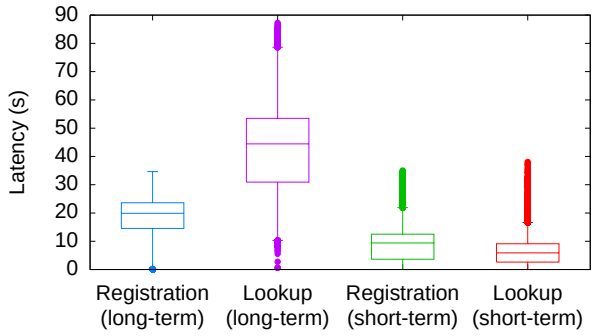


Figure 2: Overall user-facing latency for the registration and lookup operations

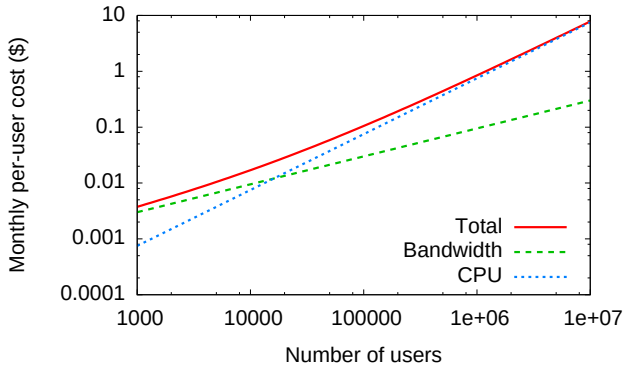


Figure 3: Per-user cost for the bandwidth and CPU associated with running a long-term PIR server with a 24-hour epoch.

nent of DP5. In our experiments it used about 15 CPU-minutes and 1 GB of bandwidth per epoch. Figure 3 plots an estimate of the cost of running such a PIR server, using $0.10 as the cost of one CPU-hour and 1 GB of data transfer[4], and a 24-hour epoch. A user population of 1 000 can easily be supported even with volunteer resources, whereas a subscription service can support as many as 10 million users at a per-user cost of less than $10/month.

7. DISCUSSION

7.1 Channel anonymity

The DP5 design allows clients to access presence services through anonymous, pseudonymous or authenticated channels. The presence service is guaranteed to preserve the properties of the channel and leak no more information about the identity of the clients, and their friends, than the channel already would allow an adversary to infer.

For clients that use DP5 over authenticated or pseudonymous channels, it provides relationship anonymity only. An adversary does not learn the friends of any clients but can observe a specific client or pseudonym being online / offline. This information is leaked by the channel, not the presence service.

Using the DP5 services over an anonymous channel provides both relationship anonymity, and unlinkability across long-term and short-term epochs, vis-a-vis the presence service. However, most deployed anonymity systems do not provide full unobservability, and therefore do leak when a network end-point is using the anonymity network and when it is out of it. Therefore, despite an adversary observing the DP5 infrastructure not being able to infer the online / offline profile of a client, it might be able to do so if the client is under direct observation. Channels that offer unobservable access to anonymity networks may mitigate against this attack.

7.2 Suspension, revocation, and pseudonyms

We divide time into short-term and long-term epochs in order to balance up-to-date presence with timely revocation. For Alice to revoke Bob she removes his public key from the long-term update mechanism, and refreshes her short-term public key. This results in Bob not being able to retrieve the next fresh short-term public key for Alice, and so he cannot query her short-term presence or associated data.

Alice may selectively allow Bob to query her presence in specific epochs. However, once Bob has access to her key for a specific long-term epochs, the presence mechanism is all-or-nothing: either all friends, including Bob, get updated or none does.

To achieve finer temporal control over which friends can or cannot see updates from Alice, she has to use multiple pseudonyms. This can be achieved by dividing her friends into a mutually exclusive sets, and providing each set with a different fresh short-term public key. During any short-term epoch Alice can register with all, or any subset of the ephemeral public keys to advertise her presence to different sets of friends. However, registering multiple pseudonyms during a short-term update is susceptible to traffic analysis. An adversary can observe the number of short-term updates originating from Alice to infer the number of sets of friends she advertises to, whether she is not advertising to some sets, and even to identify her between different short-term or long-term epochs if the number of pseudonyms is atypical. For this reason advertising multiple pseudonyms is best done when using anonymous channels, by repeating the full short-term registration protocol once for each pseudonym.

7.3 Forward secrecy and hardware stores

Various parts of the DP5 protocol have been designed, or can be easily altered, to provide stronger guarantees against end-point compromise. Purely cryptographic mechanisms can provide forms of forward secrecy, preventing retrospective privacy violations in case keys are compromised. Hardware modules with a narrow interface can be used to prevent long-term secrets being extracted in case the software stack of clients is compromised, as was the case in the recent Heartbleed attacks against OpenSSL.

By design, long-term epoch registration relies on a derived key K_{ab} that is the result of a Diffie-Hellman exchange. Once this shared key between Alice and Bob is derived there is no need to store the public or private keys that were used for the derivation any more. Simply using fresh key pairs with different friends, and deleting them after the first key derivation has taken place, is good practice.

In the DP5 design subsequent long-term epoch keys, K^j_{ab}, are derived using the master shared key K_{ab} and the epoch identifier T_j. This enables the storage of long term shared keys into a hardware security module that only exports epoch key K^j_{ab}. As a result, if a client is compromised, only the keys relating to the current

[4]This cost is arrived at by rounding up Amazon's EC2 prices (`https://aws.amazon.com/ec2/pricing/`). We note that cloud-computing providers are not an ideal site for a PIR server, as the provider could not necessarily be trusted to remain honest and preserve users' privacy, but they do provide a useful baseline for estimating the costs of computing resources.

long-term epoch are accessible. Once the intrusion has been detected, subsequent keys should still be safe. It is important to note that even this limited compromise has profound consequences that are not limited in time: once an adversary has access to Alice's secrets for one epoch, they can determine who her friends are. Thus this mechanism only protects future updates of Alice's friends list.

Another option provides some limited form of forward secrecy: we can modify the key derivation for long-term epochs to be $K_{ab}^{j} = \text{PRF}^{1}_{K_{ab}^{j-1}}(T_j)$. Since the long-term epoch shared key now only depends on the previous long-term epoch keys, previous keys can be securely deleted. This means that past databases cannot be analyzed by an adversary who compromises the keys. This mechanism does not protect future updates once keys are compromised.

Importantly, despite hardware storage of keys or the alternate derivation of keys, a compromise not only leaks the presence and status of Alice for some epochs but also of all of her friends who have authorized her to read their status. This, we believe, is a fundamental limitation of any private presence protocol: in case a user is compromised the presence of all information they were authorized to read, including the presence and status of their friends, is compromised. Thus, presence privacy seems inevitably more fragile than end-to-end encryption for which perfect forward secrecy can be achieved, but rather similar to group private communications, or long-term informational leakage on social networks.

7.4 Protecting Availability

Ensuring availability against malicious clients, servers and third parties is especially important for protocols supporting privacy, as traditional approaches, such as logging, blacklisting or auditing may not be applicable.

The first challenge is to ensure the presence database remains small, by preventing malicious clients from adding a large number of entries. If the channels are authenticated, only the confidentiality of who is friends with whom is maintained (but not the privacy of when Alice is online). In that case, traditional authentication can be used to ensure Alice only updates once per long-term and short-term epoch. In case Alice uses an anonymous channel, requiring authentication would compromise anonymity. In this case, an n-periodic anonymous ticketing scheme, such as the one proposed by Camenisch et al. [7] may be used to anonymously limit registrations per user.

A second challenge is to ensure that the registration servers do not drop entries. To ensure that Alice's entries have been added to an epoch Alice may add herself to her friend list, and check that her record is correctly returned by the servers. We note that due to the cryptographic properties of our scheme it is infeasible for the registration service or the lookup services to selectively drop entries for specific friends of Alice's, since they are indistinguishable. This mechanism may be turned into a robust auditing framework by requesting signed receipts from servers for registration and lookups—since the database is public, this allows any third party to verify a claim that they did not include or serve a specific challenge record. Finally, lookup servers may modify the database to drop records. We use robust PIR [14] that ensures that a malicious server would be detected. Preventing DoS requires at least $t + 2$ honest servers, where t is the maximum number of servers that the threat model allows to collude to determine the query.

7.5 Implementation lessons

Implementing and measuring the DP5 protocols provides us with some insights on how to improve this type of protocol in the future; we attempt to share these insights in this section.

The DP5 design assumes that a different ephemeral key is generated and communicated between friends at each long-term epoch. While this provides forward secrecy, is also creates a sequential dependency between the long-term protocols and the short-term protocols. As a result, all on-line clients must successfully query the long-term friendship database before even attempting to query the first epoch of short-term epoch database. This creates a significant amount of congestion and delay. It is preferable to only require the long-term friendship database to be updated when the friendship graph changes, and provide a mechanism for clients to only retrieve the differences, which should be small (we call this design a *delta database*, but do not explore it further in this work).

Congestion becomes a problem in protocols that require each client to query each of the epochs at least once, as the number of clients increases. The current design of DP5 is not responsive to such congestion, and clients will keep retrying to query overloaded servers effectively performing an unwitting Denial of Service attack. It is clear that a control loop is necessary to regulate the length of an epoch, given the degree of congestion experienced by the lookup servers—the most loaded in the design. There is surprisingly little prior work on how to *design secure control loops*: If an adversary is able to modulate the load on the servers by performing multiple queries, or simply lying about their load, a naive control loop would increase the epoch length and as a result degrade forward secrecy properties or increase the latency of revocation events. Thus secure load monitoring would be needed to implement secure control loops, which is beyond the scope of DP5.

8. CONCLUSION

We present DP5, the first private presence mechanism to leak no information about the topology of a contact list network. We show that the service can be realized while relying only on ephemeral secrets on a set of distributed infrastructure servers. Thus, querying the status of friends cannot be used in the future to trace them or de-anonymize the users. Furthermore, care has been taken to design a protocol that may be used when users are known to the infrastructure, but also when users are anonymous—without leaking any additional information about their identity.

Overall, the protocols are scalable to small deployments of a few thousands, to tens of thousands of concurrent clients, a size suitable for small NGO or cooperative ISP. The key scalability bottle neck is the private information retrieval scheme, and any improvement in PIR would directly translate to an improvement in the performance and scalability of DP5.

Finally, DP5 supports real-time presence, but its latency is determined by the length of the short-term epochs. It is an open problem, and a challenge to the research community, to devise protocols that could reduce this latency radically, while requiring overall low bandwidth.

Acknowledgments

The authors would like to thank Claudia Diaz (KUL) and Roger Dingledine (Tor) for key discussions relating to the design of DP5, and Schloss Dagstuhl seminars for hosting those important design discussions. We also thank Harry Halpin (W3C) and Elijah Sparrow (LEAP) for their feedback on presence requirements, Microsoft Research Cambridge for hosting two of the authors, and Daniel J. Bernstein for suggesting that a pairing-free variant of the protocol should be possible. This material is based upon work supported by the National Science Foundation under Grant No. 0953655, and by NSERC, ORF, and The Tor Project. This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

9. REFERENCES

[1] Roy Arends, Rob Austein, Matt Larson, Dan Massey, and Scott Rose. DNS Security Introduction and Requirements. RFC 4033, http://www.ietf.org/rfc/rfc4033.txt, 2005.

[2] Daniel J. Bernstein. Curve25519: New diffie-hellman speed records. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[3] Daniel J. Bernstein. DNSCurve: Usable security for DNS. http://dnscurve.org/, 2009.

[4] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.

[5] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, number 2656 in Lecture Notes in Computer Science, pages 416–432. Springer Berlin Heidelberg, January 2003.

[6] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology—ASIACRYPT 2001*, number 2248 in Lecture Notes in Computer Science, pages 514–532. Springer Berlin Heidelberg, January 2001.

[7] Jan Camenisch, Susan Hohenberger, Markulf Kohlweiss, Anna Lysyanskaya, and Mira Meyerovich. How to win the clonewars: efficient periodic n-times anonymous authentication. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM Conference on Computer and Communications Security*, pages 201–210. ACM, 2006.

[8] Sanjit Chatterjee, Darrel Hankerson, Edward Knapp, and Alfred Menezes. Comparing two pairing-based aggregate signature schemes. *Designs, Codes and Cryptography*, 55(2-3):141–167, May 2010.

[9] Benny Chor, Niv Gilboa, and Moni Naor. Private Information Retrieval by Keywords. Technical Report 1998/003, IACR, 1998. http://eprint.iacr.org/1998/003.ps.

[10] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *FOCS'95: Proceedings of the 36th Annual Symposium on the Foundations of Computer Science*, pages 41–50, Oct 1995.

[11] David Cole. We Kill People Based on Metadata. New York Review of Books, May 10 2014.

[12] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer, 2002.

[13] George Danezis, Claudia Diaz, Carmela Troncoso, and Ben Laurie. Drac: An Architecture for Anonymous Low-Volume Communications. In *Privacy Enhancing Technologies*, pages 202–219. Springer, 2010.

[14] Caset Devet, Nadia Heninger, and Ian Goldberg. Optimally Robust Private Information Retrieval. In *21st USENIX Security Symposium*, Aug 2012.

[15] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[16] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *USENIX Security Symposium*, pages 303–320. USENIX, 2004.

[17] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, 9 2001.

[18] Steven D. Galbraith, Kenneth G. Paterson, and Nigel P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, September 2008.

[19] James Glanz, Jeff Larson, and Andrew W. Lehren. Spy Agencies Tap Data Streaming From Phone Apps, January 27 2014.

[20] Ian Goldberg. Improving the robustness of private information retrieval. In *IEEE Symposium on Security and Privacy*, pages 131–148. IEEE Computer Society, 2007.

[21] Ian Goldberg, Casey Devet, Paul Hendry, and Ryan Henry. Percy++ project on SourceForge, June 2013. http://percy.sourceforge.net/.

[22] Ben Laurie. Apres—a system for anonymous presence. http://www.apache-ssl.org/apres.pdf, 2004. Technical report.

[23] David A McGrew and John Viega. The security and performance of the galois/counter mode (gcm) of operation. In *Progress in Cryptology-INDOCRYPT 2004*, pages 343–355. Springer, 2005.

[24] Arvind Narayanan and Vitaly Shmatikov. De-anonymizing social networks. In *IEEE Symposium on Security and Privacy*, pages 173–187. IEEE Computer Society, 2009.

[25] Dominic Rushe. Lavabit founder refused FBI order to hand over email encryption keys. The Guardian, October 3 2013.

[26] Peter Saint-Andre, Kevin Smith, and Remko TronCon. *XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies*. O'Reilly Media, 1st edition, 2009.

[27] Henry Tan and Micah Sherr. Censorship resistance as a side-effect. In *Security Protocols Workshop*, 2014.

[28] Matthias Wachs, Martin Schanzenbach, and Christian Grothoff. On the feasibility of a censorship resistant decentralized name system. In *6th International Symposium on Foundations & Practice of Security (FPS 2013)*, 2013.

APPENDIX

A. DP5 WITHOUT PAIRINGS

The short-term DP5 registration and update protocols make use of public key primitives over pairing-friendly curves. This is necessary for Alice and Bob to compute a signed short-term epoch dependent tag to detect Alice's presence and data. The signature prevents any third party, or even friend of Alice, from forging her presence status. Daniel J. Bernstein noted that these properties can be achieved without the use of pairing-friendly curves, but instead conventional elliptic curves that support secure digital signatures such as Ed25519 [4], as described next.

As part of the long-term registration, Alice stores in her status associated data a public key $P_a^j = g^x$ that is a point on an appropriate elliptic curve. During short-term epoch i each friend of Alice derives a variant of the public key as $P_a^{ji} = P_a^j \cdot g^{H_4(P_a^j \| i)}$, where H_4 is a secure hash function. Only Alice and friends of Alice can derive this public key since it requires knowledge of the public key P_a^j; furthermore, those public keys are unlikable across short-term epochs. Furthermore, Alice can construct the private key corresponding to this public key which is $x_i = x + H_4(P_a^j \| i)$. Both Alice and her friends can also derive a symmetric key $K_a^i = H_5(P_a^j \| i)$ to protect the confidentiality of associated data.

To perform the short-term registration protocol, Alice stores in the database the tuple $(P_a^{ij}, \text{AEAD}^i_{K_a^i}(m_a))$. Furthermore, Alice sends to the registration server a signature of the tuple under the

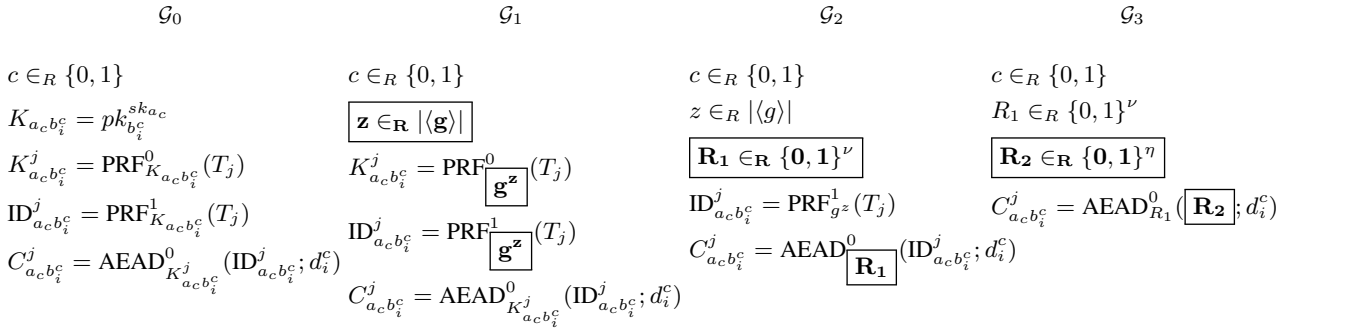


Figure 4: Computing the challenge message for contact b_i^c in successive games for the long-term protocol. η here is the length of the public identifier.

key x_i. The registration server checks that the signature verifies under the verification key included as the first element of the tuple, and then includes the tuple in the database. The full list of tuples and signatures is made available to the lookup servers for auditing purposes.

Finally, Bob can use the short-term epoch public keys of his friends—P_a^{ij} in the case of Alice—to look up their records in the database. The associated data can be decrypted using the symmetric key K_a^i.

This variant of the DP5 protocol has the advantage that is does not require any pairings, and thus the clients require fewer security assumptions, and fewer dependencies on cryptographic libraries. It also allows for the associated data to be signed by Alice, and therefore it is unforgeable subject to the security of the auditing mechanism. On the downside, this mechanism requires an additional signature on the data, which in the original DP5 is integrated with the tag generation. This overhead increases the size of the short-term database, which linearly increases the cost of each PIR query over it.

B. SECURITY PROOF OF UNLINKABILITY BETWEEN EPOCHS

Registration Unlinkability of the long-term protocol. We define the following game to represent registration unlinkability:

1. The adversary supplies the challenger with:
   - Two usernames, a_0 and a_1
   - A number of friends for the registration protocol, n
   - Two sets of n friends, $B^0 = \{b_i^0\}_{i=1}^n$ and $B^1 = \{b_i^1\}_{i=1}^n$
   - Two sets of per-user data, $D^0 = \{d_i^0\}_{i=1}^n$ and $D^1 = \{d_i^1\}_{i=1}^n$
   - An epoch T_j
2. The challenger generates a key for all users in $U = \{A_0, A_1\} \cup B^0 \cup B^1$ and sends the public keys to the adversary.
3. The challenger flips a coin to select a bit $c \in_R \{0,1\}$.
4. The challenger creates a registration message from user A_c for epoch T_j, with contact set B^c and per-user data set D^c and sends it to the adversary.
5. The adversary may ask the challenger to generate keys for any other username $u \notin U$. The challenger returns a public/secret key pair to the adversary and keeps track of these new usernames by adding them to a set U'.
6. The adversary may then query for registration messages generated by any user $u \in U \cup U'$ in an arbitrary epoch, with arbitrary lists of contacts taken for $U \cup U'$ and per-user data, with the restriction that users a_0 and a_1 cannot be asked to register again during the epoch T_j.
7. The adversary outputs its guess for the bit c.

We will first consider the case where $n = 1$; i.e., the challenge registration includes a single contact. We define the following sequence of games, in which the challenges changes the way that the challenge message is computed, as illustrated in Figure 4.

- $\mathcal{G}_0$ is the game where the challenger behaves correctly.
- In $\mathcal{G}_1$, the challenger replaces the shared key $K_{a_c b_1^c}$ with g^z, for a uniformly chosen z, instead of the key computed by Diffie-Hellman. Note that any adversary $\mathcal{A}$ that can distinguish between $\mathcal{G}_0$ and $\mathcal{G}_1$ with advantage ϵ can be turned into an adversary $\mathcal{A}'$ that solves the Decisional Diffie-Hellman problem with the same advantage: given a DDH triple (g^x, g^y, g^z), $\mathcal{A}'$ runs the challenger algorithm, using g^x as the public key for a_c, g^y as the public key for b_1^c, and g^z as their shared key. (To compute the shared secret between a_c or b_1^c and any other contact, the challenger can make use of that contact's secret key.) Observe that if $z = xy$, this is equivalent to $\mathcal{G}_0$ and if z is random, this is equivalent to $\mathcal{G}_1$.
- In $\mathcal{G}_2$, the challenger proceeds as in $\mathcal{G}_1$, but replaces $K^j_{a_c b_1^c}$ with R_1 chosen uniformly at random. Any adversary who can distinguish $\mathcal{G}_1$ from $\mathcal{G}_2$ with advantage ϵ can be turned into an adversary who distinguishes PRF^0 from random with the same advantage, since using a random function instead of $\text{PRF}^0_{g^z}$ in $\mathcal{G}_1$ turns it into $\mathcal{G}_2$. (Note that $\text{PRF}^0_{g^z}$ is only ever evaluated in the computation of the challenge message, except with a negligible probability.)
- In $\mathcal{G}_3$, we likewise replace $\text{ID}^j_{a_c b_1^c}$ with R_2 chosen uniformly at random. As before, an adversary who can distinguish between $\mathcal{G}_3$ and $\mathcal{G}_2$ can be transformed into an adversary who can distinguish PRF^1 from random.
- In $\mathcal{G}_3$, the registration message is $(R_2, \text{AEAD}^0_{R_1}(R_2; d_c))$. Any adversary who has advantage ϵ in $\mathcal{G}_3$ can be directly translated into an IND-CPA adversary for the AEAD function.

For $n > 1$, we can iterate this sequence of games n times: $\mathcal{G}_{0,1} = \mathcal{G}_0, \ldots, \mathcal{G}_{3,1} = \mathcal{G}_3, \mathcal{G}_{0,2} = \mathcal{G}_{3,1}, \ldots, \mathcal{G}_{3,2}, \ldots, \mathcal{G}_{3,n}$. In a game $\mathcal{G}_{i,j}$ we replace the keys / PRFs involving a_c and b_j^c for some b. Therefore, if the advantage of any adversary in solving DDH, PRF-IND, or IND-CPA is always negligible, the advantage of any adversary in the full registration unlinkability game will be likewise negligible.

Note that the game here provides static security, with the adversary declaring the users involved in the contact message ahead of time. The proof can be extended to an adaptive adversary who declares the challenge users after seeing some of the users' public

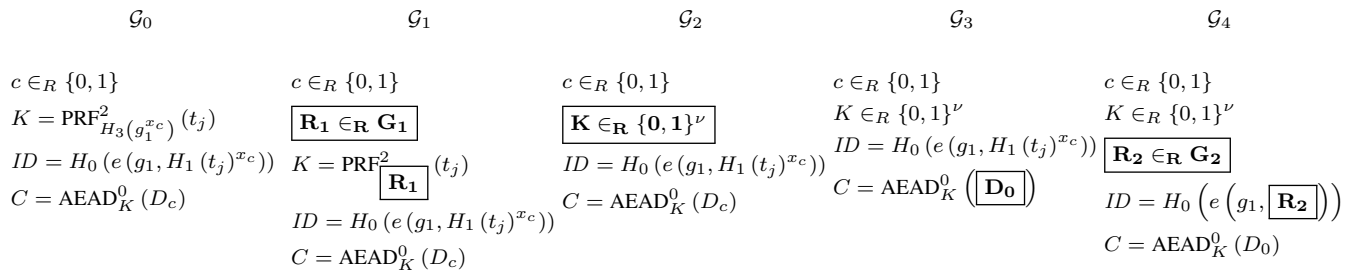


Figure 5: Computing the challenge registration message (ID, C) in successive games for the short-term protocol.

keys by, in each game, having the challenger guess which user will be chosen for a_c/a_c and b_j^c and aborting if the guess was wrong, at the cost of the reduction no longer being tight.

Registration Unlinkability of short-term protocol. We define an unlinkability game similar to the previous one.

1. The adversary supplies the challenger with:
   - Two usernames, A_0 and A_1
   - Two pieces of auxiliary data, D_0, D_1
   - An epoch t_j
2. The challenger generates secret keys x_0 and x_1.
3. The challenger flips a coin to select a bit $c \in_R \{0,1\}$.
4. The challenger produces a registration message (ID, C) for user A_c with data D_c, as shown in Figure 5, game $\mathcal{G}_0$.
5. The adversary may perform a polynomial number of queries Register (i, D, t'_j), which will result in a registration message produced by user A_i with data D and epoch t'_j, as long as $t'_j \neq t_j$.
6. The adversary outputs its guess for the bit c.

We start with $\mathcal{G}_0$, where the challenger behaves as defined above, and make successive modifications to the computation of the challenge message, as shown in Figure 5. In $\mathcal{G}_1$, we replace $H_3(g_1^{x_c})$ in the computation of the challenge registration message with a random number R_1. Note that an adversary $\mathcal{A}$ cannot distinguish between $\mathcal{G}_0$ and $\mathcal{G}_1$ unless it queries H_3 with $g_1^{x_c}$ as the input. If this happens with a non-negligible probability, we can construct an adversary $\mathcal{A}'$ that will solve the computational Co-Diffie-Hellman (co-CDH) problem [5] with the same probability. In the co-CDH game, we are given $h_2, h_2^\alpha \in G_2$ and $h_1 \in G_1$ and asked to produce h_1^α. Our adversary $\mathcal{A}'$ acts as a challenger for $\mathcal{A}$, by following the game $\mathcal{G}_1$, setting $g_1 = h_1$. Instead of choosing x_c explicitly, it implicitly sets $x_c = \alpha$. During queries to the random oracle $H_1(t_k)$, $\mathcal{A}'$ chooses a random z_k and returns $H_1(t_k) = h_2^{z_k}$. Therefore, whenever a registration message needs to be computed for A_c, $\mathcal{A}'$ computes $H_1(t_k)^{x_c}$ as $(h_2^\alpha)^{z_k}$. Additionally, for every query of $H_3(w)$, $\mathcal{A}'$ checks whether $e(w, h_2) = e(g_1, h_2^\alpha)$. If so, then $w = h_1^\alpha = g_1^{x_c}$ and it outputs it as the solution for the co-CDH problem.

In game $\mathcal{G}_2$, we generate K randomly; since in $\mathcal{G}_1$ it is the output of a PRF with a random key, distinguishing $\mathcal{G}_1$ from $\mathcal{G}_2$ with a non-negligible advantage produces an adversary with the same advantage in the PRF-IND game.

In game $\mathcal{G}_3$, the challenger behaves as in $\mathcal{G}_2$, but always supplies D_0 to the AEAD encryption in the challenge message. Any adversary that distinguishes between $\mathcal{G}_3$ and $\mathcal{G}_2$ with advantage ϵ can be turned into an adversary that wins the IND-CPA game for the AEAD function with the same advantage.

Finally, in game $\mathcal{G}_4$, the challenger replaces $H_1(t_j)^{x_c}$ with a random element of G_2. Any adversary who can distinguish between $\mathcal{G}_3$ and $\mathcal{G}_4$ can be turned into an adversary who wins the DDH game in G_2: given a DDH triple (g_2^x, g_2^y, g_2^z) it sets $H_1(t_j)$ to g_2^x and $H_1(t_j)^{x_c}$ to g_2^z. To be able to respond to registration queries for A_c in other epochs, the challenger sets $H(t'_j) = g_2^r$ for some random r, and uses $(g_2^y)^r$ for $H(t'_j)^{x_c}$.

Note that in game $\mathcal{G}_4$, the challenge message is computed independently of c, and hence the adversary can guess c correctly with probability at most $1/2$.