MP3: A More Efficient Private Presence Protocol

Rahul Parhi, Michael Schliep, and Nicholas Hopper(✉)

University of Minnesota, Minneapolis, MN, USA
{parhi003,schli116,hoppernj}@umn.edu

Abstract. This paper presents a novel private presence protocol, referred to as MP3, where the service provider does not have any knowledge of the social graph of its users. In prior work, a private presence protocol, referred to as DP5, was presented. However, the size of the presence database in this protocol grows rapidly as the number of users increases; this limits its scalability and increases its cost. In the proposed protocol, the size of the presence database is reduced significantly, enabling significantly cheaper registration and lookup compared to that of DP5. MP3 requires about two-thirds the bandwidth of DP5 for $N = 200\,000$ users and about one-half the bandwidth of DP5 for $N = 1\,000\,000$ users. Furthermore, these savings grow asymptotically with the number of users. Additionally, the client-facing latency in MP3 is an order of magnitude less than that of DP5. We provide an evaluation of the performance and scalability of both protocols.

1 Introduction

Due to the recent threat and concern of government surveillance and collection of user data [1], an increasing number of services have emerged with the goal of protecting users' privacy from the provider of the service. A common approach is end-to-end secure messaging, which is currently employed in services such as Apple's iMessage, Open Whisper Systems' Signal Protocol, and Facebook's Messenger [2–4]. Secure messaging hides the content of the conversation from the provider by using strong cryptographic techniques, but the metadata of the conversation is still known to the service provider.

A critical part of secure messaging is presence, i.e., knowing when a friend is online. Although secure messaging providers do not have access to the plaintext conversation, they still have access to the set of friends of every user. This means that these services know the social graph of their users as well as the presence status of every user at any given time. To have a truly private communication platform, the metadata of the communication must also be protected.

An existing solution to this problem is DP5—the Daghstul Privacy Preserving Presence Protocol P—proposed by Borisov, Danezis, and Goldberg [5]. DP5 is the first private presence protocol to leak no information about the social graph to third parties and limit the information retained by the service itself.

S. Meiklejohn and K. Sako (Eds.): FC 2018, LNCS 10957, pp. 38–57, 2018.
https://doi.org/10.1007/978-3-662-58387-6_3

DP5 allows its users to see the online status of their friends in a completely private manner. It utilizes Private Information Retrieval (PIR) [6] for querying the service for a given user's buddy's presence.

The major weakness of DP5 is its *lack of* scalability for a large number of users. The presence database of DP5 grows much more rapidly than the number of users of the service. This results in a very expensive service for even a small number of users. To overcome this bottleneck, we propose MP3—the Minnesota Private Presence Protocol. MP3 maintains the same security goals as DP5. This is elaborated in Sects. 2.3 and 3.7. Compared to DP5, MP3 assumes that revoking and unrevoking friends is uncommon and thus we are able to significantly reduce the size of the presence database by using a *dynamic broadcast encryption scheme* [7]. As a result, MP3 is significantly cheaper to run for even a relatively modest user base. Additionally, these savings increase as the number of users increase. The *key contribution* of this paper lies in significantly reducing the size of the presence database compared to DP5; this allows cheaper registration and lookup queries in the context of the bandwidth required. The client-facing latency of MP3 is also an order of magnitude less than that of DP5 due to the smaller presence database.

This paper is organized as follows. Section 2 describes the background, goals, and related work pertaining to the MP3 protocol. Section 3 presents a detailed description of the MP3 protocol. Section 4 analyzes the performance of MP3 and compares it to that of DP5.

2 Background

The primary functionality of a private presence protocol is to allow for the registration of one's online presence and to allow for the query of the presence status of one's buddies in a completely private manner.

2.1 DP5 Overview

Since our design shares many characteristics with DP5, we give a brief overview of the protocol here. DP5 uses Private Information Retrieval (PIR), in which a database is distributed to several servers so that a user can query the servers to retrieve a specific record without revealing which record they retrieve. Given this functionality, a "trivial" private presence protocol would have each user A with n_A friends encrypt n_A *presence records* recording their status (and possibly other information, such as a contact address), with a shared key for each friend, and periodically upload this information to a presence database. When A's friend B wants to check on A's status, they would query the current database (using PIR) for the presence records encrypted under the symmetric key A shares with B. To hide information about the social graph, each user would need to pad the number of presence records uploaded per period to some maximum value denoted by N_{fmax}. This protocol results in a nearly quadratic-size database in the number of users. Moreover, the server-side computational costs of PIR scale linearly in the size of the database (and the bandwidth costs also increase, though sub-linearly).

To combat this inefficiency, DP5 splits the presence service into two asymmetric services. The primary, short-term, service is used to register and query presence of users with the precision of short windows (on the order of minutes). The secondary service is the long-term service, which is used to provide metadata for querying during the short-term. The long-term service also provides friend registration and revocation with the precision of long windows (on the order of days). As in the "trivial" protocol, it is assumed that users share a unique secret with each of their friends. Additionally, in order to not leak information about how many friends a given user has, DP5 defines a limit of N_{fmax} as the maximum number of friends a user may have.

In each period of the long-term service, a user (let's say Alice) of DP5 will upload her presence to the registration mechanism of DP5. This is referred to as Alice's long-term presence record. This record is actually *several* records, one for each of Alice's friends, padded with random records upto N_{fmax}. If, N_{fmax} is on the order of the number of users of the service, then the long-term presence database scales quadratically with the number of users, which in turn increases the amount of bandwidth and CPU this service requires. These long-term records contain information used by her friends to identify her during the short-term period. Then, during the short-term period, Alice uploads a single record to the short-term presence database in each short-term period she is online. Additionally, a *single* record containing a signature is uploaded to an auditable *signature database* during every short-term period. Thus, the short-term service, which is queried more often, grows only linearly with the number of online users.

To improve the DP5 protocol, we address the issue of scaling in the long-term service of DP5 by reducing the number of records Alice uploads in each long-term period to only a relatively constant number of records, yielding performance closer to that of the short-term service. We leave the short-term service of DP5 unchanged as it is already quite cheap.

2.2 Threat Model

We make standard assumptions about the users and adversaries of MP3. They are real world adversaries with common capabilities.

- We assume that honest users' end systems are secure and not compromised. We also assume that honest servers can maintain secrecy and integrity. Our design maintains forward security and does not require servers to store any long-term secrets.
- We allow the adversary to be an observer or a dishonest user of the system, and we assume they have not made any recent breakthrough in computational cryptographic assumptions, and assume that they cannot distinguish between different ciphertexts. More detailed assumptions are described in the protocol description in Sect. 3.

– Our security properties are under the covert model, i.e., adversaries will not act dishonestly if it would cause them to be detected and identified.
– Our protocol maintains availability against malicious parties. This is further detailed in Appendix B.

2.3 Security Goals

Here, we describe the goals required for a private presence service. Our goals are the same as the goals of DP5.

Privacy of Presence and Auxiliary Data. The presence status of a user and their auxiliary data should be available to only that user's explicit friends.

Integrity of Authentication and Auxiliary Data. The friends of Alice should not accept the presence and auxiliary data unless it was submitted by Alice.

Unlinkability of Sessions. It should be infeasible for a user to be linked between multiple uses of the service. The infrastructure and non-friend users should not be able to link the presence of another between epochs.

Privacy of the Social Graph. No information about the social graph of a user should be revealed to any other party of the service. More specifically, friends should not learn about other friends and the infrastructure should not learn any new information about a user.

Forward/Backward Secrecy of the Infrastructure. Any compromised keys stored in the infrastructure servers should not reveal past or future information that is secured with previous or future keys.

Auditability. All operations performed by the infrastructure should be auditable. A user should detect when their friend registration or presence registration has not been performed honestly by the service provider.

Support for Anonymous Channels. The protocol should not require any identifying information for operation. The use of an anonymous channel should only enhance the privacy of the system and the service will not compromise the anonymity of the user.

Indistinguishability of Revocation, Suspension, and Offline Status. A user is revoked if they are no longer able to query the presence status of the buddy that revoked them. A suspension is a temporary revocation, i.e., for some period of time, a user cannot query the presence status of the buddy that suspended them. This means a suspended buddy can be "unrevoked." Revocations and suspensions should not be distinguishable from being offline. For example, if Bob's buddy Alice appeared to be offline, Bob would not know if he was revoked or suspended, or if Alice was genuinely offline. MP3 provides *plausibly deniable* revocation and suspension of buddies. Plausibly deniable revocation means the transcript does not prove a user has been revoked. However, if a user does not see their friend come online for an extended period of time they may begin to assume they have been revoked. This is discussed in further detail in Sect. 3.7.

2.4 Related Work

DP5 seems to have been the first and only previous work to address social graph privacy in the context of presence services. Similar, but different, related work includes Dissent [8] and Riposte [9] which offer anonymous micro-blogging services; these systems are similar to private presence in that posting a micro-blog implies the author was online. Dissent is based on a DC-net with a client-server architecture. Clients in Dissent must form a group to post anonymous messages for each other using distrusted servers. Dissent provides anonymity within a static group. Riposte utilizes a novel private database writing mechanism based on techniques of PIR. Both of these systems have high latency when dealing with large anonymity sets and are not concerned with social graph privacy.

Two other similar and relevant anonymous messaging/microblogging systems that build on PIR techniques include Riffle [10] and Pung [11]. These systems allow for a user to send a message to their friends without revealing the social graph of the users. These messages could be used to indicate presence. However, these two systems assume every user uploads a message during every epoch. This implies that all users must be present at all times which is unrealistic and negates the need for a private presence protocol.

3 The MP3 Protocol

3.1 Overview

In this section, we provide an overview of MP3. MP3 is composed of two databases, a long-term database and a short-term database. The short-term database contains the presence status and information of a user. A new short-term database is generated on the order of minutes (5 min), we refer to these as short-term epochs. The long-term database contains information for a user to identify their friends' short-term database entries. A new long-term database is generated less frequently (once every 24 h), these are referred to as long-term epochs. Alice uploads her presence at most once for every long-term and short-term epoch. When Bob wants to check if his friend Alice is online, he first queries the long-term database and retrieves Alice's entry. Then he computes her short-term identifier and queries the short-term database for her presence. Each long-term database entry of a user contains information for looking up the next long-term database entry of that user. Alice may be unable to upload every long-term epoch so MP3 keeps the most recent long-term databases corresponding to 30 days.

The database entries are simple ⟨key, value⟩ pairs where the key is a unique identifier for a user in that specific epoch. These databases are queried using a hash-based PIR protocol of retrieving ⟨key, value⟩ pairs. Since these databases are queried with PIR, the infrastructure does not learn any information about a user's queries. In our implementation users upload their database entries to a single registration server and use a distributed PIR protocol with N_{lookup} servers for database queries. Similar to DP5, when querying the long-term database the users must query all long-term databases to avoid revealing which database contained the entry they needed.

The short-term database contains a single entry (presence message) per online user. For Alice (a) we denote presence message as $m_a(i)$ during short-term epoch t_i. This presence message may contain information such as how to contact Alice or a public key. If $m_a(i)$ is not present in during t_i, Alice is assumed to not be online.

MP3 uses a Dynamic Broadcast Encryption (DBE) [7] scheme for long-term database entries. DBE allows Alice to create a single constant-sized ciphertext that can be decrypted by all of her friends. Before participating in MP3, Alice generates a single DBE encryption key (mk) and a decryption key (dk_{af}) for each friend (f). During long-term epoch T_j Alice creates a DBE ciphertext with her short-term identity information for all short-term epochs that occur during T_j.

The dynamic part of DBE allows Alice to revoke decryption keys so the revoked keys cannot decrypt future ciphertexts. We utilize this to allow Alice to revoke up to N_{rev} friends in each long-term epoch. To provide plausible deniability of revocation, Alice distributes new decryption keys to the revoked friends. If Alice wants to truly revoke the friend she gives them a new randomly generated decryption key. If they are revoked for deniability reasons she gives them a new correctly generated decryption key. For the rest of the paper DBE.Revoke is used to refer to DBE revocations and MP3.Revoke is used to refer to Alice actually revoking a friend.

To unrevoke a previously MP3.Revoked friend, Alice constructs a special long-term database entry using the revoked friend's random decryption key. This record DBE.Revokes the random key and issues a new valid decryption key. This special entry must maintain N_{rev} DBE revocations to be indistinguishable from a regular entry. MP3 allows N_{unrev} friends to be unrevoked during a single long-term epoch. This means Alice uploads $N_{\text{unrev}} + 1$ entries, each of N_{rev} size for a long-term epoch. One entry to distribute short-term lookup information to her friends and possibly revoke N_{rev} friends. And N_{unrev} entries that allow her to unrevoke friends.

Finally, to prevent forged records, we employ two signature schemes, one for long-term presence records and another for short-term presence records. Thus, all of Alice's friends can be confident that the record they received from MP3's lookup mechanism can only belong to Alice. We use Elliptic Curve Digital Signature Algorithm (ECDSA) [12] for long-term epochs and Boneh-Lynn-Shacham (BLS) [13] signature scheme for short-term epochs. Moreover, an auditable *signature database* is employed during each short-term epoch.

3.2 Cryptographic Primitives

It is assumed that everyone participating in MP3 shares a set of known cryptographic primitives. We assume three primes $p_{dbe}, p_{dsa}, p_{bls}$, for simplicity we omit the subscripts when it is clear which prime we are discussing. Let G_1 and G_2 be two cyclic groups of prime order p_{dbe} and let G_T be a cyclic group also of prime order p_{dbe}. Denote $\mathbb{Z}_p$ as the ring of integers modulo p and denote $\mathbb{Z}_p^{\times}$ as the set of units in $\mathbb{Z}_p$, also denote $g \leftarrow G$ as randomly selecting an element g from a set G. An efficiently computable asymmetric pairing defined by the map $e : G_1 \times G_2 \rightarrow G_T$ is known so that for generators $g_1 \in G_1$, $g_2 \in G_2$ and for all $u, v \in \mathbb{Z}_p$

$$e(g_1^u, g_2^v) = e(g_1, g_2)^{uv}$$

The decisional Diffie-Hellman problem and the decisional co-Diffie-Hellman problem are assumed hard for G_1 and G_2. It is also assumed that our pairing is non-degenerate.

MP3 utilizes the following:

- ECDSA [12] signature scheme with group G_{dsa} of prime order p_{dsa} with generator g_{dsa}.
- BLS signature scheme [13] with groups G_3, G_4, G_5 of prime order p_{bls} with g_3 generating G_3 and an efficiently computable asymmetric pairing defined by the map $e_{bls} : G_3 \times G_4 \rightarrow G_5$.
- PRF_K, a keyed pseudorandom function that maps a short-term epoch timestamp to keys for AEAD described below.
- H_1, an efficiently computable hash function that maps the concatenation of the byte representations of long-term epoch timestamps and elements of G_T to elements of $\mathbb{Z}_p^{\times}$.
- H_2, an efficiently computable hash function that maps long-term public key to shared identifiers.
- H_3, an efficiently computable hash function that maps elements of G_T to elements of $\mathbb{Z}_p^{\times}$.
- H_4, an efficiently computable hash function that maps elements of G_1 to keys in the pseudorandom function above.
- H_5, an efficiently computable hash function that maps short-term epoch timestamps to elements of G_4.
- H_6, an efficiently computable hash function that maps elements of G_T to shared identifiers.
- $\text{AEAD}_K^{\text{IV}}(m)$, an authenticated encryption function where m is the message, K is the key, and IV is the initialization vector.

3.3 Dynamic Broadcast Encryption

In our construction of MP3, long-term epochs share many characteristics with a broadcast encryption scheme. In the context of MP3's long-term epoch, we use a *dynamic broadcast encryption* scheme with constant-size ciphertexts and decryption keys.

The definition of a Dynamic Broadcast Encryption (DBE) [7] is slightly different from a conventional broadcast encryption scheme in that it involves *two* authoritative parties: a *group-manager* and a *broadcaster*. The job of the group manager is to grant new users access to the group. The job of the broadcaster is to encrypt and transport messages to this group of users. When a message is encrypted, some members of this group can be revoked temporarily (suspended) or permanently (revoked) from decrypting the broadcasted message. Formally, a DBE scheme with revocation is a tuple of probabilistic algorithms (Setup, Join, Encrypt, Decrypt, Revoke, Update). These algorithms rely on an asymmetric pairing as described in Sect. 3.2.

Our design relies on the DBE scheme proposed by Delerablée et al. [7]. In our use case, every user is both a group-manager and a broadcaster. We introduce two additional operations (ShiftMK, ShiftDK) to support MP3's plausibly deniable revocation and suspension. Our modifications are detailed in Appendix A. We provide the relevant components of DBE as it pertains to MP3 below:

- DBE.Setup() - generates a *manager key* as the tuple $mk := (G, H, \gamma)$, where $G \leftarrow G_1$ and $H \leftarrow G_2$ are randomly selected generators and $\gamma \leftarrow \mathbb{Z}_p^\times$.
- DBE.Join($mk = (G, H, \gamma)$) - allows a user to friend someone by sharing a *decryption key* derived from their manager key as the tuple $dk := (x, A, B)$, where $x \leftarrow \mathbb{Z}_p^\times$ is fresh, $A := G^{\frac{x}{\gamma+x}}$, $B := H^{\frac{1}{\gamma+x}}$. If x and γ happen to be inverses, resample x. In our construction of MP3, we add an additional component to the decryption key, κ, a shared random symmetric key for AEAD that is persistent even when (x, A, B) is reassigned. Thus, the decryption key derived is $dk := (x, A, B, \kappa)$.
- DBE.Encrypt($mk = (G, H, \gamma)$) - generates a shared secret $K := e(G, H)^w$ and two ciphertexts: $C_1 := G^{w\gamma}$ and $C_2 := H^w$, where $w \leftarrow \mathbb{Z}_p^\times$.
- DBE.Decrypt($dk = (x, A, B, \kappa), C_1, C_2$) - takes as input a decryption key and the ciphertexts and computes the shared secret $K = e(C_1, B) \cdot e(A, C_2)$. Notice that this is the same shared secret computed by the broadcast manager in DBE.Encrypt.
- DBE.Revoke($mk = (G, H, \gamma), x_r, B_r$) - revokes the user with decryption key that contains x_r and B_r from the group of the user with manager key mk by updating $H := H^{\frac{1}{\gamma+x_r}}$. The user then advertises x_r and $H^{\frac{1}{\gamma+x_r}}$ to all their buddies.
- DBE.Update($dk = (x, A, B, \kappa), x_r, B_r$) - every non-revoked user with decryption key $dk = (x, A, B, \kappa)$ must update their decryption key via $B := \left(\frac{B_r}{B}\right)^{\frac{1}{x-x_r}}$ to revoke the user who owns x_r and B_r. Note that the revoked buddy will not be able to update their B value due to not being able to compute $\frac{1}{x_r-x_r}$. It is important to notice that this revocation is explicit to the revoked buddy, but in our construction of MP3 we add some auxiliary functionality in the long-term epoch to make revocations plausibly deniable as described in Sect. 3.7.
- DBE.ShiftMK($mk = (G, H, \gamma), \lambda$) - updates $G := G^\lambda$ and $H := H^\lambda$.
- DBE.ShiftDK($dk = (x, A, B, \kappa), \lambda$) - updates $A := A^\lambda$ and $B := B^\lambda$.

3.4 Setup

Initialization. To participate in MP3, Alice generates a manager key

$$mk_a \coloneqq \text{DBE.}\textsc{Setup}()$$

She also generates a set of private-public base key pairs

$$(Y_{a,\mathsf{init}}, P_{a,\mathsf{init}}) \quad \text{and} \quad (z_{a,\mathsf{init}}, q_{a,\mathsf{init}})$$

where $Y_{a,\mathsf{init}}$ is a randomly generated ECDSA private key and $P_{a,\mathsf{init}} = g_{dsa}^{Y_{a,\mathsf{init}}}$, and $z_{a,\mathsf{init}}$ is a BLS private key randomly selected from $\mathbb{Z}_p^\times$ and $q_{a,\mathsf{init}} = g_3^{z_{a,\mathsf{init}}}$.

Adding Buddies. For every friend f of Alice, she derives a decryption key

$$dk_{af} \coloneqq \text{DBE.}\textsc{Join}(mk_a)$$

and shares it along with her public base keys out-of-band. Assuming that the current long-term epoch is T_j, Alice also shares her most recent shared secrets K, from DBE.Encrypt, and R, her most recently generated random bit-string R, i.e., Alice shares[1] $(dk_{af}, P_{a,\mathsf{init}}, q_{a,\mathsf{init}}, K, R)$ with friend f out-of-band.

3.5 Long-Term Epoch

Registration. To register for epoch T_j, Alice must register during epoch T_{j-1}. Let $T_{j'}$ be the last long-term epoch in which Alice has registered. To begin, Alice computes a new long-term private-public key pair for T_j

$$h_j \coloneqq H_1(T_j \parallel K_{j'} \oplus R_{j'}), \quad Y_a(j) \coloneqq Y_{a,\mathsf{init}} \cdot h_j, \quad P_a(j) \coloneqq P_{a,\mathsf{init}}^{h_j}$$

as well as a new short-term private-public key pair for all short-term epochs during T_j

$$z_a(j) \coloneqq z_{a,\mathsf{init}} \cdot h_j, \quad q_a(j) \coloneqq q_{a,\mathsf{init}}^{h_j}$$

where $K_{j'}$ is Alice's shared secret from calling DBE.Encrypt in epoch $T_{j'}$ and $R_{j'}$ is a random bit-string generated in epoch $T_{j'}$. The $\oplus$ between $K_{j'}$ and $R_{j'}$ implicitly converts $K_{j'}$ to a bit-string of some length and the length of $R_{j'}$ is defined to be that length.

During a long-term registration, Alice has the opportunity to revoke or suspend, or unrevoke a certain number of her buddies. Denote the number of buddies she can revoke or suspend at each long-term epoch as N_{rev} and the number of buddies she can unrevoke at each long-term epoch as N_{unrev}.

Every long-term epoch registration, Alice will upload $1+N_{\text{unrev}}$ records to the long-term presence database. A single record is for all her buddies she wishes

[1] If this is the very first epoch, Alice must generate an initial $K \coloneqq K_{\mathsf{init}}$ and $R \coloneqq R_{\mathsf{init}}$ to share with her buddies to bootstrap the protocol.

to continue being buddies with or that she wishes to revoke or suspend. The remaining N_{unrev} records are for buddies she had previously suspended and wishes to "re-friend." Alice constructs the single record according to Algorithm 1 and she constructs the other N_{unrev} records according to Algorithm 2.

All long-term records are of the form

$$\overbrace{P}^{(a)} \parallel \underbrace{x_1 \parallel B_1 \parallel \cdots \parallel x_{N_{\text{rev}}} \parallel B_{N_{\text{rev}}}}_{(b)} \parallel \overbrace{E_1 \parallel \cdots \parallel E_{N_{\text{rev}}}}^{(c)} \parallel \underbrace{C_1 \parallel C_2}_{(d)} \parallel \overbrace{R}^{(e)} \parallel \underbrace{S}_{(f)}$$

where (a) is a long-term identifier, (b) contains the x and B values of all buddies to revoke or suspend, (c) contains new encrypted decryption keys for all buddies revoked in (b), (d) contains the ciphertext components from DBE, (e) is a random bit string, and (f) is the signature for the record that can be verified with (a). Further details of how (b) and (c) are used to allow *plausibly deniable* revocation and suspension are described in Sect. 3.7.

Upon receiving a record of the form

$$P \parallel x_1 \parallel B_1 \parallel \cdots \parallel x_{N_{\text{rev}}} \parallel B_{N_{\text{rev}}} \parallel E_1 \parallel \cdots \parallel E_{N_{\text{rev}}} \parallel C_1 \parallel C_2 \parallel R \parallel S$$

from Alice, the registration server verifies the signature S with P. If the signature is valid, it computes $\text{ID}_a(j) := H_2(P)$ and stores the $\langle$key, value$\rangle$ pair

$$\langle \text{ID}_a(j), x_1 \parallel B_1 \parallel \cdots \parallel x_{N_{\text{rev}}} \parallel B_{N_{\text{rev}}} \parallel E_1 \parallel \cdots \parallel E_{N_{\text{rev}}} \parallel C_1 \parallel C_2 \parallel R \parallel S \rangle$$

Otherwise, if the signature is invalid, nothing is stored.

Lookup. To look up Alice's presence for epoch T_j, Bob first requests the metadata associated with the databases from each lookup server. Since all lookup servers have the same database, the metadata should be the same, but in the event that some of the servers are dishonest, he takes the majority of the received metadata. The metadata contains information about the number of buckets and size of the buckets. He then computes $P_a(j) := P_{a,\mathsf{init}}^{H_1(T_j \parallel K_{j'} \oplus R_{j'})}$ using $K_{j'}$ and $R_{j'}$ he queried from the most recent long-term epoch in which Alice registered, and subsequently computes $\text{ID}_a(j) = H_2(P_a(j))$. Finally, he builds a PIR request using the metadata for $\text{ID}_a(j)$ to retrieve a record of the form

Algorithm 1. Alice computing her long-term presence record

```
    Input:
    $mk_a = (G_a, H_a, \gamma_a)$, Alice's manager key
    $(Y_a(j), P_a(j))$, Alice's private-public key pair for $T_j$
    $\mathcal{R}$, the set of buddies' decryption keys to MP3.Revoke or suspend
    $\mathcal{B}$, the set of buddies' decryption keys to continue being buddies
    Output: Long-term presence record
1:  function
2:      $(C_1, C_2, K) \coloneqq \text{DBE.Encrypt}(mk_a)$
3:      Generate and store a random bit-string $R_j$
4:      $\lambda \coloneqq H_3(K)$
5:      $\text{DBE.ShiftMK}(mk_a, \lambda)$
6:      Store $K_j \coloneqq K^{\lambda^2}$
7:      $record \coloneqq P_a(j)$
8:      $n \coloneqq N_{\text{rev}} - |\mathcal{R}|$
9:      $\mathcal{B}_{\mathsf{padded}} \coloneqq$ pad $\mathcal{B}$ with random decryption keys upto $N_{\text{fmax}}$
10:     $\mathcal{F} \coloneqq n$ decryption keys chosen uniformly from $\mathcal{B}_{\mathsf{padded}}$
11:     for each $(x, A, B, \kappa) \in \mathcal{R} \cup \mathcal{F}$ do
12:         $record \coloneqq record \parallel x \parallel H^{\frac{1}{\gamma + x_r}}$
13:         $\text{DBE.Revoke}(mk_a, x, B)$
14:     end for
15:     for each $(x, A, B, \kappa) \in \mathcal{R}$ do
16:         $x' \leftarrow \mathbb{Z}_p^{\times}$, $A' \leftarrow G_1$, $B' \leftarrow G_2$            ▷ Here, $x'$ is fresh
17:         ▷ Store the following in case we want to unrevoke this buddy
18:         Store $((x', A', B', \kappa), C_1, C_2, R_j)$ in a global set $\mathcal{U}$
19:         $E \coloneqq \text{AEAD}_{\kappa}^{j}(x' \parallel A' \parallel B')$
20:         $record \coloneqq record \parallel E$
21:     end for
22:     for each $(x, A, B, \kappa) \in \mathcal{F}$ do
23:         ▷ generate new and valid $x$, $A$, and $B$
24:         $(x', A', B', \_) \coloneqq \text{DBE.Join}(mk_a)$
25:         $E \coloneqq \text{AEAD}_{\kappa}^{j}(x' \parallel A' \parallel B')$
26:         $record \coloneqq record \parallel E$
27:     end for
28:     Shuffle the ciphertexts $E$ (the encrypted $x \parallel A \parallel B$) in $record$
29:     $record \coloneqq record \parallel C_1 \parallel C_2 \parallel R_j$
30:     $S \coloneqq \text{ECDSA-Sign}_{Y_a(j)}(record)$
31:     $record \coloneqq record \parallel S$
32:     return $record$
33: end function
```

$$x_1 \parallel B_1 \parallel \cdots \parallel x_{N_{\text{rev}}} \parallel B_{N_{\text{rev}}} \parallel E_1 \parallel \cdots \parallel E_{N_{\text{rev}}} \parallel C_1 \parallel C_2 \parallel R \parallel S$$

Bob then processes this long-term record according to Algorithm 3 and stores the returned K and R values for computing Alice's long- and short-term identifiers in the next long-term epochs.

Algorithm 2. Alice computing N_{unrev} presence records to unrevoke buddies

```
    Input:
    $\mathcal{U}$, the global set of buddies to unrevoke
    $T_j$, the current long-term epoch
    $mk_a$, Alice's manager key
    $(Y_{a,\mathsf{init}}, P_{a,\mathsf{init}})$, Alice's initial long-term epoch base keys
    $K_j$ stored from Algorithm 1
    $R_j$ stored from Algorithm 1
    Output: list of $N_{\text{unrev}}$ long-term presence records
 1: function
 2:     $ret$ := new list
 3:     for each $((x, A, B, \kappa), C_1, C_2, R_{\mathsf{revoked}}) \in \mathcal{U}$ to unrevoke do
 4:         Remove $((x, A, B, \kappa), C_1, C_2, R_{\mathsf{revoked}})$ from $\mathcal{U}$
 5:         $K_{\mathsf{revoked}} := \text{DBE.}\textsc{Decrypt}((x, A, B, \kappa), C_1, C_2)$
 6:         $Y_{\mathsf{revoked}} := Y_{a,\mathsf{init}} \cdot H_1(T_j \parallel K_{\mathsf{revoked}} \oplus R_{\mathsf{revoked}})$
 7:         $P_{\mathsf{revoked}} := P_{a,\mathsf{init}}^{H_1(T_j \parallel K_{\mathsf{revoked}} \oplus R_{\mathsf{revoked}})}$
 8:         $R_{\mathsf{unrevoked}} := K_j \oplus R_j \oplus K_{\mathsf{revoked}}$
 9:         $record := P_{\mathsf{revoked}}$
10:         ▷ We concatenate enough bytes so the record is the same length as that of Algorithm 1.
            We also make sure the byte encodings are of the correct type.
11:         $record := record \parallel \langle N_{\text{unrev}} - 1$ encrypted random triples $\overline{x} \leftarrow \mathbb{Z}_p^\times \parallel \overline{A} \leftarrow G_1 \parallel \overline{B} \leftarrow G_2\rangle$
12:         ▷ generate a fresh decryption key for the unrevoked buddy
13:         $(x', A', B', \_) := \text{DBE.}\textsc{Join}(mk_a)$
14:         $record := record \parallel \text{AEAD}_\kappa^j(x' \parallel A' \parallel B')$
15:         Shuffle the record as in Algorithm 1 line 28
16:         $record := record \parallel C_1 \parallel C_2 \parallel R_{\mathsf{unrevoked}}$
17:         $S_{\mathsf{unrevoked}} := \text{ECDSA-}\textsc{Sign}_{Y_{\mathsf{revoked}}}(record)$
18:         $record := record \parallel S_{\mathsf{unrevoked}}$
19:         store $record$ in $ret$
20:     end for
21:     if number of buddies unrevoked $< N_{\text{unrev}}$ then
22:         ▷ these records must be of the correct encoding
23:         store random records in $ret$ so that its length is $N_{\text{unrev}}$
24:     end if
25:     return $ret$
26: end function
```

3.6 Short-Term Epoch

Registration. To register for epoch t_i, Alice must register during t_{i-1}. Assume that epoch t_i is during epoch T_j. Recall that Alice computed the private-public key pair $(z_a(j), q_a(j))$ during the long-term registration for T_j. To begin, Alice encrypts her presence message, $m_a(i)$ as follows:

$$k_a(i) := \text{PRF}_{H_4(q_a(j))}(t_i) \qquad\qquad c_a(i) := \text{AEAD}^i_{k_a(i)}(m_a(i))$$

She then computes the unforgeable signature:

$$s_a(i) := H_5(t_i)^{z_a(j)}$$

Alice then uploads $c_a(i) \parallel s_a(i)$ to the short-term registration server.

Upon receiving Alice's record, the short-term registration server will compute $\text{id}_a(i) = H_6(e_{bls}(g_1, s_a(i)))$, and store $\langle \text{id}_a(i), c_a(i)\rangle$. Additionally, $\langle \text{id}_a(i), s_a(i)\rangle$ is stored in a short-term *signature database* to audit.

Algorithm 3. Bob processing the long-term record of Alice

```
**Input:**
$P_a(j)$, Alice's long-term key for $T_j$
$x_1 \| B_1 \| \cdots \| x_{N_{\text{rev}}} \| B_{N_{\text{rev}}} \| E_1 \| \cdots \| E_{N_{\text{rev}}} \| C_1 \| C_2 \| R \| S$, long-term record
**Output:** $K$ and $R$ for the next long-term epoch
 1: function
 2:     ▷ Only process the record if the signature is valid, otherwise the lookup server
        was malicious
 3:     if $S$ is valid signature for the record with $P_a(j)$ then
 4:         $\mathit{plausiblyRevoked} := \mathbf{false}$
 5:         for $i = 1$ to $N_{\text{rev}}$ do
 6:             if $x_{ab} \neq x_i$ then
 7:                 $\text{DBE.Update}(dk_{ab}, x_i, B_i)$
 8:             else
 9:                 $\mathit{plausiblyRevoked} := \mathbf{true}$
10:             end if
11:         end for
12:         if $\mathit{plausiblyRevoked}$ then
13:             for $i := 1$ to $N_{\text{rev}}$ do
14:                 Try to decrypt $E_i$ with $\kappa_{ab}$
15:                 if successfully decrypted $E_i$ then
16:                     $(x', A', B') := E_i$ decrypted with $\kappa_{ab}$
17:                     $x_{ab} := x'$, $A_{ab} := A'$, $B_{ab} := B'$
18:                 end if
19:             end for
20:             ▷ Note that we updated $dk_{ab}$ in line 17
21:             $K := \text{DBE.Decrypt}(dk_{ab}, C_1, C_2)$
22:             return $K$, $R$
23:         else
24:             $K := \text{DBE.Decrypt}(dk_{ab}, C_1, C_2)$
25:             $\lambda := H_3(K)$
26:             $\text{DBE.ShiftDK}(dk_{ab}, \lambda)$
27:             $K := K^{\lambda^2}$
28:             return $K$, $R$
29:         end if
30:     end if
31: end function
```

Lookup. To lookup Alice's presence for epoch t_i, Bob requests the metadata associated with the short-term databases in the same manner as in the long-term epoch. To begin, he first computes $q_a(j) := q_{a,\mathsf{init}}^{H_1(T_j \| K_{j'} \oplus R_{j'})}$ using $K_{j'}$ and $R_{j'}$ he queried from the most recent long-term epoch in which Alice registered. Then he computes $\text{id}_a(i) := H_6(e_{bls}(q_a(j), H_5(t_i))$ which is equivalent to the registration server's computation of $H_6(e_{bls}(g_3, s_a(i)))$ by the properties of pairings. He then builds a PIR request for $\text{id}_a(i)$ to retrieve $c_a(i)$. Bob can be certain that $c_a(i)$ is from Alice due to the unforgeable signature $s_a(i)$ by auditing the signature database.

Bob can then compute $k_a(i) := \text{PRF}_{H_4(q_a(j))}(t_i)$ and decrypts $c_a(i)$ retrieving $m_a(i)$. The decryption being successful implies that Alice is online during epoch t_i. As in the long-term epoch, in order to not leak information of how many buddies Bob has, he must pad his lookup to N_{fmax} ids.

3.7 Details

Unrevoking and Unsuspending. For simplicity in this section we assume $N_{\text{rev}} = 1$. The long-term presence record of Alice that MP3.Revokes Bob takes the form

$$P \parallel x_{ab} \parallel B_{ab} \parallel E_{ab} \parallel C_1 \parallel C_2 \parallel R \parallel S$$

That is x_{ab} and B_{ab} are Bob's x and B values and E_{ab} contains a triple of random (x, A, B). encrypted with κ_{ab}, i.e., x, A, and B were *not* generated from Alice's manager key. This means that for future epochs, Bob cannot compute the proper K (Algorithm 3 line 21), and thus can no longer query for Alice. Alice can unrevoke Bob by computing the incorrect K that he computes from when he was revoked (Algorithm 2 line 5) and use this to upload a record that Bob *can* query for. This record will allow Bob to compute the correct K and R values for Alice for future epochs, as well as providing Bob with fresh x, A, and B values that *are* generated from Alice's manager key (Algorithm 2 line 14). This is possible by storing the correct K value within the R value in this "unrevoking" record (Algorithm 2 line 8). This allows Bob to compute the proper identifier for Alice in the next long-term epoch. Thus, Bob can continue to query Alice's presence as before.

Alice's friends must process all of her long-term database entries so they must query all long-term databases. To allow users to be offline for extended periods of time MP3 stores the previous 30 days worth of long-term database. If a user does not come online for more than 30 days they must share new keys with all of their friends. Revocations in the database entries are plausible deniable but a user may be able to notice if a friend does not come online anymore indicating they may have been revoked. This problem is inherent to presence systems.

Plausible Deniability of Revocation and Suspension. For MP3.Revoke to be deniable a revoked user must not be able to determine they have been revoked. MP3 implements this by DBE.Revoking users that have not been MP3.Revoked and issues these users new valid DBE decryption keys. Where as friends which are MP3.Revoked are issued a new random decryption key. For a friend to determine if they have been MP3.Revoked they must be able to distinguish between valid and random decryption keys. We introduce DBE.ShiftMK and DBE.ShiftDK to make the decryption keys indistinguishable. Appendix A details a distinguisher if DBE.ShitfMK and DBE.ShiftDK are not used.

More formally, if a friend can distinguish a transcript where they have been MP3.Revoked from a transcript where they have been DBE.Revoked but not MP3.Revoked they can be used as a Decisional Diffie-Hellman (DDH) distinguisher. That is, given (g, g^x, g^y, g^z) determine if $g^{xy} = g^z$. We assume all hash functions are modeled as random oracles.

We now quickly sketch the proof. If given a decryption key (x, A, B) and ciphertext $(C_1 = G^{w_0\gamma}, C_2 = H^{w_0})$ and ciphertext $(C_1' = G^{w_1\gamma\lambda}, C_2 = H^{w_1\frac{1}{\gamma+x}\lambda})$ where w_0, γ, w_1, and λ are random, and given (x', A', B'), determine if (x', A', B') are valid decryption keys or random. Given a DDH challenger we can construct the above problem. Let the manager key $mk = (G = g, H = g^x, \gamma)$. Then $C_1 = g^{w_0\gamma}$, $C_2 = g^{xw_0}$. Define $\lambda = y$, thus $mk' = (G = g^y, H = g^{z\frac{1}{\gamma+x}}, \gamma)$ and $C_1' = g^{yw_1\gamma}$, and $C_2' = g^{z\frac{1}{\gamma+x}w_1}$. If a friend can determine they have been MP3.Revoked they can win the DDH game.

Complexity Comparisons. During each long-term epoch in DP5, $N \cdot N_{\text{fmax}}$ records are stored, where each record is a constant size. Thus, the registration bandwidth is $\Theta(N \cdot N_{\text{fmax}})$. During each long-term epoch in MP3, $N \cdot (N_{\text{unrev}} + 1)$ records are stored, where each record's length scales with N_{rev}. Thus, the registration bandwidth complexity is $\Theta(N \cdot N_{\text{unrev}} \cdot N_{\text{rev}})$. In reality, N_{unrev} and N_{rev} will be relatively constant[2] compared to N. This implies that (as functions of N and N_{fmax}) the registration bandwidth complexities for DP5 and MP3 are $\Theta(N \cdot N_{\text{fmax}})$ and $\Theta(N)$, respectively.

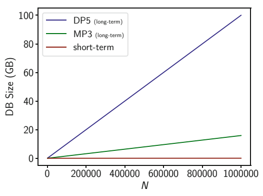
With the PIR protocol used in both DP5 and MP3, the bandwidth cost per query of a single record scales with the square root of the size of the database[3]. Also, recall that each user must query for N_{fmax} buddies to not reveal any information about their number of buddies. This implies that the bandwidth complexities for an entire long-term epoch (assuming all users query) for DP5 and MP3 are $\Theta\left(N^{3/2} \cdot N_{\text{fmax}}^{3/2}\right)$ and $\Theta\left(N^{3/2} \cdot N_{\text{rev}}^{1/2} \cdot N_{\text{unrev}}^{1/2} \cdot N_{\text{fmax}}\right)$, respectively. Using the same approximations for N_{unrev} and N_{rev} as above, this results in lookup bandwidth complexities of $\Theta\left(N^{3/2} \cdot N_{\text{fmax}}^{3/2}\right)$ for DP5 and $\Theta\left(N^{3/2} \cdot N_{\text{fmax}}\right)$ for MP3. Similar arguments can be made for the shared short-term epoch of DP5 and MP3 and are summarized in Table 1.

4 Experimental Results

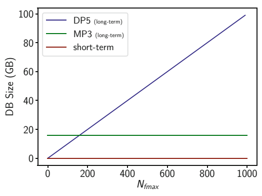
Implementation. Our MP3 library is implemented in 350 lines of C and 4000 lines of C++. The core cryptography relies on OpenSSL for AES, SHA-256, and elliptic curve arithmetic and signatures; RELIC [14] for pairing-friendly curves; and Percy++ [15] for PIR. The groups G_1, G_2, and G_T are defined by the Optimal Ate pairing over a 256-bit Barreto-Naehrig curve. We use a 224-bit Elliptic Curve, specifically secp224r1, for ECDSA, though the choice was

[2] Arguments for why this is a valid assumption are discussed in Sect. 4.

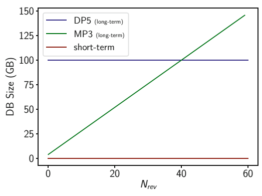
[3] As in the PIR protocol in DP5, we constructed $r = \lceil \sqrt{ns} \rceil$ buckets and we can upper bound the size of each bucket by $\left(\frac{n}{r} + \sqrt{\frac{n}{r}}\right) \cdot s \approx \sqrt{ns} + \sqrt[4]{ns^3}$ (here, s is the size of each record in bytes); since a query results in an entire bucket, this scales with the square root of the size of the database.



(a) $N_{\text{fmax}} = 1000,\ N_{\text{rev}} = N_{\text{unrev}} = 5$

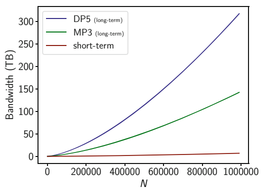


(b) $N = 1\,000\,000,\ N_{\text{rev}} = N_{\text{unrev}} = 5$

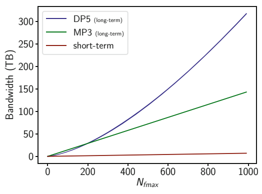


(c) $N = 1\,000\,000,\ N_{\text{fmax}} = 1000,\ N_{\text{unrev}} = 5$

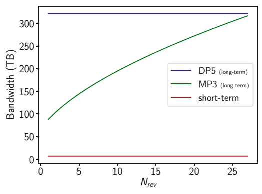
Fig. 1. Presence database sizes



(a) $N_{\text{fmax}} = 1000,\ N_{\text{rev}} = N_{\text{unrev}} = 5$



(b) $N = 1000000,\ N_{\text{rev}} = N_{\text{unrev}} = 5$



(c) $N = 1000000,\ N_{\text{fmax}} = 1000,\ N_{\text{unrev}} = 5$

Fig. 2. Lookup server bandwidth

arbitrary. AEAD is implemented using AES in Galois/Counter Mode (GCM). The PRFs and hash functions are implemented using SHA-256.

Evaluation. To evaluate the performance of MP3 vs. DP5, we simulated both protocols in a "worst-case" scenario with the number of users ranging from 1000 to 1 000 000 clients. To simulate the worst-case scenario, we had all clients perform registration and lookup for all epochs. All simulations were run on a machine with dual Intel Xeon E5-2630 v3 CPUs and 256GB of RAM. The number of PIR servers (N_{lookup}) was held fixed at 3 for all setups and both protocols. The equivalent of 1 year of execution were simulated in all setups. For both MP3 and DP5, the most expensive components are the lookup servers from both CPU and bandwidth perspectives.

Figure 1 compares the size of the presence databases for the long-term epoch of MP3 and DP5 as well as the shared short-term epoch. If we fix N_{rev} and N_{unrev} to small constants compared to N, it is obvious that the size of the long-term database of MP3 is significantly less than that of DP5. A smaller database implies cheaper lookup costs in the context of both bandwidth and CPU. Additionally, we can raise N_{rev} quite a bit and still maintain a smaller long-term database than that of DP5. It's also important to see how cheap the short-term database is relative to the long-term databases. Also note that the presence database sizes are proportional to the registration bandwidth.

Figure 3 compares the client-facing latency of the long-term epochs of MP3 and DP5 as well as the shared short-term epoch, for $N = 100\ 000$ users. The latency of MP3's long-term epoch is smaller than that of DP5 due to the inherently smaller database. The short-term epoch's latency is even less as the short-term databases are even smaller.

Figure 2 compares the bandwidth of a single long-term lookup server for the long-term epoch of MP3 and DP5 as well as the shared short-term epoch. The same pattern occurs that we saw in Fig. 1 - for relatively constant N_{rev} and N_{unrev}, the bandwidth required is significantly less for MP3 than for DP5 and the bandwidth requirements of the short-term epoch are negligible compared to that of the long-term epochs.

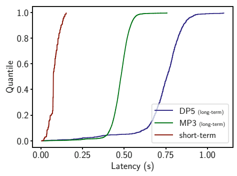


Fig. 3. Client-facing lookup latency excluding RTT. $N = 100\ 000$, $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$

Table 1. Bandwidth complexities comparing MP3 and DP5 as a function of N, N_{fmax}, N_{rev}, and N_{unrev}.

		Client		Server	
		registration	lookup	registration	lookup
long-term	DP5	$\Theta(N_{\text{fmax}})$	$\Theta\left(N^{1/2} \cdot N_{\text{fmax}}^{3/2}\right)$	$\Theta(N \cdot N_{\text{fmax}})$	$\Theta\left(N^{3/2} \cdot N_{\text{fmax}}^{3/2}\right)$
	MP3	$\Theta(N_{\text{rev}} \cdot N_{\text{unrev}})$	$\Theta\left(N^{1/2} \cdot N_{\text{rev}}^{1/2} \cdot N_{\text{unrev}}^{1/2} \cdot N_{\text{fmax}}\right)$	$\Theta(N \cdot N_{\text{rev}} \cdot N_{\text{unrev}})$	$\Theta\left(N^{3/2} \cdot N_{\text{rev}}^{1/2} \cdot N_{\text{unrev}}^{1/2} \cdot N_{\text{fmax}}\right)$
short-term	DP5 MP3	$\Theta(1)$	$\Theta\left(N^{1/2} \cdot N_{\text{fmax}}\right)$	$\Theta(N)$	$\Theta\left(N^{3/2} \cdot N_{\text{fmax}}\right)$

Discussion of Scalability and Cost Improvements. A primary bottleneck in DP5 is its lack of scaling with large number of users, specifically for long-term epochs. MP3 solves just that. The complexity for bandwidth usage of all operations are summarized in Table 1.

The bulk of the cost in running a service such as MP3 or DP5 comes from the bandwidth usage of the given protocol. N_{rev} and N_{unrev} are always less than or equal to N_{fmax} by definition. In reality, with a 24-h long-term epoch, setting N_{rev} and N_{unrev} to a small constant is very reasonable[4]; therefore, MP3 is significantly cheaper during long-term epochs, and thus overcomes the scalability bottleneck of DP5.

In Fig. 2a, with $N = 1\ 000\ 000$ users, $N_{\text{fmax}} = 1000$, $N_{\text{rev}} = N_{\text{unrev}} = 5$, we can see that MP3 uses about half the bandwidth of that of DP5 and it's evident that the savings grow as the number of users increases.

5 Conclusion

The proposed protocol reduced the complexity and cost of the most expensive component of DP5, i.e., the long-term presence database. In reference to

[4] Social networks such as Twitter disallow bulk unfollowing [16], making our argument about setting N_{rev} and N_{unrev} to a small/constant value even stronger.

DP5, MP3 requires about half the bandwidth for $N = 1\,000\,000$ users, and this reduction only increases as the number of users increases. Therefore, MP3 is a more efficient private presence protocol than DP5. Future work will be directed towards formally proving the security properties of MP3.

Acknowledgments. This research was partially supported by the NSF under grant 1314637 and by an Undergraduate Research Opportunities Program (UROP) Award from the University of Minnesota.

A Modifications to Dynamic Broadcast Encryption

Recall the operations of DBE from Sect. 3.3. Our modifications to DBE [7] only add the DBE.ShiftMK and DBE.ShiftDK operations. These operations are required for plausibly deniable revocations and suspensions. Recall that in a long-term database record for Alice in which Bob is actually revoked, Bob's old decryption key (dk_{ab}) is revoked and he is issued a new, but invalid, decryption key (dk'_{ab}). Without DBE.ShiftMK and DBE.ShiftDK, Bob could use dk'_{ab} to invert the Update operation and detect whether he was revoked or not.

The inverse update function is:

– DBE.Update^{-1}($dk' = (x', A', B', \kappa'), x_r, B_r$) - takes as input a decryption key and a revocation values and computes a new decryption key $dk := (x', A', B, \kappa')$ where $B := \frac{B_r}{B'^{(x'-x_r)}}$ that *can* decrypt ciphertexts *created before* the revocation.

Assuming DBE.ShiftMK and DBE.ShiftDK are not in place, a revoked user Bob can use DBE.Update^{-1} to *detect* that he has been revoked by Alice. Given two long-term presence records of Alice, where the former has not revoked Bob and the latter has revoked Bob, Bob can apply the DBE.Update^{-1} function to his new decryption key and compute a decryption key for the former presence record.

Let Bob's decryption keys for the former presence record be dk_{ab} and let Bob's decryption key for the latter presence record (after calling DBE.Update^{-1}) be dk'_{ab}. Also let C_1, C_2 be the ciphertext components from the former presence record. To detect if he has been revoked, all he must do is check if DBE.Decrypt(dk_{ab}, C_1, C_2) $\neq$ DBE.Decrypt(dk'_{ab}, C_1, C_2). If the statement is true, then Bob has been revoked by Alice.

By introducing DBE.ShiftMK and DBE.ShiftDK we create a one-way operation to the revocation process of MP3; thus Bob *cannot* invert the DBE.ShiftMK and DBE.ShiftDK functions without the knowledge of the plaintext of (C_1, C_2) and therefore cannot detect whether or not he was revoked.

B Availability Against Malicious Parties

Some conventional approaches to ensure availability against malicious parties cannot be applied directly to privacy-preserving protocols, as they can leak information. This causes several challenges: keeping the databases small, ensuring the registration server stores all uploaded presence records, ensuring the lookup servers store all presence records and do not modify them.

A malicious client could upload many presence records during a given epoch, causing a denial of service (DoS) for all other clients. If a malicious client were using an anonymous channel, authentication would compromise the anonymity of that client, defeating the purpose of MP3. To eliminate this, k-times anonymous authentication schemes have been proposed [17,18]. In these schemes, users are guaranteed anonymity up to k times; that is, if a user authenticates $k+1$ times, the identity of the user can be computed. Such private rate-limiting schemes can be used to limit the number of times a client registers during a given epoch without losing anonymity.

In the case that the registration server is dishonest and drops records, a user could "friend themself" to ensure that their presence records are being stored, by looking themself up during every epoch. Note that all presence records are indistinguishable, so the registration server can not target specific records for dropping.

Lastly, in the case that the lookup servers are dishonest and modify the database, Devet et al. propose a robust PIR scheme [19] that allows detection of malicious servers. This detection requires at least $t+2$ honest servers, where t is the number of servers needed to collude to be able to determine the data in a query. This robust PIR scheme is implemented in MP3.

References

1. Rushe, D.: Lavabit founder refused FBI order to hand over email encryption keys. The Guardian, October 2013
2. Apple: Approach to privacy. http://www.apple.com/privacy/approach-to-privacy/
3. Open Whisper Systems. https://whispersystems.org/
4. Marlinspike, M.: Facebook messenger deploys signal protocol for end to end encryption. https://whispersystems.org/blog/facebook-messenger
5. Borisov, N., Danezis, G., Goldberg, I.: DP5: a private presence service. Proc. Priv. Enhanc. Technol. **2015**(2), 4–24 (2015)
6. Chor, B., Goldreich, O., Kushilevitz, E., Sudan, M.: Private information retrieval. In: IEEE Symposium on Foundations of Computer Science (1995)
7. Delerablée, C., Paillier, P., Pointcheval, D.: Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In: Takagi, T., Okamoto, E., Okamoto, T., Okamoto, T. (eds.) Pairing 2007. LNCS, vol. 4575, pp. 39–59. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73489-5_4
8. Wolinsky, D.I., Corrigan-Gibbs, H., Ford, B., Johnson, A.: Dissent in numbers: making strong anonymity scale. Presented as Part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2012), pp. 179–182 (2012)
9. Corrigan-Gibbs, H., Boneh, D., Mazières, D.: Riposte: an anonymous messaging system handling millions of users. In: 2015 IEEE Symposium on Security and Privacy, pp. 321–338. IEEE (2015)
10. Kwon, A., Lazar, D., Devadas, S., Ford, B.: Riffle. Proc. Priv. Enhanc. Technol. **2016**(2), 115–134 (2016)

11. Angel, S., Setty, S.T.: Unobservable communication over fully untrusted infrastructure. In: OSDI, pp. 551–569 (2016)
12. ANSI X9.62–1998: Public key cryptography for the financial services industry: the elliptic curve digital signature algorithm (ECDSA). American National Standards Institute (ANSI), Washington, DC (1998)
13. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the Weil pairing. In: Boyd, C. (ed.) ASIACRYPT 2001. LNCS, vol. 2248, pp. 514–532. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45682-1_30
14. Aranha, D.F., Gouvêa, C.P.L.: RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.
15. Goldberg, I., Devet, C., Lueks, W., Yang, A., Hendry, P., Henry, R.: Percy++ project on sourceforge (2014). http://percy.sourceforge.net
16. Suspension - what's the daily/hourly unfollow limit for each user? What is the aggressive behaviour? https://twittercommunity.com/t/suspension-whats-the-daily-hourly-unfollow-limit-for-each-user-what-is-the-aggressive-behaviour/13971
17. Liu, J.K., Wei, V.K., Wong, D.S.: Linkable spontaneous anonymous group signature for ad hoc groups. In: Wang, H., Pieprzyk, J., Varadharajan, V. (eds.) ACISP 2004. LNCS, vol. 3108, pp. 325–335. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27800-9_28
18. Tsang, P.P., Au, M.H., Kapadia, A., Smith, S.W.: Blacklistable anonymous credentials: blocking misbehaving users without TTPs. In: Proceedings of the 14th ACM Conference on Computer and Communications Security, pp. 72–81. ACM (2007)
19. Devet, C., Goldberg, I., Heninger, N.: Optimally robust private information retrieval. Presented as Part of the 21st USENIX Security Symposium (USENIX Security 12), pp. 269–283 (2012)