Nikita Borisov*, George Danezis*, and Ian Goldberg*

# DP5: A Private Presence Service

**Abstract:** Users of social applications like to be notified when their friends are online. Typically, this is done by a central server keeping track of who is online and offline, as well as of all of the users' "buddy lists", which contain sensitive information. We present DP5, a cryptographic service that implements online presence indication in a privacy-friendly way. DP5 allows clients to register their online presence and query the 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 scalability and performance.

DOI 10.1515/popets-2015-0008
Received 2014-11-15; revised 2015-05-15; accepted 2015-05-15.

## 1 Introduction

> *"We kill people based on metadata."*
> – General Michael Hayden [16]

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 [38] 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 auxiliary data associated with each of her online friends, such as the friend's current IP address, preferred device, encryption public key, or other information 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 the 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 auxiliary data. 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, it has been recently revealed that governments exercise legal compulsion powers on service providers to disclose their private data, as was the case for the Lavabit service [37]. In 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 surveillance by the United States' and United Kingdom's signal intelligence agencies [29]. This illustrates the surveillance value of contact tracing and presence information and 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 auxiliary 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. (§2)
- Describe a design, DP5, that fulfills the security requirements, based on private information retrieval and unlinkable pseudonyms in consecutive epochs. (§3)
- Show that the DP5 security mechanism provides unlinkability, and argue that it also provides forward secrecy even when all infrastructure components are compromised. (§4)
- Evaluate the system performance of all DP5 subprotocols. (§5)

***Corresponding Author: Nikita Borisov:** University of Illinois at Urbana-Champaign, E-mail: nikita@illinois.edu
***Corresponding Author: George Danezis:** University College London, E-mail: g.danezis@ucl.ac.uk
***Corresponding Author: Ian Goldberg:** University of Waterloo, E-mail: iang@cs.uwaterloo.ca

1 The extra 'P' is for extra privacy.

BY-NC-ND

– Discuss design and implementation options to strengthen the security of DP5, notably against client compromise. (§6)

# 2 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.

## 2.1 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 users have established a shared secret key "out-of-band" with each of their friends. This can be done in practice using a Diffie-Hellman key agreement, after downloading all one's friends' public keys, using a physical anonymous mechanism for transferring the key (such as a USB drive or smartphone), 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 perform a number of operations, through the DP5 infrastructure.

**Friend & Presence Registration.** A user Alice is able to use a secret key she shares with Bob to register Bob as her friend. As a result Bob is authorized to receive Alice's online status and other auxiliary 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.

Alice may then register her online status at a particular time period (epoch), along with a small amount of auxiliary data for that time period. The registration is valid for the duration of the epoch only and Alice's status is automatically changed to offline in the next epoch unless she re-registers. Registration is facilitated in DP5 though protocols with a registration server.

**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 auxiliary data of Alice is provided to Bob if she is online. Status queries are implemented through protocols between users and DP5 lookup servers.

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

## 2.2 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 we assume that honest users' end systems are secure. In particular DP5 makes use of public-key encryption, for which the long-term private keys of users 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 6.3 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. We similarly assume 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 transit 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, particularly to implement information-theoretic PIR (IT-PIR), an inherently multi-server protocol (see §3.3). 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 6.4.

The assumption of a threshold of honest servers is standard for building privacy technologies. Most practical anonymizers, including onion routing and mix networks, rely on a threshold of honest servers to provide privacy properties. These servers can either be provided commercially, as was done in the Freedom network [10], by volunteers, as is the case in Tor [25], or by privacy-aware organizations. We specifically design DP5 to be deployed by a small coalition of independent service providers that wish to offer their users a high degree of privacy. That said, the threshold assumption may be relaxed by using a computational PIR (CPIR) scheme in place of IT-PIR, at a 70–100 times higher computational cost on the (single) server. Alternately, the hybrid IT-PIR + CPIR scheme by Devet and Goldberg [21] provides some additional protection to an IT-PIR scheme, even when all servers are colluding, with negligible additional server computation.

**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.

## 2.3 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, as observed by all the infrastructure servers, 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. Conversely, if an honest friend finds Alice to be offline, this must be due to Alice not having (successfully) completed the registration protocol for that epoch.

**Privacy of the social graph.** 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, and the storage or retrieval of auxiliary data.

**Unlinkability 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 auxiliary data.** Only friends can recover the plaintext of a user's auxiliary presence data. If the adversary submits to the user two candidate plaintexts, and the user chooses one as their auxiliary data for a specific epoch, the adversary cannot efficiently distinguish which of the two was chosen.

**Integrity of auxiliary data.** If an honest friend of Alice recovers a plaintext of auxiliary data it must be the case that Alice ran the 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—which means that using DP5 over an anonymous channel preserves anonymity.

We note that although Alice's friends can never be convinced that Alice is online when she is not, the second component of integrity of presence, namely that Alice's registration is not dropped, is enforced by an auditing mechanism. The integrity of auxiliary data requires either the mechanism described in Appendix A or the use of digital signatures.

# 3 The DP5 Presence Protocol

## 3.1 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 whom 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 with 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) lookup servers 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.

## 3.2 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"—the key $K_{ba}$ is also shared but different from $K_{ab}$). This key can be computed through a Diffie-Hellman key agreement [23], assuming Alice and Bob can each learn discover each other's public key. 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 primitives: three families of keyed pseudo-random functions ($\text{PRF}^{\ell}_{K}(m)$, $\ell \in \{1, 2, 3\}$, implemented using a hash function such as SHA-256 [27]); an authenticated encryption primitive ($\text{AEAD}^{IV}_{K}(h; m)$)[2] (such as AES [18] in GCM mode [35]); and access to secure channels between clients and infrastructure (using TLS [22]).

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 $e(G_1, G_2) \rightarrow G_T$ 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 [28], 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) [8] problem for $G_1$ and $G_2$. An efficiently computable hash function $H_0 : G_T \rightarrow \{0,1\}^{\eta}$ from elements of $G_T$ to $\eta$-bit strings is known by all ($\eta$ is the length of an identifier). Everyone also knows two efficiently computable hash functions $H_1 : \mathcal{T} \rightarrow G_2$ (where $\mathcal{T}$ is the set of valid epoch timestamps) and $H_3 : G_1 \rightarrow \{0,1\}^{\nu}$ (where $\nu$ is the key size of the PRF[3] function).

Finally, all users share some global parameters, such as a maximum number of friends $N_{\text{fmax}}$, the number $N_{\text{pirmax}}$ 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.

**2** $h$ here is the part of the message that is not encrypted but included in the authentication (what the AEAD calls "associated data"—not to be confused with the DP5 auxiliary data); we omit $h$ when it is empty.

## 3.3 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. In DP5, we use *information-theoretic* PIR, in which multiple (non-colluding) PIR lookup servers are employed. We choose IT-PIR for its 70–100 times speed improvement over computational PIR, but see §2.2 for more discussion of this choice. The databases to be searched are dictionaries of ⟨key,value⟩ pairs, where the keys are arbitrary ID strings of some fixed length, and the values are ciphertexts $C$. There will be one such database for each short-term and for each long-term epoch. A DP5 client seeks to retrieve from a particular database the values corresponding to a list of given dictionary keys, *without revealing* that list of keys to the lookup servers. (The client will also typically pad the list of keys to some fixed length in order to hide even the *number* of keys being retrieved.)

We denote by $\text{PIR}\textsc{Lookup}(\tau, \langle ID_1, ID_2, \ldots, ID_k \rangle)$ the interactive protocol performed between the DP5 client and each of the $N_{\text{pirmax}}$ lookup servers. The parameters are $\tau$—an epoch identifier (short term $t_i$ or long term $T_j$) to select the database to query—and a list of dictionary keys to look up. The protocol consists of two round trips with each server: the client sends $\tau$ to each server (in parallel), and receives a response containing metadata for the corresponding database; the client then sends a PIR query to each server (again in parallel), and receives the PIR responses.

At the end of the protocol, the client learns the associated values $\langle C_1, C_2, \ldots, C_k \rangle$ (where some of the $C_i$ may be $\perp$ if the corresponding $ID_i$ was not in the database corresponding to the epoch $\tau$), and the servers learn $\tau$ and $k$ (though, as above, $k$ may be larger than the number of $ID$s the client was actually interested in). Importantly, the servers do not learn the $ID_i$, the $C_i$, or even which $C_i$ are $\perp$. The details of the protocol can be found in Appendix B.

## 3.4 DP5 overview

Sections 3.5 and 3.6 provide the full details of the DP5 registration and query protocols respectively. The message flows between user, registration server and lookup servers are also fully illustrated in Figure 1. Here we start by presenting a high-level overview of interactions, and the rationale behind the DP5 design.

The intuition behind DP5 is as follows. When Alice is online, she will upload an indication of her presence, as well as her auxiliary data (her "presence record"), to a DP5 registration server, encrypted in a way that only her friends can read it. Her friend Bob will then query the server for Alice's (and all of Bob's other friends') presence records. However, even though the server cannot read Alice's encrypted record, if this lookup were done naively, the server would learn that Bob requested the record uploaded by Alice, and would therefore learn that Alice and Bob were friends.

Instead, Bob queries for his friends' presence records using PIR. To accomplish this, time is divided into epochs. At the end of each epoch, the registration server collects all of the presence records uploaded during that epoch, and creates the PIR database for that epoch as described in Appendix B. It then sends a copy of this database to each of the $N_{\text{pirmax}}$ PIR lookup servers. During the next epoch, Bob will use multi-server IT-PIR to look up his friends' records. Note that the separate (non-colluding) PIR lookup servers would be unnecessary if CPIR were used instead of IT-PIR, but we choose to use IT-PIR for computational cost reasons, as described in Section 3.3.

Even so, the computation cost of IT-PIR is somewhat too high. In order to upload her presence record encrypted so that only her friends can read it, the straightforward approach is for Alice to upload, for each of her friends, one presence record encrypted with a symmetric key derived from the key she shares with that friend, so that Alice uploads $N_{\text{fmax}}$ presence records in total during each epoch. This makes the size of the database very large, and since the computational cost of PIR is proportional to the size of the database, it makes the PIR computation expensive. We could ameliorate this by making the epochs longer, but Bob only sees Alice as being online once the next epoch starts. If this epoch length is too long, it will affect the user acceptance of the system.

To address this problem, we add a layer of indirection. We have long-term epochs and short-term epochs. In each long-term epoch $T_{j-1}$, Alice uploads one presence record for each of her friends as above, but her auxiliary data is replaced with a presence key $P_a^j$ ("$a$" for Alice). She uses the *same* presence key in each of the $N_{\text{fmax}}$ records (encrypted individually for each friend), so that each of her friends can learn it, but no one else can. Then in each short-term epoch $t_i$ contained in the next long-term epoch $T_j$, Alice uploads a *single* presence record, encrypted with a key derived from $P_a^j$, known only to her friends.

The databases for the short-term epochs are then much smaller, and using PIR to query them frequently is more reasonable, while the databases for the long-term epochs are larger, but are queried less often: the length of the long-term epoch now governs how quickly a *friend suspension or revocation* will take effect, while the length of the short-term epoch continues to govern how quickly friends are visible as being online.

The final twist is that all of Alice's friends will learn her presence key $P_a^j$ for long-term epoch $T_j$, and so we need some way to prevent one of Alice's friends from uploading a presence record that makes it appear as if Alice is online during some short-term epoch $t_i$ contained in $T_j$, when she is really not. To this end, we make the presence key $P_a^j$ not a symmetric key, but rather a public key, and Alice will use the corresponding private key to produce a signature. The lookup servers (at least a threshold of which are assumed to be honest, recall) will then check that each entry in the short-term database is accompanied by a valid signature. The lookup servers must not learn the public key $P_a^j$, however. In the remainder of this section, we show that by making the dictionary key for the short-term database to be the common result of the two pairings in BLS signature verification [9], the lookup servers can ensure that when Alice's friends look up her presence record in the short-term database (using her public key $P_a^j$), only Alice could have produced the required signature checked by the lookup server, even though the lookup server does not itself learn Alice's public key. In Appendix A, we give an alternate construction that uses more standard digital signatures, but using one-time-use public/private key pairs derived from $P_a^j$ during each short-term epoch. The derived public key is made available to the lookup servers, but it is unlinkable to either $P_a^j$ or to the derived keys from other short-term epochs.

## 3.5 DP5 registration

Alice registers her presence and auxiliary data for each epoch, by updating a number of databases at epoch $t_{i-1}$ and $T_{j-1}$, which 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 may update the *long-term epoch friendship database* for the next long-term epoch $T_j$ with a record for each of the friends to whom she wishes to advertise her presence. Alice only needs to update this long-term database if she wishes to modify the set of friends she advertises presence to, by adding new or removing older friends. Otherwise she may skip the long-term epoch registration (see detailed discussion in Section 3.7).

The long-term epoch 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 *presence key* $P_a^j = g_1^x$, and deletes any older key pairs. Then for each friend 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 $ID_{ab}^j = \text{PRF}_{K_{ab}}^2(T_j)$, and encrypts her public key as $C_{ab}^j = \text{AEAD}_{K_{ab}^j}^0(ID_{ab}^j; P_a^j)$. The resulting entry is $\langle ID_{ab}^j, C_{ab}^j \rangle$.

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 $N_{\text{fmax}}$. Those random entries are generated by Alice performing the encoding process above using a randomly chosen fresh 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. This procedure is illustrated in Figure 1a.

**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 auxiliary data $m_a^i$. Alice first derives $s_a^i = H_1(t_i)^x$, which represents an unforgeable signature that Alice is online. Furthermore, Alice encrypts her auxiliary data as $c_a^i = \text{AEAD}_{K_a^i}^i(m_a^i)$, 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.

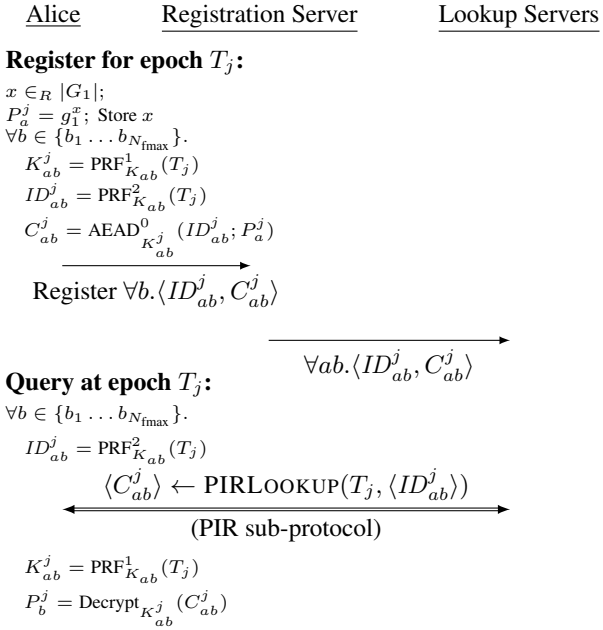
The registration server, upon receiving an entry $(s_a^i, c_a^i)$ from Alice, first derives an identifier $ID_a^i = H_0(e(g_1, s_a^i))$. (Note that this value also equals $H_0(e(P_a^j, H_1(t_i)))$, by the properties of pairings.) Then the server updates two parallel databases: the entry $\langle ID_a^i, c_a^i \rangle$ is added to the short-term epoch user database for $t_i$, and the entry $\langle ID_a^i, s_a^i \rangle$ is added to the short-term epoch signature database. This procedure is illustrated in Figure 1b.

## 3.6 DP5 query

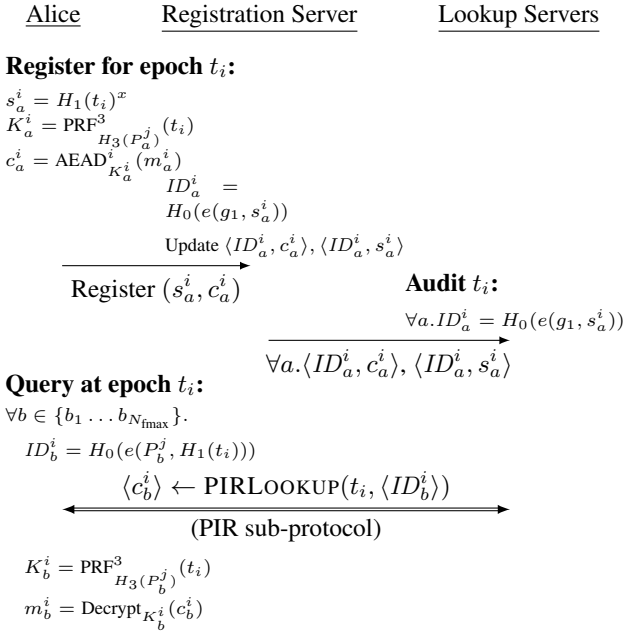
At the beginning of epoch $T_j$ the registration service makes public the full long-term epoch friendship database with all entries received during epoch $T_{j-1}$. Similarly, at the beginning of each short-term epoch $t_i$, the registration server makes available the separate short-term user and signature databases 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 $\langle ID_a^i, c_a^i \rangle$ in the user database must correspond to an entry $\langle ID_a^i, s_a^i \rangle$ in the signature database, such that $ID_a^i = H_0(e(g_1, s_a^i))$. If the audit succeeds the PIR server proceeds to answer requests for entries in the databases.

Once per long-term epoch $T_j$, during $T_j$ or at a later long-term epoch, Bob queries the long-term friendship database for




**(a)** DP5 protocols to register and query presence for long-term epoch $T_j$




**(b)** DP5 protocols to register and query presence for short-term epoch $t_i$

**Fig. 1.** DP5 protocols for long and short term epochs

entries corresponding to each of his friends. First, he reconstructs for each of his friends a shared identifier; e.g., for Alice he computes the identifier $ID_{ab}^j = \text{PRF}^2_{K_{ab}}(T_j)$. He then pads this list of friend identifiers with a number of random identifiers, up to a maximum number of friends $N_{\text{fmax}}$. Finally, using PIRLOOKUP, 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 $\langle ID_{fb}^j, C_{fb}^j \rangle$, 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 $K_{ab}^j = \text{PRF}^1_{K_{ab}}(T_j)$ to yield Alice's current public presence key $P_a^j$. This procedure is diagrammed in Figure 1a.

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 presence key $P_a^j$ available for her. To this end he computes $ID_a^i = H_0(e(P_a^j, H_1(t_i)))$ for Alice, and similarly an ID for each of his other friends. He then uses PIRLOOKUP on the list of those identifiers, padded with random strings to a maximal length of $N_{\text{fmax}}$. Upon completion of the retrieval protocol, Bob decrypts each returned ciphertext entry with a symmetric key derived as $K_a^i = \text{PRF}^3_{H_3(P_a^j)}(t_i)$. If the decryption succeeds the status of the friend is set as "online", and the auxiliary data $m_a^i$ is returned. Otherwise the friend's status is set as "offline", and no auxiliary data is returned. This procedure is illustrated in Figure 1b.

## 3.7 Protocol details and options

The DP5 protocols can be parametrized to achieve different trade-offs and optional client-side steps may be taken to achieve additional properties

**Epoch lengths.** The presence mechanism is divided into long-term epochs and short-term epochs. The length of the short-term epoch governs how quickly Bob will notice that Alice has come online; presence indications only change at the beginning of each new short-term epoch. The length of the long-term epoch governs how quickly Alice can suspend or revoke Bob as a friend; such a de-friending will only take effect at the beginning of the next long-term epoch.

In addition, the two epoch lengths have differing performance characteristics. In the long-term epoch the registration phase has a space complexity of $\Theta(N_{\text{fmax}})$, where $N_{\text{fmax}}$ is the maximal number of friends. However, in the short-term epoch the registration space complexity is merely $\Theta(1)$, making short-term updates extremely efficient. Both mechanisms require $N_{\text{fmax}}$ 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 $N \cdot N_{\text{fmax}}$ entries, where $N$ is the number of distinct registered clients, whereas the short-term database only contains $N$ 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.

**Adding friends, suspending presence, and revocation.** Adding a friend so that they receive DP5 updates is very efficient: Alice and Bob merely have to establish a shared secret key, and exchange their public presence keys $P_A^j$ and $P_B^j$. How this exchange is performed is outside the scope of DP5 but the resulting social link should be unobservable by an adversary. Using the ephemeral keys, new friends can query the short-term epoch databases and retrieve presence information immediately at the next short-term epoch. Starting with the next long-term epoch they may query and update DP5 normally using their shared secret key.

Removing friends and obscuring presence takes longer. As mentioned above, the cost of a 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 change her presence key before the next time she registers with the long-term friendship database, and not use her shared key with Bob in the next long-term registration protocol. Any updates to the long-term friendship database can only take place once the long-term epoch changes, and thus the immediacy of revocation is limited by the long-term epoch length.

**Filling in long-term updates.** Alice's friends may 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.

In a practical DP5 deployment lookup server may wish to limit the number of older long-term epoch databases they retain. In such settings, all users should be required to execute the long-term epoch registration process often enough to ensure that their keys are still present in a retained past epoch. Given the relatively small size of the long-term database (see Table 2) as compared to the cost of bulk storage, we do not explore this possibility further.

**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 entries are not included in the databases, but not that the registration server does not drop valid entries. Each client may perform some limited checks to reduce the likelihood of a malicious registration server not serving the full database. A client may 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 further extended so that the registration servers provide a signed receipt upon each registration. This would allow non-inclusion of records to be verified by third parties, and the registration server replaced.

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 query for fewer friends per epoch than the maximum allowed.

# 4 Security argument

We present proofs or proof sketches for key properties of DP5 in Appendix C. Table 1 summarizes the assumptions that are made about the infrastructure with respect to each of these properties. We briefly discuss the remaining properties here.

– **Indistinguishability of offline status, suspension, and revocation.** To revoke or suspend Bob's access, Alice

**Table 1.** Assumptions necessary for enforcing security properties. The **Auditable?** column describes whether auditing can detect violation of this property, should the trust assumptions be violated. $t$ is the system threshold parameter for the privacy of IT-PIR.

| Property | Example | Assumptions | Auditable? |
|---|---|---|---|
| **Privacy of presence** | **Cannot tell if Alice online in epoch $t_j$** | **Alice's (unrevoked) buddies are uncompromised** | **no** |
| **Integrity of presence** | **Cannot make Alice appear online in an epoch when she does not register** | **none** | **no** |
| | **Cannot make Alice appear offline when she *does* register** | **honest-but-curious registration server and $t+2$ PIR servers** | **yes** |
| **Privacy of social graph** | **Cannot tell if Alice has Bob as a buddy** | **no collusion among more than $t$ PIR servers** | **no** |
| **Unlinkability between epochs** | **Cannot link two registrations across epochs** | **Alice's (unrevoked) buddies are uncompromised** | **no** |
| **Privacy and integrity of auxiliary data** | **Cannot learn Alice's auxiliary data** | **Alice's (unrevoked) buddies are uncompromised** | **no** |
| | **Cannot modify Alice's auxiliary data** | **Registration server does not collude with Alice's buddies** | **yes** |
| **Indistinguishability of offline status, suspension, and revocation** | **Bob cannot tell if Alice is offline or has revoked his access** | **Alice's other (unrevoked) buddies do not collude with Bob** | **no** |

chooses a new public presence key $P_a^j$ and does not share it to Bob. To maintain indistinguishability, Alice may generate a separate key $\widehat{P}_a^j$ 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, reveals no information about the identity of the querier.

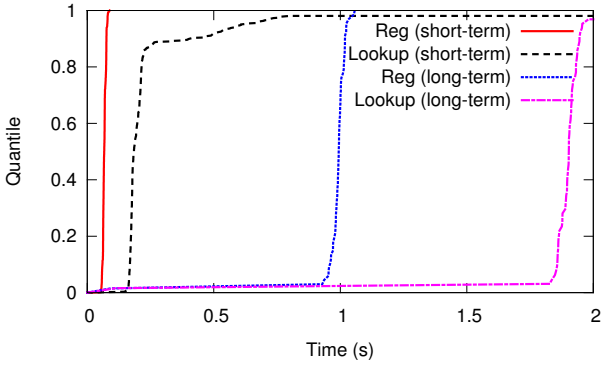



**Fig. 2.** Overall user-facing latency for the registration and lookup operations, excluding four network RTTs for registration and five for lookup.

# 5 Evaluation

## 5.1 Implementation

The implementation of the DP5 protocol consists of publicly available open-source 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. 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). All PRFs are based on hashing the secret key and other inputs using SHA-256 [27] and trun-

**Table 2.** Data sizes for $N = 1000$ users, $N_{fmax} = 100$. Registration and lookup costs are per user.

| | Long-term | | Short-term | | Scaling |
|---|---|---|---|---|---|
| | Req | Resp | Req | Resp | (in # of users) |
| DB size | 13 MiB | | 84 KiB | | $\Theta(N)$ |
| Registration | 9 004 B | 5 B | 164 B | 5 B | $\Theta(1)$ |
| Lookup | 300 KiB | 500 KiB | 200 B | 400 B | $\Theta(N^{1/2})$ |

cating the output hash to 16 bytes. The group $G$ is defined as Curve25519 [5] using Adam Langley's implementation.[3] Pairing-friendly curves are provided by the RELIC library [1], and the Optimal Ate pairing over a 256-bit Barreto-Naehrig curve defines groups $G_1$, $G_2$ and $G_T$. We use the Percy++ library [31] 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. Our code is available under an open-source license.[4]

## 5.2 Performance

To evaluate the performance of DP5, we ran 1000 simultaneous clients accessing the DP5 infrastructure. The clients were running on an 80-core Xeon 2.4 GHz server with 2 TiB of RAM. (Note that only a small fraction of the RAM was utilized during the experiments.) For each of the short-term and long-term protocols, we used one server for registration and three servers supporting PIR. Each server was running on a 16-core Xeon 2.0 GHz machine with 256 GiB of RAM. The machines were interconnected using 1 Gbps Ethernet.

Figure 2 summarizes the user-facing latency of the operations over a 10-hour execution, with the short-term and long-term epochs set to 1 minute and 10 minutes, respectively. (In practice, we expect long-term epochs to be much longer, around a day; 10 minutes was chosen to stress-test the system.) In order to measure a worst-case scenario, we had all clients perform their registration and lookup operations simultaneously, thus putting maximum load on the servers. For the short-term epoch, we expect this to represent real-world behavior, as all users will want to look up their friends' presence information right at the start of each epoch. For the long-term epoch, however, clients will come online during different parts of an epoch, and thus may experience lower delays. Monitoring the behavior of the servers, the PIR servers for the long-term database had high CPU utilization for the approximately 65 seconds following an epoch change; other servers were minimally utilized and thus appear to be able to support significantly larger numbers of clients. Note that the figure excludes network latency. Given the RTT between the client and the (registration or lookup) server, one should add three RTTs for the TCP/TLS handshake, and one more RTT for the registration protocol, or two more RTTs for the PIRLOOKUP protocol.

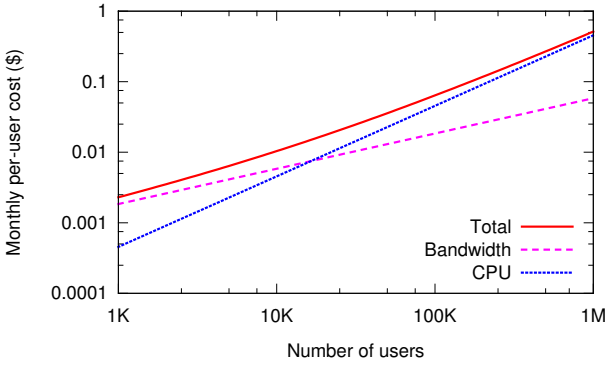



**Fig. 3.** Per-user cost for the bandwidth and CPU associated with running a long-term PIR server with a 24-hour epoch.

Table 2 lists the sizes of the requests and responses in our protocol, excluding the overhead from the HTTP and TLS protocols. The size of the long-term epoch database is about 13 KiB per user, while the size of the short-term epoch database is about 80 bytes per user. The bandwidth costs of synchronizing the databases are therefore much smaller than those of serving the lookup requests, particularly as the number of users grows: the *per-user* lookup size grows as the square root of the number of users, while the *per-user* database synchronization cost is fixed, and independent of the number of users.

We used an $N_{fmax}$ of 100 for our experiments. Note that buddies in DP5 represent users whose presence you want to actively follow, and thus is going to be smaller than the number

3 https://github.com/agl/curve25519-donna

4 git://git-crysp.uwaterloo.ca/dp5

of contacts in a typical social network such as Facebook. We do assign 100 buddies for each user, but we note that the actual number of buddies per user or the friendship topology have no impact on the performance of DP5 as requests are always padded to $N_{fmax}$.

Increasing $N_{fmax}$ system-wide would increase the size of the long-term database by a linear factor, but will not affect the size of the short-term database. Correspondingly, the long-term lookup communication would grow as $\Theta(N_{fmax}^{1.5})$ while the short-term communication as $\Theta(N_{fmax})$. As an alternative, users with unusually large numbers of friends could execute the registration and lookup protocols several times each epoch; such distinguished behavior may, however, be subject to traffic analysis and allow an adversary to isolate heavier users from the rest of the user population.

## 5.3 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 the *square root* of 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 component of DP5. In our experiments it used about 15 core-minutes and 1 GB of bandwidth per epoch. Figure 3 plots an estimate of the cost of running such a PIR server, using $0.84 as the cost of one hour on a 16-core machine and $0.12 as the cost of 1 GB of data transfer,[5] and a 24-hour epoch. User populations in the thousands can easily be supported with volunteer resources. We observe that a subscription service can support as many as 1 million users at a per-user cost of about $0.50/month. As mentioned in §2.2, our target deployment for DP5 is a coalition of privacy-sensitive service providers, and not something on the scale of Google or Facebook, so our above deployment costs are reasonable.

**5** These costs are Amazon's EC2 prices as of November 2014 (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.

# 6 Discussion

## 6.1 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.

## 6.2 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 the key she shared with Bob ($K_{ab}$) from the long-term update mechanism, and refreshes her ephemeral epoch key. This results in Bob not being able to retrieve the next fresh ephemeral epoch key for Alice, and so he cannot query her short-term presence or auxiliary 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 presence key. During any short-term epoch Alice can register with all, or any subset of the presence 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.

## 6.3 Forward secrecy and hardware stores

Various parts of the DP5 protocol have been designed, or can be easily altered, to provide stronger guarantees against endpoint 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 Heartbleed attacks against OpenSSL.

A natural way to derive the shared key $K_{ab}$, used for long-term registration, is by using a Diffie-Hellman key exchange. Once this key is derived, there is no need to store the public or private Diffie-Hellman 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, the long-term epoch keys $K_{ab}^{j}$ 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 $K_{ab}^{j}$. As a result, if a client is compromised, only the keys relating to the current long-term epoch are accessible.[6] 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, it 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 $K_{ab}^{j} = \mathrm{PRF}^{1}_{K_{ab}^{j-1}}(T_j)$. 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: if a user is compromised, the set 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.

## 6.4 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. [11] 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 [20] that ensures that a malicious server would be detected. Preventing DoS requires at least $t + 2$ honest servers, where $t$ is the maximum number

**6** The hardware security module would need to have a secure source of time, or maintain state to prevent rolling back to previous epochs.

of servers that the threat model allows to collude to determine the query.

## 6.5 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 provides for a public presence key that is generated and communicated between friends at each long-term epoch, and then subsequently used in the short-term epochs. While this provides forward secrecy, it also creates a sequential dependency between the long-term protocols and the short-term protocols. As a result, all online 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 lies 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.

# 7 Related work

A possible design for private presence consists of simply running a conventional presence system, or even a full chat server, with clients accessing it through an anonymous channel, such as Tor [25]. It is noteworthy that all such anonymous channels rely on third parties for their security properties, as does the IT-PIR scheme DP5 uses. This mechanism allows users to hide their identities behind pseudonyms. However, the relations *between pseudonyms* are revealed to the presence service. The revealed relationship graph between pseudonyms is isomorphic to the one between users, and thus users may be re-identified using some side information and standard techniques [36].

Another possible design uses Tor *hidden services*, which provide anonymity to both clients and servers. In this case, Bob could run a presence service that Alice and his other friends could query whenever they wish to query whether he is online. Note that Alice must use a separate anonymous channel (called a *circuit* in Tor) to connect to each of her friends' presence services, creating a far greater number of circuits than the one assumed by current anonymity systems, such as Tor, that re-use circuits for some time. Besides the performance problems, anonymous communication channels, be it mixes or onion routers, are susceptible to long-term intersection attacks; in Tor, realistic adversaries are likely to learn a relationship after a few months of repeated connection patterns [32].[7]

DP5 does not suffer from any traffic analysis attacks, and leaks no information about the relationship graph. The use of an IT-PIR scheme allows operators to increase the security of a DP5 deployment by adding additional servers; in contrast, an onion routing system does not provide such a security parameter: increasing the length of circuits does not improve security due to the end-to-end correlation attack [39]. Higher-latency mix-based anonymity systems also rely on a threshold security assumption of at least one honest mix, similar to the IT-PIR threat model, but such systems are slower and also subject to intersection attacks.

Laurie's Apres [33] 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 [19], which proposes a simplified presence mechanism 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, revealed by Apres. 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

**7** In fact, a recent attack on Tor specifically focused on learning over time which users were interested in which hidden services [24].

network graphs if their topology remains the same [36]. 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.

Dissent [43] and Riposte [17] are anonymous broadcast messaging systems that are also immune from traffic analysis; Dissent is based on DC-nets [13], while Riposte uses a PIR-like mechanism to allow clients to write to a private location in a per-epoch database. Riposte in particular can scale to millions of users under the assumption that only a small fraction of users write to the database, and that epochs are several hours in length. The broadcast nature of these systems imposes significantly higher communication costs on the users as compared with DP5.

The Tor Project is in the process of redesigning the Hidden Services mechanism [25], 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 [2] and DNSCurve [6] 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 hand, DNSCurve does support dynamic binding of names to addresses, through stronger channel security. It protects presence against network adversaries and limited traffic analysis protection due to potential local caching, but is not immune to traffic analysis as DP5 is.

The GNU Name System [41] and a proposal by Tan and Sherr [40] use a Distributed Hash Table (DHT) maintained by users to mirror all peers' name records and mappings rendezvous points. DHTs, however, do not provide strong privacy protections; extensions that add anonymity to DHTs [3, 42] generally rely on strong assumptions such as the absence of Sybil attacks [26] and provide only loose probabilistic guarantees for a single query, and unknown protection against long-term traffic analysis.

Proposals for privacy-friendly social networking protocols, such as Diaspora,[8] are related to the DP5 effort but provide privacy within a very different threat model. Users of such systems share their information with small local providers that federate to provide a global social network. Thus single points of trust exist that an adversary could corrupt or compel to reveal some people's social graph. Even cryptographically sophisticated designs, such as Persona [4], only protect the social network content, but not the social graph or presence against traffic analysis attacks. Building decentralized designs immune to long-term traffic analysis remains an open research problem.

# 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, we have carefully designed 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 identities.

Overall, the protocols are scalable to small deployments of a few thousands, to tens of thousands of concurrent clients, a size suitable for small NGOs or cooperative service providers that care about users' privacy. The key scalability bottleneck 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 and Roger Dingledine for key discussions relating to the design of DP5, and Schloss Dagstuhl seminars for hosting those important design discussions. We also thank Harry Halpin and Elijah Sparrow for their feedback on presence requirements, Microsoft Research Cambridge for hosting two of the authors, Daniel J. Bernstein and Tanja Lange for suggesting that a pairing-free variant of the protocol should be possible, and the anonymous reviewers for their helpful feedback. This material is based upon work supported by the National Science Foundation under Grant No. 0953655, and by NSERC, ORF, The Tor Project

**8** https://joindiaspora.com/

and EPSRC Grant EP/M013286/1. This work benefited from the use of the CrySP RIPPLE Facility at the University of Waterloo.

# References


[1] Diego F. Aranha and Conrado Porto Lopes Gouvêa. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic, 2015.

[2] 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.

[3] Michael Backes, Ian Goldberg, Aniket Kate, and Tomas Toft. Adding query privacy to robust DHTs. In Heung Youl Youm and Yoojae Won, editors, *7th ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 30–31. ACM, 2012.

[4] Randolph Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. In Pablo Rodriguez, Ernst W. Biersack, Konstantina Papagiannaki, and Luigi Rizzo, editors, *ACM SIGCOMM Conference on Data Communication*, pages 135–146. ACM, 2009.

[5] 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.

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

[7] 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.

[8] 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*, number 2656 in Lecture Notes in Computer Science, pages 416–432. Springer, January 2003.

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

[10] Philippe Boucher, Adam Shostack, and Ian Goldberg. Freedom systems 2.0 architecture. White paper, Zero Knowledge Systems, Inc., December 2000.

[11] 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.

[12] 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.

[13] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, 1988.

[14] 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.

[15] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on the Foundations of Computer Science (FOCS)*, pages 41–50, Oct 1995.

[16] David Cole. We kill people based on metadata. New York Review of Books, May 10 2014.

[17] Henry Corrigan-Gibbs, Dan Boneh, and David Mazieres. Riposte: An anonymous messaging system handling millions of users. In *36th IEEE Symposium on Security and Privacy*, May 2015.

[18] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES—The Advanced Encryption Standard*. Springer, 2002.

[19] 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.

[20] Caset Devet, Nadia Heninger, and Ian Goldberg. Optimally robust private information retrieval. In *21st USENIX Security Symposium*, Aug 2012.

[21] Casey Devet and Ian Goldberg. The best of both worlds: Combining information-theoretic and computational pir for communication efficiency. In *14th Privacy Enhancing Technologies Symposium*, pages 63–82, July 2014.

[22] T. Dierks and E. Rescorla. The transport layer security (TLS) protocol version 1.2. RFC 5246 (Proposed Standard), August 2008.

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

[24] Roger Dingledine. Tor security advisory: “relay early” traffic confirmation attack. https://blog.torproject.org/blog/tor-security-advisory-relay-early-traffic-confirmation-attack, July 2014.

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

[26] John R. Douceur. The Sybil attack. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer, 2002.

[27] Donald Eastlake and Paul Jones. US Secure Hash Algorithm 1 (SHA1). RFC 3174, September 2001.

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

[29] James Glanz, Jeff Larson, and Andrew W. Lehren. Spy agencies tap data streaming from phone apps, January 27 2014.

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

[31] Ian Goldberg, Casey Devet, Wouter Lueks, Ann Yang, Paul Hendry, and Ryan Henry. Percy++ project on SourceForge, October 2014. http://percy.sourceforge.net/.

[32] Aaron Johnson, Chris Wacek, Rob Jansen, Micah Sherr, and Paul Syverson. Users get routed: Traffic correlation on Tor by realistic adversaries. In *20th ACM Conference on Computer and Communications Security (CCS)*, November 2013.

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

[34] Wouter Lueks and Ian Goldberg. Sublinear scaling for multi-client private information retrieval. In *19th International Conference on Financial Cryptography and Data Security*, January 2015.

[35] David A McGrew and John Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In *Progress in Cryptology-INDOCRYPT*, pages 343–355. Springer, 2005.

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

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

[38] 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.

[39] Paul F. Syverson, Gene Tsudik, Michael G. Reed, and Carl E. Landwehr. Towards an analysis of onion routing security. In Hannes Federrath, editor, *Workshop on Design Issues in Anonymity and Unobservability*, volume 2009 of *Lecture Notes in Computer Science*, pages 96–114. Springer, 2000.

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

[41] 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.

[42] Qiyan Wang and Nikita Borisov. Octopus: A secure and anonymous DHT lookup. In Xavier Defago and Wang-Chien Lee, editors, *32nd IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 325–334, June 2012.

[43] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *10th USENIX Symposium on Operating Systems Design and Implementation*, pages 179–182. USENIX, 2012.

# 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 and Tanja Lange 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 [7], as described next.

As part of the long-term registration, Alice stores in her status auxiliary data a public presence 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 presence key $P_a^j$; furthermore, those public keys are unlikable across short-term epochs. 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 auxiliary data.

To perform the short-term registration protocol, Alice stores in the database the tuple $\langle P_a^{ij}, \mathrm{AEAD}_{K_a^i}^i(m_a^i) \rangle$. Furthermore, Alice sends to the registration server a signature of the tuple under the 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 auxiliary 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 auxiliary 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 Details of the PIR subprotocol

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. When using IT-PIR, the client information-theoretically splits her query across the set of $N_{\text{pirmax}}$ 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. [15]. 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 [30] 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. [20] 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 [31], which we use in our implementation of DP5.

Our situation is not quite as simple as the above protocols provide for, however. Our databases are dictionaries of ⟨key,value⟩ pairs—the keys are arbitrary ID strings of some fixed length, and the values are fixed-length ciphertexts $C$—and our goal is to retrieve the values corresponding to a list of given dictionary keys, 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 [14, §5.3]. This extension works as follows: Let $s$ be the (fixed) size (in *bytes*, as we use $w = 8$) of one ⟨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 $\Pi^{(r)}$ with codomain $\{1, \ldots, r\}$, using each to hash all $n$ keys in the database, and find the largest number $m$ of records hashed into any one bucket. We then keep the PRF key $\kappa$ that yielded the smallest such largest bucket. The database we will query via PIR then consists of $r$ blocks, each of size $m \cdot s$ bytes, where block $j$ is the concatenation of all of the $\langle ID_i, C_i \rangle$ pairs for which $\Pi^{(r)}_\kappa(ID_i) = j$ (padded to $m \cdot s$ bytes if there are fewer than $m$ of them), sorted by $ID_i$.

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.

The complete PIRLOOKUP protocol is then as follows:

- Client input: Epoch identifier $\tau$, list of dictionary keys $\langle ID_1, \ldots, ID_k \rangle$
- Each server input: One dictionary of $\langle ID, C \rangle$ pairs for each long-term and short-term epoch (each server has a copy of the same dictionary for each epoch)
- Client to each server: $\tau$
- Each server to client: $r$, $m$, $\kappa$ corresponding to database $\tau$
- The responses from each server should be identical, so the client takes the majority response and stops talking to any server that deviated.
- For each $1 \leq i \leq k$, the client uses a robust IT-PIR protocol to query the servers for block $\Pi^{(r)}(ID_i)$, of size $m \cdot s$ bytes (sent as a single message from the client to each server).
- Each server computes its response according to the IT-PIR protocol being used. Some PIR protocols support batch queries [34], which reduces the computation cost of replying to the $k$ simultaneous queries per client, and indeed to multiple simultaneous clients.
- The client receives the servers' responses, and combine them to recover the requested blocks. For each $1 \leq i \leq k$, binary search block $\Pi^{(r)}(ID_i)$ for the presence of a pair whose key is $ID_i$. If it is present, set $C_i$ to the corresponding value. Otherwise, set $C_i$ to $\perp$.
- Return the list $\langle C_1, \ldots, C_k \rangle$ to the client.

# C Security Definitions and Proofs or Arguments

We present formal definitions of the security properties of DP5. The system can be modeled by the following algorithms:

- $\mathsf{GenParams}(1^\lambda) \rightarrow params$: Generate system parameters based on a security level $\lambda$
- $\mathsf{GenLongTermKey}(params, A) \rightarrow (K_A, k_A)$: Generate a public/private key pair for $A$'s long-term identity

| $\mathcal{G}_0$ | $\mathcal{G}_1$ | $\mathcal{G}_2$ | $\mathcal{G}_3$ |
|---|---|---|---|
| $c \in_R \{0,1\}$ | $c \in_R \{0,1\}$ | $c \in_R \{0,1\}$ | $c \in_R \{0,1\}$ |
| $K_{a_c b_c} = pk_{b_c}^{sk_{a_c}}$ | $\boxed{\mathbf{z} \in_{\mathbf{R}} \lvert\langle \mathbf{g} \rangle\rvert}$ | $z \in_R \lvert\langle g \rangle\rvert$ | $R_1 \in_R \{0,1\}^\nu$ |
| $K^j_{a_c b_c} = \text{PRF}^0_{K_{a_c b_c}}(T_j)$ | $K^j_{a_c b_c} = \text{PRF}^0_{\boxed{\mathbf{g^z}}}(T_j)$ | $\boxed{\mathbf{R_1} \in_{\mathbf{R}} \{\mathbf{0},\mathbf{1}\}^\nu}$ | $\boxed{\mathbf{R_2} \in_{\mathbf{R}} \{\mathbf{0},\mathbf{1}\}^\eta}$ |
| $\text{ID}^j_{a_c b_c} = \text{PRF}^1_{K_{a_c b_c}}(T_j)$ | $\text{ID}^j_{a_c b_c} = \text{PRF}^1_{\boxed{\mathbf{g^z}}}(T_j)$ | $\text{ID}^j_{a_c b_c} = \text{PRF}^1_{g^z}(T_j)$ | $C^j_{a_c b_c} = \text{AEAD}^0_{R_1}(\boxed{\mathbf{R_2}}; d_i^c)$ |
| $C^j_{a_c b_c} = \text{AEAD}^0_{K^j_{a_c b_c}}(\text{ID}^j_{a_c b_c}; d_i^c)$ | $C^j_{a_c b_c} = \text{AEAD}^0_{K^j_{a_c b_c}}(\text{ID}^j_{a_c b_c}; d_i^c)$ | $C^j_{a_c b_c} = \text{AEAD}^0_{\boxed{\mathbf{R_1}}}(\text{ID}^j_{a_c b_c}; d_i^c)$ | |

**Fig. 4.** Computing the challenge message for contact $b_c$ in successive games for the long-term protocol. $\eta$ here is the length of the public identifier.

- $\mathsf{GenShortTermKey}(params, A) \rightarrow (K'_A, k'_A)$: Generate a short-term public/private key pair for $A$.
- $\mathsf{RegisterLongTerm}(k_a, T_j, K'_A, \{K_b\}_{b \in buddies}) \rightarrow regtoken$: Register in the long-term database for the epoch $T_j$, making the short-term public key $K'_A$ available to all buddies. The result is the registration token that is sent to the registration server.
- $\mathsf{RegisterShortTerm}(k'_a, t_j, D) \rightarrow regtoken$: Register in the short-term database for epoch $t_j$, using the short-term keys and auxiliary data $D$.
- $\mathsf{LongTermDB}(\{regtoken_i\}) \rightarrow DB$. Generate a long-term registration database for an epoch, taking a complete set of registration transcripts for an epoch as an input and producing a database that will be distributed to PIR servers.
- $\mathsf{ShortTermDB}(\{regtoken_i\}) \rightarrow DB$. Generate a short-term registration database.
- $\mathsf{LongTermQuery}(k_a, T_j, \{K_b\}_{b \in buddies}) \rightarrow \{query_i\}$. Generate a set of lookup queries to look up a set of buddies in the long-term database.
- $\mathsf{ShortTermQuery}(t_j, \{K'_b\}_{b \in buddies}) \rightarrow \{query_i\}$. Generate a set of lookup queries to look up a set of buddies in the short-term database.
- $\mathsf{PIRResponse}(DB, query) \rightarrow response$. Process a PIR query and produce a response using a particular database.
- $\mathsf{LongTermResult}(\{response_i\}) \rightarrow \{K'_b | \perp\}_{b \in buddies}$: Process PIR responses to obtain the result of a long-term query, returning each buddy's short-term key or $\perp$ if that buddy's registration is missing.
- $\mathsf{ShortTermResult}(\{response_i\}) \rightarrow \{D_b | \perp\}_{b \in buddies}$: Process PIR responses to obtain the result of a short-term query, returning the auxiliary data for online buddies and $\perp$ for offline ones.

We next define a series of games that will formalize the security properties specified in Section 2.3. In each game, we assume that the challenger generates long-term public and private keys for a large number of identities in a set $H$, representing honest users and supplies the adversary with the public keys for each of them. The challenger also generates short-term public keys and private for users in $H$ for each relevant epoch $T_j$ but does *not* supply those to the adversary.

**Game 1** (Long-term Registration Unlinkability)**.** *1. The adversary supplies the challenger with:*
   - *Two identities, $A_0, A_1 \in H$.*
   - *Two sets of friends $B_0, B_1 \subset H$*
   - *Two sets of short-term public keys, $K'_{A_0}$, $K'_{A_1}$*
   - *An epoch $T_j$*
2. *The challenger flips a coin to select a bit $c \in_R \{0,1\}$.*
3. *The challenger creates a registration token by running $\mathsf{RegisterLongTerm}(k_{A_c}, T_j, K'_{A_c}, B_c)$ and returns it to the challenger.*
4. *The adversary may then query for registration messages generated by any user $u \in H$ in an arbitrary epoch, with arbitrary lists of contacts taken for $H \cup C$ and arbitrary public/private keypairs, with the restriction that users $A_0$ and $A_1$ cannot be asked to register again during the epoch $T_j$.*
5. *The adversary outputs its guess for the bit $c$.*

We note that in addition to the unlinkability properties of long-term registration, this property also guarantees the privacy of (long-term) presence, since an adversary who could tell whether, say, $A_0$ is present in the long-term database would be able to win this game.

*Proof.* We will first consider the case where $|B_i| = 1$; i.e., the challenge registration includes a single contact, $b_i$. 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 $K_{a_c b_c}$ with $g^z$, for a uniformly chosen $z$, instead of the key com-

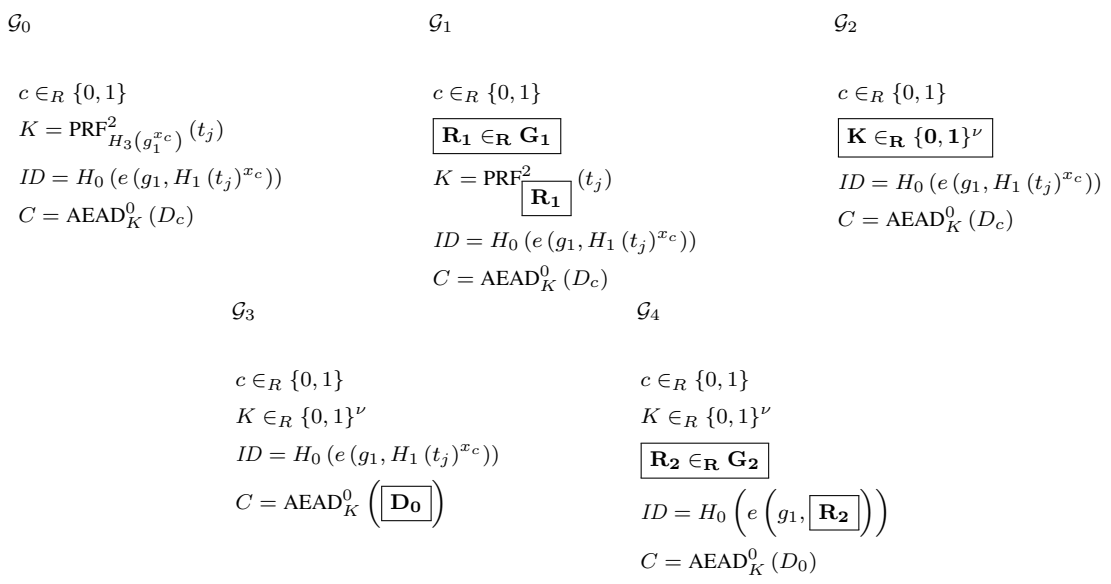



**Fig. 5.** Computing the challenge registration message $\langle ID, C\rangle$ in successive games for the short-term protocol.

puted by Diffie-Hellman. Note that any adversary $\mathcal{A}$ that can distinguish between $\mathcal{G}_0$ and $\mathcal{G}_1$ with advantage $\epsilon$ 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 $b_c$ 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 $K^j_{a_c b_c}$ with $R_1$ chosen uniformly at random. Any adversary who can distinguish $\mathcal{G}_1$ from $\mathcal{G}_2$ with advantage $\epsilon$ can be turned into an adversary who distinguishes $\text{PRF}^0$ 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 $\text{ID}^j_{a_c b_c}$ 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 $\text{PRF}^1$ from random.
- In $\mathcal{G}_3$, the registration message is $\langle R_2, \text{AEAD}^0_{R_1}(R_2; d_c)\rangle$. Any adversary who has advantage $\epsilon$ 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 some $b \in B_c$. 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 keys by, in each game, having the challenger guess which users will be chosen for $a_0/a_1$ and $B$ and aborting if the guess was wrong, at the cost of the reduction no longer being tight.

**Game 2** (Short-term Unlinkability). *1. The adversary supplies the challenger with:*
- *Two usernames, $A_0, A_1 \in H$.*
- *Two pieces of auxiliary data, $D_0, D_1$*
- *An epoch $t_j$*

2. *The challenger flips a coin to select a bit $c \in_R \{0,1\}$.*
3. *The challenger produces a registration message* $\mathsf{RegisterShortTerm}(k'_{A_c}, t_j, D_c)$ *as shown in Figure 5, game $\mathcal{G}_0$.*
4. *The adversary may perform a polynomial number of queries to perform a short-term registration of users in $H$, supplying the epoch and data, with the constraint that neither $A_0$ nor $A_1$ can be asked to register in epoch $t_j$ again.*
5. *The adversary outputs its guess for the bit $c$.*

Note that, similar to the long-term unlinkability game, this game also implies the privacy of (short-term) presence. Ad-

ditionally, since the adversary may choose the auxiliary data, this game implies the privacy of the auxiliary data.

*Proof.* 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 [8] 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 $h_1^\alpha$. 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 $\epsilon$ 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 $1/2$. □

**Game 3** (Integrity of Presence)**.** *1. The adversary submits long-term registration requests for users in $H$, specifying an epoch $T_j$ and buddy lists in $H \cup C$; the challenger returns the corresponding registration tokens.*

2. *The adversary supplies the challenger with two users $A, B \in H$ and an epoch $T_j$, with the constraint that it never sent a registration request for $A$ in epoch $T_j$. The challenger runs* $\mathsf{LongTermQuery}(k_A, T_j\{K_B\})$ *and returns the resulting queries to the adversary.*
3. *The adversary produces a set of PIR responses $\{response_i\}$. The adversary wins if* $\mathsf{LongTermResult}(\{response_i\}) \neq \{\perp\}$.

Note that the adversary here performs all of the computation of all the registration and PIR servers and thus this model captures a fully malicious infrastructure. The adversary has a negligible advantage in this game since the registration token contains an identifier $ID$, produced using a Diffie-Hellman key exchange using the private keys $k_A$ and $k_B$, neither of which are available to the adversary. (Note that $k_A$ and $k_B$ may be used in other registration queries by the challenger, but the corresponding shared secret only serves as input to key-derivation PRFs, and thus the adversary learns no information about the private keys.)

We can define an analogous game for short-term presence. A presence registration requires $H(t_i)^x$, which is a BLS signature [9] on the epoch number $t_i$. 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 [8] under a uniform generator assumption [12]. This signature is not, however, included in the PIR database, so the game would need to require that $A$'s buddies are not compromised. (Alternatively, they may be compromised but the registration server must be honest-but-curious.) A dishonest registration server colluding with a compromised buddy will be caught by the PIR servers, who check the signatures separately.

**Game 4** (Registration Buddy Privacy)**.** *1. The adversary submits a long-term registration challenge: $A \in H, B_0, B_1 \subset H \cup C$ and an epoch $T_j$, with the constraints that $B_0 \cap C = B_1 \cap C$.*

2. *The challenger flips a coin to obtain the challenge bit $c$ and computes the registration token* $\mathsf{LongTermRegister}(K_A, \{K_b\}_{b \in B_c})$*, returning it to the adversary.*
3. *The adversary may further issue queries to obtain registration tokens for $A' \in H, B' \subset H \cup C, T_{j'}$ with the restriction that $A' \neq A$ or $T_{j'} \neq T_j$.*
4. *The adversary outputs a guess for the challenge bit $c$*

This game is similar to the registration unlinkability game, except that here, Alice's identity is kept fixed and the adversary is allowed to compromise her buddies, yet the identity of the uncompromised buddies remains hidden. The proof is similar to that of the unlinkability game.

**Game 5** (Lookup Buddy Privacy). *1. The adversary submits a short-term query challenge: $A \in H, B_0, B_1 \subset H \cup C$ and an epoch $T_j$, with the constraints that $B_0 \cap C = B_1 \cap C$.*

2. *The challenger flips a coin to obtain the challenge bit $c$ and computes the query $\mathsf{LongTermQuery}(k_A, T_j, \{K_b\}_{b \in B_c})$. It returns a subset of $t$ queries to the adversary.*
3. *The adversary may further issue queries to obtain short- and long-term registrations for $A' \in H, B' \subset H \cup C$ for any epoch without restriction. It may also obtain long- and short-term queries for arbitrary sets of identities and buddies, obtaining the full set of queries (i.e., not just $t$).*
4. *The adversary outputs a guess for the challenge bit $c$*

The security of this game is a direct consequence of the security of the PIR protocol. Note that since each PIR instance uses its own randomness, even giving the adversary oracle access to PIR queries with the same parameters does not give the adversary an advantage. This definition addresses long-term lookups, short-term lookups can be defined analogously. Together with registration buddy privacy, these games ensure that the adversary does not learn part of the social graph.

**Game 6** (Integrity of Auxiliary Data). *1. The adversary selects an identity $A \in H$ and requests the challenger to compute the short-term registration token $\mathsf{RegisterShortTerm}(k'_A, t_j, D)$*

2. *The adversary additionally can request a number of other short- and long-term registration tokens, with the constraints that:*
   - *$A$ cannot perform any other short-term registration for the epoch $t_j$*
   - *$A$'s long-term registration for the epoch $T_j$ containing $t_j$ must only include buddies from $H$ (i.e., the adversary does not learn $K'_A$ for epoch $T_j$)*
3. *The adversary supplies an identity $B \in H$; the challenger then computes $\mathsf{ShortTermQuery}(t_j, \{K'_A\})$ and returns the queries to the adversary.*
4. *The adversary then comes up with a set of responses $\{response_i\}$.*
5. *The adversary wins if $\mathsf{ShortTermResult}(\{response_i\}) \notin \{\{D\}, \{\perp\}\}$*

Note that in this game, the adversary once again simulates the actions of the registration and PIR servers. Importantly, the adversary must not have compromised any of $A$'s buddies for the corresponding long-term epoch, as the auxiliary data is protected using AEAD with a key that is known to all the buddies.