Privately (and Unlinkably) Exchanging Messages Using a Public Bulletin Board*

Jaap-Henk Hoepman
Institute for Computing and Information Sciences (ICIS)
Radboud University
the Netherlands
jhh@cs.ru.nl

ABSTRACT

We implement a secure and privacy friendly asynchronous unidirectional message transmission protocol using a *public* bulletin board that makes individual send or receive events unlinkable to one another. While the clients must securely run in the user's endpoint device, the bulletin board can be hosted on an arbitrary public cloud at no additional risk. In fact the protocol provides the same unlinkability guarantees as the underlying mixing network. The protocol is efficient, and the central bulletin board can adaptively be scaled and distributed depending on the load. We show how the basic unidirectional protocol can be used to hide the metadata in bidirectional message exchange applications like WhatsApp, and how it can be used to implement a private presence service efficiently. In these applications our protocol protects the so-called social graph.

1. INTRODUCTION

Messaging apps that allow people to asynchronously exchange messages are hugely popular. However, people have become increasingly aware of the extent of the dragnet surveillance performed by intelligence agencies and show a decreasing trust in the providers of such messaging services service themselves. This has created a market for more secure and privacy friendly implementations of such messaging services.

Ideally such services should both hide the *content* of the messages exchanged as well as the sender and recipient (i.e the *metadata*) associated with the communication. Moreover, both content and metadata should be hidden from both an external (active) adversary as well as the service provider itself. In particular the social graph, i.e. the graph representing who is connected to whom, should be protected. In the extreme case the service should offer you the possibility to anonymously establish a secure contact that allows you to securely and anonymously communicate with that contact at a later stage.

Our research is partly motivated by the following use case. Consider a journalist that wants to stay in touch with (anonymous) contacts and informants in a hostile environment, like a civil war zone or a country with limited protection to journalists (or informants). Exchanging phone numbers to stay in touch is dangerous, as these can be traced back to their owner. Yet even very secure and privacy friendly messaging applications like Signal need such a phone number to create an account and to establish first contact. If you don't even need a phone number to connect to each other, loss or theft of your phone means your contacts cannot be traced (as would be the case if their phone numbers would be in your address book). This is relevant, as the following quote from the recent Freedom on the Net 2014 report [17] shows.

> The increased use of "social engineering" – essentially tricking users into revealing information – and account hijacking has reinforced the idea that one's own digital security often depends on the actions and judgment of those in one's broader social or professional network.

We wish to stress that although this example motivated our research, the system we propose has not been analysed, implemented or tested to ensure that it provides strong enough protection for this particular use case.

Substantial improvements to protect the *content* of messages exchanged has been made. And in fact several so called secure messaging apps are readily available[1]. However, none of these apps protect the *metadata*. This is problematic because this metadata allows adversaries (or the service providers themselves) to reconstruct the social graph, which is in itself quite sensitive[2] [18].

One way to limit metadata collection by the service provider is to avoid any centralised server and instead implement a peer-to-peer protocol. Yet a truly peer-to-peer solution is often impractical, because devices like mobile phones may not always be on line, have non-fixed IP addresses (especially when roaming or when connected using a cellular network) or may live behind a firewall[3]. Also, many people may be reluctant or incapable to install and run a local server that could serve as a proxy to implement such a P2P protocol.

*This research is supported by the Netherlands Organization for Scientific Research (NWO) as project 'Patterns for Privacy' (CYBSEC.14.030). This research is conducted within the Privacy and Identity Lab (PI.lab) and funded by SIDN.nl (http://www.sidn.nl).
Version: Wed Jul 29 20:22:57 2015 +0200 / wpes2015-cr / abb-wpes.tex

WPES'15, October 12, 2015, Denver, Colorado, USA.

ACM 978-1-4503-3820-2/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2808138.2808142.

[1] https://www.eff.org/secure-messaging-scorecard
[2] See http://www.principiadiscordia.com/forum/index.php?topic=34746.msg1285642#msg1285642 and http://firstmonday.org/article/view/2611/2302.
[3] One of the main reasons Skype was so successful was that it was smart in figuring out how to bypass specific firewall settings.

In short, for large scale adoption of a messaging app a more centralised approach that does not require any user-side setup is preferred. Yet any centralised solution runs, almost by definition, the risk of exposing the social graph. Existing solutions to prevent this (as discussed in section 8) are inefficient or otherwise inappropriate.

In this paper we show that it is indeed possible to protect the metadata associated with exchanging asynchronous messages, even when implementing the system using a centralised server. In fact we implement a secure and privacy friendly message transmission protocol (defined in section 3) using a *public* bulletin board that makes individual send or receive events unlinkable to one another. The protocol is *asynchronous*, allowing senders to send their messages even when the intended receiver is off line, and *unidirectional*, meaning that Alice can send messages to Bob without necessarily allowing Bob to reach Alice. The protocol is present in section 5 and the correctness and security arguments are provided in section 6.

The system assumes that the communication network itself does not leak the identity of the sender or the receiver connecting to the bulletin board. This assumption can for example be met by letting all users connect to the bulletin board over a mixing network [5] like Mixminion [10], using a compact yet secure packet format like Sphinx [11]. A low-latency anonymising network like Tor[4] is not necessary for an application like messaging, and actually provides weaker anonymity guarantees.

The use of a generic mixing network, that is not tied the particular application at hand, allows the application traffic to be mixed with traffic from other applications that also use the same mixing network. This increases the level of privacy provided to both applications. We note that a different way of looking at our results is to view the protocol that we design as a way to make mixing networks asynchronous.

The cost of sending or receiving a message is $O(1)$. The size of the public bulletin board can be adapted dynamically, and the message processing load can be distributed over several servers. The use of a public bulletin board maximises the amount of information available to both the service provider and an external adversary (and in fact makes them essentially equally powerful). Still the content *and* metadata are protected by the protocol. Because of this design choice, the bulletin board can be hosted on an arbitrary public cloud at no additional risk. Of course the client side code must securely run on the endpoint devices of the users.

The basic unidirectional protocol can be used to hide the metadata and the social graph in bidirectional messaging apps like WhatsApp or Signal. It can also can be used to implement a private presence service efficiently. Section 7 discusses this. We finish the paper with a discussion of our results in Section 9 and our conclusions in Section 10. But first we present a simple, but broken idea to solve the problem at hand, because it serves to motivate the actual approach taken in this paper.

2. A SIMPLE, BUT BAD IDEA

Suppose Alice and Bob share two secret 16 character strings `AB` and `BA`. These strings should be unrelated, and should be different from any other such strings shared among other users. Alice also has, for each user B she wants to communicate with, a pair of asymmetric encryption keys (k_{AB}, K_{AB}) and asymmetric signature keys (s_{AB}, S_{AB}). The same holds for Bob.

[4] https://www.torproject.org

Let Bob create a Tor hidden service `AB.onion` hosted by the bulletin board server, using the 16 character name `AB`. This hidden service is like a point-to-point mailbox responsible for relaying messages from Alice to Bob. It implements a post function for Alice to drop messages to Bob and a fetch function for Bob to retrieve any messages left by Alice since the last fetch. These messages are signed using s_{AB} and then encrypted against K_{BA}. (The signature is necessary to prevent fake messages, but it needs to be embedded within a layer of encryption to prevent tracking of Alice based on the signatures on her messages.) Similarly let Alice create 'mailbox' `BA.onion`.

The problem with this approach is twofold. First of all, when Alice and Bob engage in a conversation, the server will observe that mailboxes `AB.onion` and `BA.onion` are accessed alternatingly quickly in succession. The server will eventually conclude that these two mailboxes connect two users. Secondly, if Alice has several mailboxes `BA.onion`, `CA.onion`, etc., and routinely checks her mailboxes every morning at breakfast, then the server will see that each day at around the same time a sequence of mailboxes is accessed. It may then conclude that `BA.onion`, `CA.onion` belong to the same person. Combining these two heuristics allows the server to reconstruct the social graph!

3. DEFINING THE PROTOCOL

An *asynchronous unidirectional private point-to-point message transmission protocol* allows one user to send messages asynchronously to another user in private. The user can send a message even if the recipient is off line, such that the message will be delivered later once the recipient becomes on line. It protects the content as well as the metadata associated with the transmission of any message, against both an external adversary as well as the provider of the service itself.

Such a message transmission protocol implements the following functions for a sender Alice (A) and a receiver Bob (B).

$setup_{AB}$ Used by Alice and Bob to establish contact. Exchanges keys and necessary state information required before Alice can send messages to Bob.

$send_{AB}(m)$ Used by Alice to send a message m to Bob.

$receive_{AB}$ Used by Bob to receive the next available message from Alice, if available. Returns $\perp$ if no message is available.

In this paper we focus on the design of the protocols to send and receive messages, and will simply assume that Alice and Bob have found a way to initially exchange keys and any other necessary information[5]. We do this to show that such message transmission protocols can indeed be implemented with very strong privacy guarantees even if we use a public bulletin board to implement the service.

3.1 Requirements

The point-to-point message transmission protocol should satisfy the following requirements. Suppose Alice successfully sends a message m to Bob. Then

correctness Bob eventually receives m (unaltered). Moreover, Bob receives its messages in the same order as Alice sends them.

[5] In the use case described in the introduction, one way to anonymously establish contact when Alice and Bob meet in person is to 'bump' their smart phones to exchange data using some proximity channel. Alternatively, they can mutually scan a QR code displayed on the other device.

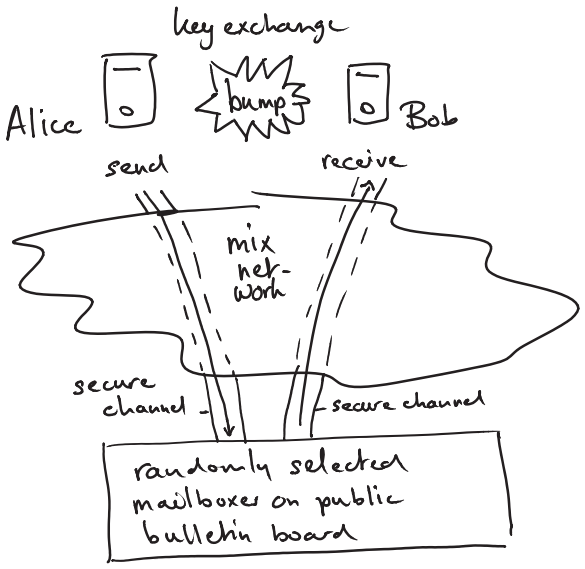

Figure 1: System model.

confidentiality Nobody else (except Alice and Bob) learns m.

integrity If Bob receives a message m', m' was sent to Bob (by one of his contacts).

authenticity Alice is guaranteed that only Bob receives m, and Bob is guaranteed that Alice was the sender of m (if he receives m).

unlinkability of events The adversary cannot link any two send and/or receive events, *except* the events that happen at the bulletin board that receives a single message with the corresponding event that sent that message, or any sequence of unsuccessful receives of the next expected message from a particular user and the final successful receipt of that message.

unlinkability of relationships No two users can determine whether they are sharing a contact.

forward security If a user device is compromised, the adversary is only able to obtain the contents of (or tamper with) future messages to or from this user. In particular, the adversary is unable to trace the contacts of a compromised user.

availability A sender will eventually be able to send a message, and a message sent will eventually be delivered.

3.2 Informal system model

For the implementation of our transmission protocol we assume the following system model.

A central server hosts a public bulletin board, that all users in the system have access to. Conceptually, it is an array of n cells, indexed by arbitrary integers modulo n. Although the bulletin board is in principle publicly accessible as is, an API containing a set of functions to access the cells in a structured manner (tailored to the protocol we will define) will be defined further on.

Users send and receive messages using a personal device with a network connection. This endpoint device is trusted. Users access the bulletin board over an anonymising network. This ensures that individual requests to the bulletin board cannot be linked to particular users. The connections between user devices and the bulletin board are server-side authenticated: the user device is guaranteed it is talking to the bulletin board. Moreover, confidentiality of messages between user device and bulletin board is guaranteed end-to-end (i.e. independent of the additional encryption used by the mixing network). We assume the mixing networks guarantees that any message sent is eventually delivered. The setup is depicted in figure 1.

3.3 Threat model

We assume the server is honest but curious: it will not modify the contents of the bulletin board, except when instructed by a user, but it *will* try to learn all it can by observing all accesses to the bulletin board. We do not assume anything about the behaviour of the users, except that they do not act against their own interests. The anonymising network is assumed to provide (some degree of) unlinkability between user actions and accesses observed by the server.

We do not specify the power of the adversary explicitly in terms of what it can and cannot do, as is typically done. Instead we assume that the adversary (whatever its particular capabilities) has a certain maximal probability of linking messages entering or leaving the mixing network (given a particular choice of mixing network). In what follows we show our the protocol does not give the adversary a better chance to link communicating users than this.

4. PRELIMINARIES

We use the following primitives in our protocol. First of all we use an authenticated encryption scheme [2]. Such a scheme uses a secret symmetric key $K_{AB} \in \mathcal{K}$ shared between Alice and Bob, and provides confidentiality, integrity and authenticity of the messages transmitted from Alice to Bob. In the protocol users have different keys for each party they communicate with, and keep these keys private. We write

$$\{[m]\}_{AB}$$

for the authenticated encryption of message m using K_{AB}. The inverse of this function is

$$open_{AB}(c)$$

that returns the plaintext contained in ciphertext c by decrypting it using K_{AB}. This may fail if the ciphertext has been tampered with or has been created using a different key. In that case, the function returns $\perp$. We have

$$m = open_{AB}(\{[m]\}_{AB}) .$$

We assume $\{[m]\}_{AB}$ leaks no information about m or K_{AB}, and that it provides no information about $\{[m']\}_{A'B'}$ if $m \neq m'$ or $A \neq A'$ or $B \neq B'$. In the protocol each message is encrypted using a fresh key. Hence we only require *one-time* secure Authenticated Encryption schemes [15]. These can be efficiently and generically build from (one-time secure) IND-CPA symmetric encryption and message authentication codes [2].

We also use a key derivation function $\text{KDF}(\cdot) : \mathcal{K} \mapsto \mathcal{K}$ that given a key generates a new key such that given $\text{KDF}(K)$, no information about K can efficiently be computed [13, 20]. This forms the basis for the forward security of our scheme.

We write $\|$ for the concatenation of two strings (including the additional information necessary to unambiguously determine

the point where the first string terminates and the second string starts).

5. IMPLEMENTATION

The problem with the simple protocol outlined in section 2 is caused by the fact that users are tied to specific, fixed, mailboxes. Their accesses to these mailboxes allow the server to eventually reconstruct the social graph. To avoid this problem, in the protocol below users essentially never access a particular mailbox more than once.

5.1 The bulletin board

As explained in section 3.2 and sketched in figure 1, we assume a central server is hosting a publicly accessible bulletin board $B[\,]$ of n cells, indexed by arbitrary integers modulo n. Users access the bulletin board through an anonymising mixing network. This ensures that individual requests to the bulletin board cannot be linked to particular users. Each cell of the bulletin board contains a *set* of value/tag pairs $\langle v, t\rangle$. Associated with the bulletin board is a public cryptographic hash function $\mathscr{B}(\cdot) : T \mapsto T$. $|T| > 2^{\ell}$, where ℓ is the security parameter. The hash function $\mathscr{B}(\cdot)$ together with the tag associated with a value is used to prevent malicious users from deleting values at will. Only intended recipients of messages can delete the messages addressed to them.

Users access the bulletin board by sending one of the following commands that the server implements.

add(i, v, t)**:** Add $\langle v, t\rangle$ to the set at cell i: $B[i] := B[i] \cup \{\langle v, t\rangle\}$.

get(i, b)**:** Let $t = \mathscr{B}(b)$. If $\langle v, t\rangle \in B[i]$ for some value v, return v and remove $\langle v, t\rangle$ from $B[i]$. Otherwise return $\perp$, and leave $B[i]$ unchanged.

Initially, all cells contain the empty set $\emptyset$. Note that to read a value from the bulletin board, one must know the preimage of the corresponding tag. This preimage is also used to locate the value to return among the set of values stored in the cell.

5.2 Client protocols

Again suppose Alice and Bob share a symmetric key K_{AB} for an authenticated encryption scheme. This key is used for messages sent by Alice to Bob. A different key K_{BA} is used for messages in the other direction.

The protocol is depicted in figure 2. To send a message m to Bob, Alice runs the $send_{AB}(m)$ protocol. To receive the next available message (if any) from Alice, Bob executes the $receive_{AB}()$ protocol in the same figure[6]. If no message is available, $\perp$ is returned.

The main idea of the protocol is that users use a random cell in the bulletin board for each new message they want to send. Initially (at the same time they also exchange the necessary cryptographic keys) they agree on a tag and the index of the first cell to use. (How to do this is outside the scope of the protocol, but is discussed in the application section below.) These are stored in (idx_{AB}, tag_{AB}) at both the sender Alice and the receiver Bob.

When sending a message, Alice already randomly picks a cell index and a tag for the *next* message to be sent. This index and tag are piggybacked to the *current* message intended for Bob. The index tells Bob where to look for a new message. The tag allows him to find the right one among the set of messages stored in that cell. Alice's messages are encrypted using an authenticated encryption scheme using key K_{AB}. The final message added to the set in the cell equals

$$\{[m \,\|\, idx' \,\|\, tag']\}_{AB}\,.$$

If Bob successfully received a message it is also removed from the cell used to store it to prevent the message board from overflowing. Note that if $u \neq \perp$ in the receive protocol in figure 2 then $open_{AB}(u)$ should normally be successful.

Forward security is achieved using a scheme inspired by Silent Circle's SCIMP protocol [19]. After sending a message, Alice updates her key using a key derivation function KDF$(\cdot)$. Bob does the same after successfully receiving a message. Both Alice and Bob destroy the old copies of their keys after this step. Note that we cannot use Off-The-Record (OTR) style ephemeral key exchange [4] in our protocol because the reverse channel from Bob back to Alice is unrelated to the forward channel. And in any case, because of the way messages are delivered in different cells, sender and receiver need to be synchronised anyway. This means using a key derivation function the way SCIMP does, does not impose further restrictions. We discuss synchronisation issues further in section 5.4.

5.3 Informal analysis

To see that the scheme is correct we observe the following. When Alice sends messages, she essentially creates a linked list of cells containing future messages for Bob to receive[7]. Bob receives them in the correct order by traversing this list.

Suppose $receive_{AB}()$ returns a message. This message was obtained from a cell with a message that could successfully be decrypted using K_{AB}. This guarantees the message was sent by Alice and addressed to Bob. The cell also contains the index and tag to use for the next message to be received from Alice. Bob stores these two values in (idx_{AB}, tag_{AB}).

Suppose Alice sends Bob a message. Then the index idx_{AB} of the cell to store the message in was sent to Bob in the previous message, while the tag value associated with that cell equals $\mathscr{B}(tag_{AB})$. The preimage tag_{AB} was also sent to Bob in the previous message. This allows Bob to retrieve the message (and remove it from the cell).

Both the send and the receive protocol accesses the bulletin board exactly once. The send protocol clearly takes constant time to complete. The receiver knows the index of the cell of the bulletin board to inspect for the next incoming message. The time complexity of the receive protocol (or rather the get operation implemented by the bulletin board) therefore equals the expected number of entries in this cell. This depends on the size of the bulletin board n and the number of messages c it contains. As the index of a cell to use for a message is selected at random, the expected size of a cell equals c/n (see Lemma 6.1). If there is a message waiting to be received, the expected size of the cell becomes $1+(c-1)/n$. (Typically the service provider will ensure that t never becomes larger than n, which means that in most cases, any cell either contains no entries, or just a single one. This is further discussed in Section 5.5.)

To see the scheme satisfies the security requirements from section 3.1, we observe the following. All messages are encrypted using an authenticated encryption scheme, with different keys for each connected pair of users. This assures confidentiality, authenticity and integrity of the messages transmitted. Unlinkability follows from the random selection of the next cell to be

[6] The AB in the subscript denotes that the receive function is used to receive a message from A to B.

[7] This list is embedded in the array of cells representing the bulletin board. The indices of the bulletin board serve as list pointers.

function $send_{AB}(m)$
$idx' \in_R \{0,\ldots,n-1\}$
$tag' \in_R T$
$u := \{[m \,\|\, idx' \,\|\, tag']\}_{AB}$
write$(idx_{AB}, u, \mathscr{B}(tag_{AB}))$
$(idx_{AB}, tag_{AB}) := (idx', tag')$
$K_{AB} := \mathrm{KDF}(K_{AB})$

function $receive_{AB}()$
$u :=$ **get**(idx_{AB}, tag_{AB})
if $u \neq \perp$
$\wedge\ (m \,\|\, idx' \,\|\, tag') := open_{AB}(u)$ is successful
then $(idx_{AB}, tag_{AB}) := (idx', tag')$
$K_{AB} := \mathrm{KDF}(K_{AB})$
return m
else return $\perp$

Figure 2: Protocols to send or receive a message.

used to transmit a message, and the fact clients connect to the bulletin board over a mixing network. Consider any two different events that are not a send and the corresponding receive, or two different tries to receive the same message. These use totally unrelated cells because of the random selection process. The encryption used in the transmission of the next index to use ensures that the adversary cannot predict the next index in advance.

5.4 Keeping clients synchronised

One issue with the approach taken by the protocol is that users need to remember indices, tags and keys. What if these get corrupted?

If any key is corrupted, this means that messages will no longer be correctly decrypted or verified by the receiver. As a result messages will fail to get across. If the sender continues to send new messages eventually the bulletin board will get full. Note that only receivers are able to tell that their keys are corrupted, when they fail to decrypt a message that is stored in a cell they successfully accessed.

Indices and tags can also get corrupted. If the index or the tag of a sender gets corrupted, the write will store the message in the wrong cell. If the index or the tag of a receiver gets corrupted, the get will fail because with overwhelming probability there is no value/tag pair at the given index that matches the supplied tag. For the read, this will simply look as if no message is currently waiting for him. We conclude that the protocol itself allows neither a sender or a receiver to determine whether their index and tag values have been corrupted. This can be overcome by adding redundancy to the local state of each user client. One possibility is to store a hash of the state along with the state itself, and verify this hash at the start of each send or receive.

This allows users to at least discover that something is wrong. The question remains how to resolve the issue. The obvious solution is to allow clients to encrypt their state (keys, index and tag for each of their communicating parties) against some long term static key (with an offline copy) and store this somewhere. However this idea does increase the risk of linkability: every time a user performs an action his state needs to be updated, and these events can be correlated. Using the mixing network to also obfuscate this storage of the state partially resolves this, but still gives the adversary additional information for the potential identity of the sender or the receiver of a particular message. This increases his chances.

5.5 Scalability

Another engineering issue is scalability. If the number of users is small, a single server maintaining a small bulletin board will suffice. As the number of users grows, or the number of messages sent increases, bandwidth and storage requirements increase as well. The question is whether the design allows us to dynamically accommodate for such changes.

If the load on the server gets too high, the bulletin board can be partitioned into chunks that are each served by a different server. The partitioning is communicated to the users, which allows all read and write operations to be directed at the appropriate server depending on the index of the cell to be accessed. One issue to be aware of, is that this allows the adversary to distinguish which of the servers is being accessed. In an extreme case, if the anonymity set of Alice's first write access to the new server is small, and so is the anonymity set of Bob's first get from the new server, then the adversary has a better than usual chance to determine whether Alice and Bob are acquainted.

To change the size of the bulletin board a new bulletin board with the appropriate size has to be created, and all send function calls will be instructed to direct their locks and writes to the new bulletin board. At the same point in time, all receives are instructed to look for any remaining (old) messages on the old board once, and to start looking for messages on the new board from that point onward. Once the old bulletin board only contains empty cells, it can be discarded. This can be monitored by letting the bulletin board keep track of the difference δ between successful write and get calls, as this in fact equals the number of messages c the bulletin board currently contains. Because both bulletin boards are served by the same server, an external adversary cannot tell the difference and use this information to increase his chances to trace a user.

This difference can also be used to determine whether the bulletin board size should be increased or decreased. A bulletin board with very few occupied cells wastes storage space and should be decreased. A bulletin board with many occupied cells will make reads become slower (as the expected size of the set of messages stored in a cell increases).

5.6 Availability

The protocol as it stands assumes honest users and is highly susceptible to denial-of-service attacks. Although deletion of entries from cells is prevented by requiring the knowledge of the preimage of the corresponding tag value[8], anybody can write to a cell. This allows the adversary to completely fill the bulletin board essentially blocking new writes.

A natural approach to limit the damage an adversary can do is to a associate a certain cost with sending a message. The cost could be monetary (a small payment for each write) or involve other resources (like solving a complex puzzle [1]). Alternatively

[8]The preimage of the tag value cannot be obtained by an adversary, even if Bob is using it to check for a new message that has not been sent yet, because Bob uses an authenticated and encrypted channel to communicate with the bulletin board server.

users can register with the service, at which point they receive a few anonymous send-coins, that need to be spent when sending a message. Users that receive a message (and hence free up space on the bulletin board) also receive a new send-coin. This approach limits the number of messages pending between A and B. None of these approaches completely solve the problem though.

Unfortunately any more fundamental solution to the problem does not appear to exist, in essence because an adversary can always spawn an overwhelming number of users that individually behave within the limits deemed acceptable, while collectively bringing the system down to its knees.

6. PROOFS

We now proceed to prove the protocol in Figure 2 correct and secure. In the lemma's below we are slightly more precise about corruptions of nodes, compared to the statement of the requirements in section 3.1.

Lemma 6.1 *If the bulletin board has size n and contains c messages in total, then the expected number of elements stored in a cell is c/n. If there is a message waiting to be received, the expected size of the set becomes $1+(c-1)/n$.*

PROOF. The cell to store a message in is selected uniformly at random from a total of n locations. Hence the probability p that a particular message is stored at a particular cell i equals $1/n$. When storing c messages like this in the bulletin board, the number of messages X in a particular cell follows the binomial distribution, with

$$\mathbf{Pr}[X=x]=\binom{c}{x}p^x(1-p)^{t-x}$$

The expected value

$$\mathbf{E}[X]=\sum_{x=0}^{t}x\mathbf{Pr}[X=x]$$

equals $c\times p=c/n$ [14].

If there is a message waiting to be received, i.e. if there is certainly one message stored in that cell, then the expected size of the set is determined by distributing the remaining $c-1$ messages over the n cells. This gives $1+(c-1)/n$. □

Let us focus on the case where Alice is sending messages to Bob. The *state* of a Alice (or Bob) is the value of K_{AB}, idx_{AB}, and tag_{AB}.

Assumption 6.2 *Alice's state for sending messages to Bob is independent of her state for sending messages to any other user.*

In what follows we simplify notation and omit mentioning the label $_{AB}$ explicitly. Let us write $m_{A,i}$ for the i-the message sent by Alice (to Bob). Let us write $m_{B,i}$ for the i-the message received by Bob (from Alice). Let us write $\sigma_{A,i}$ for the state of Alice in which she starts sending message $m_{A,i}$. Similarly let $\sigma_{B,i}$ be the state of B in which he starts receiving $m_{B,i}$. Let $K_{A,i}$ be the value of K_{AB} in $\sigma_{A,i}$. Define $idx_{A,i}$, $tag_{A,i}$, $K_{B,i}$, $idx_{B,i}$ and $tag_{B,i}$ similarly.

We assume the following about the initial exchange of keys and other data between two users that execute $setup_{AB}$.

Assumption 6.3 $\sigma_{A,1}=\sigma_{B,1}$. *Initially, only Alice and Bob know the values in these states.*

Proposition 6.4 *In the absence of any adversarial activity, for all $i>1$, $m_{A,i-1}=m_{B,i-1}$ and $\sigma_{A,i}=\sigma_{B,i}$ with overwhelming probability (provided both states are reached).*

PROOF. By induction on i. $\sigma_{A,1}=\sigma_{B,1}$ by assumption 6.3. Now suppose $\sigma_{A,j}=\sigma_{B,j}$, we will prove the proposition holds for $i=j+1$. We assume $\sigma_{A,j+1}$ and $\sigma_{B,j+1}$ are both reached. To reach $\sigma_{A,j+1}$, Alice successfully sent $m_{A,j}$. This means she actually sent $u=\{[m_{A,j}\,\|\,idx_{A,j+1}\,\|\,tag_{A,j+1}]\}_{A,j}$ using **write**$(idx_{A,j},u,\mathscr{B}(tag_{A,j}))$. To reach $\sigma_{B,j+1}$, Bob successfully received a message when started in state $\sigma_{B,j}$. Bob's call to **get**$(idx_{B,j},tag_{B,j})$ returns with overwhelming probability $\{[m_{A,j}\,\|\,idx_{A,j+1}\,\|\,tag_{A,j+1}]\}_{A,j}$. (With probability roughly $2^{-\ell}$ there is a collision such that another tuple in the set stored at cell $idx_{B,j}$ contains the same tag.) Bob then sets $m_{B,j}=m_{A,j}$, $idx_{AB}=idx_{A,j+1}$ and $tag_{AB}=tag_{A,j+1}$ as required. Alice sets $K_{A,j+1}:=\mathrm{KDF}(K_{A,j})$ and Bob sets $K_{B,j+1}:=\mathrm{KDF}(K_{B,j})$. If $K_{A,j}=K_{B,j}$ then $K_{A,j+1}=K_{B,j+1}$ □

The following lemma is an easy corollary of this proposition.

Lemma 6.5 (Correctness) *Suppose Alice successfully sends a message m to Bob. Then Bob eventually receives m unaltered. Moreover, Bob receives its messages in the same order as Alice sends them.*

Proposition 6.6 *For all $i>0$, unless Alice or Bob gets corrupted at a state $c<i$, it is guaranteed that only Alice and Bob know $\sigma_{A,i}$, with the following two exceptions. The bulletin board learns $idx_{A,i}$ as soon as Alice writes $m_{A,i}$. The bulletin board learns $tag_{B,i}$ as soon as Bob tries to get $m_{B,i}$.*

PROOF. By induction on i. The proposition holds initially by assumption 6.3. Suppose the proposition holds for $i=j$. We show it also holds for $i=j+1$. If Alice or Bob are corrupted at state j the proposition trivially holds. If Alice is corrupted at state $j+1$, then by construction she has erased $idx_{A,j}$ and $tag_{A,j}$ and has set $K_{A,j+1}:=\mathrm{KDF}(K_{A,j})$. From the latter value $K_{A,j}$ cannot be reconstructed (see Section 4). The same holds for Bob. $K_{A,j}$ is never revealed in the normal course of the protocol. The two exceptions follow from the way Alice and Bob send and receive messages in state j. □

Lemma 6.7 (Confidentiality) *Suppose Alice successfully sends a message m to Bob before being corrupted. Then nobody else (except Alice and Bob) learns m, provided Bob receives it before being corrupted.*

PROOF. Follows from proposition 6.6 and the fact that Alice encrypts m as $\{[m\,\|\,idx'\,\|\,tag']\}_{AB}$ using key K_{AB} before sending it to the bulletin board. □

We note that as long as Bob has not received the message, there is a danger that Bob is compromised giving the adversary access to his current and all future keys.

Lemma 6.8 (Integrity) *If Bob receives a message m while not being corrupted, m was sent to Bob (by one of his contacts), provided this contact was not corrupted before sending the message.*

PROOF. Follows from proposition 6.6 and the fact that if Bob receives m, it got from the bulletin board $\{[m\,\|\,idx'\,\|\,tag']\}_{AB}$ using key K_{AB} for some contact A. □

Lemma 6.9 (Authenticity) *Suppose Alice successfully sends a message m to Bob before being corrupted. Then Alice is guaranteed that only Bob receives m, and Bob is guaranteed that Alice was the sender of m (if he receives m, before being corrupted).*

PROOF. Similar to the proof of Lemma 6.8. □

Unlinkability of events was defined in section 3.1 as "The adversary cannot link any two send and/or receive events, *except* the events that happen at the bulletin board that receives a single message with the corresponding event that sent that message, or any sequence of unsuccessful receives of the next expected message from a particular user and the final successful receipt of that message." We make that more precise as follows.

We assume some upper bound on the power of the adversary in obtaining information by observing or interacting with the mixing network. Instead of strictly specifying what the adversary can or cannot do, we assume we are given an upper bound on the probability that the adversary correctly links two events he observes in the mixing network.

Assumption 6.10 *The adversary is able to guess with probability at most p whether any message entering the mixing network corresponds to some other message leaving the mixing network.*

This allows us to show that the adversary gains no additional information from the protocol, in the sense that it does not matter whether Alice and Bob would communicate directly through the mixing network, or us our protocol to do so. Of course, using our protocol allows Alice and Bob to communicate asynchronously. This is impossible when communicating through a mixing network directly.

Lemma 6.11 (Unlinkability of events) *Suppose Alice successfully sends a message m to Bob, and that Bob queries the bulletin board k times until receiving the message. Then the adversary is able to guess with probability at most p that Alice sent this message to Bob.*

PROOF. Alice uses a random location on the bulletin board for each new message she sends. This ensures that write events on the bulletin board are completely independent. By assumption $\{[m]\}_{AB}$ leaks no information about m or K_{AB}, or information about $\{[m']\}_{A'B'}$ if $m \neq m'$ or $A \neq A'$ or $B \neq B'$. Hence these write events provide no useful information to the adversary.

Suppose Alice sends a message to Bob. This involves sending one message from Alice to the bulletin board over the mixing network for the call to the **write**(idx_{AB}, u, b) function. Then the adversary has probability at most p to guess correctly that the message sent by Alice is stored at index idx_{AB} on the bulletin board. (Note that this requires cooperation of the bulletin board server, because the index is protected by the secure channel from Alice to the bulletin board.)

Suppose Bob queries the bulletin board k times until receiving the message. Each query involves sending one message from Bob to the bulletin board over the mixing network for the call to the **get**(idx_{AB}, t) function, and another return message from the bulletin board back to Bob over the mixing network to return the result. In total then the adversary sees $2k$ messages to or from index idx_{AB}, and for each message he has probability p to guess Bob is involved in them.

How these $2k$ observations of the adversary determine the probability he guesses correctly that Bob is the recipient, very much depends on the particular observations. On the one hand, if the adversary observes in all $2k$ cases the same $1/p$ users each having probability p to be accessing idx_{AB}, then the probability remains p. On the other hand, if Bob is the only user in the intersection of all observations, then $p = 1$.

However, the uncertainty about Alice being the sender remains p as argued in the beginning of the proof, which concludes the proof. □

Lemma 6.12 (Unlinkability of contacts) *Suppose Alice has Bob as contact, and Charlie has Donna as contact. Then Alice and Charlie cannot determine whether Bob and Donna are the same person or not.*

PROOF. This follows from the fact that independent of whether Bob and Donna are the same person or not, the key, index and tag Alice uses to communicate with Bob is chosen at random. The same values would have been used in both cases. □

Note that Alice and Charlie can always try to determine whether Bob and Donna are the same person by sending particular messages to their contacts and compare their responses. If both talk about the same event in a very similar way, or even by simply comparing their writing style, a link can be deduced.

Lemma 6.13 (Forward security) *If a user device is compromised, the adversary is only able to obtain the contents of (or tamper with) future messages to or from this user. In particular, the adversary is unable to trace the contacts of a compromised user.*

PROOF. Let Alice be compromised in state i. From the fact that message $m_{A,j}$ is protected using $\{[m_{A,j} \,\|\, idx' \,\|\, tag']\}_{AB}$ using key $K_{A,j}$, using proposition 6.6 we conclude that $m_{A,j}$ is safe for all $j < i$. The same holds for Bob.

Because both senders and receivers each use a mixing network to connect to the bulletin board, they cannot be traced even when the adversary, the other user, and the bulletin board collaborate. □

As discussed in section 5.6 our system only guarantees partial availability.

Lemma 6.14 ((Partial) Availability) *A message sent will eventually be delivered.*

PROOF. Suppose $\langle v, t \rangle$ is the value/tag record stored in the bulletin board at location i that corresponds to a message sent by Alice but that never is received by Bob. A sent message will fail to be delivered if it is deleted before the intended recipient asks for its delivery. We assume the bulletin board server itself is honest but curious, so deletion only happens if the bulletin board receives a **get**(i, b) command with $t = \mathcal{B}(b)$. Bob receives b from Alice through an encrypted message that can only be decrypted using key K_{AB} known to only Alice and Bob. (We note that Alice updates K_{AB} and destroys b right after she sends the messages, so even Alice or an adversary compromising her afterwards cannot recover b.) We conclude that only Bob eventually (after receiving the preceding message) learns b. Once Bob starts asking for the message, the bulletin board also learns b of course, but this is not an issue. Because communication between Bob and the bulletin board is protected end-to-end using an authenticated and encrypted channel, no other party can learn b. We conclude that only Bob himself can send the **get**(i, b) command with $t = \mathcal{B}(b)$ to delete the message. But then by definition Bob received the message. □

7. APPLICATIONS

7.1 Anonymous and secure messaging

Clearly a unidirectional asynchronous message transmission protocol can be used to implement a bidirectional asynchronous message exchange protocol, by running two instances of the protocol in parallel: one for messages to be sent from Alice to Bob,

and one for messages to be sent from Bob to Alice. These instances run completely independent of each other, and do not share state or keys.

To be able to use the message exchange protocol Alice and Bob need to exchange keys, tags and indices for both unidirectional channels in private. One intuitive way to do so is the following. Assume both Alice and Bob have the app implementing the protocol installed on their smart phone. By bumping their phone, a key exchange protocol is triggered. The protocol could use any short range transmission protocol to ensure that no man in the middle could physically be interfering with the exchange. Alternatively, Alice and Bob need to verify fingerprints over the exchanged keys on their own devices.

The fact that communication is bidirectional may help overcome synchronisation issues in the underlying unidirectional transmission protocol (as described in Section 5.4). Recall that clients can tell whether their state is corrupted or not, by maintaining some redundancy (e.g. a hash over the state) together with their state. Now if corruptions of the sender and receiver part of the state of a bidirectional messaging client are independent (and this is a *big* if), then one channel can be used to inform the party at the other end of this corruption. Both users can then use out-of-band means to resynchronise.

7.2 An efficient presence protocol

Our primitive can also be used to efficiently implement a privacy friendly presence protocol, similar to DP5 [3] (discussed in more detail below). Such a presence protocol allows Alice to inform her contacts whether she is on line or not, a function found on many popular chat, messaging and VoIP systems. A privacy friendly presence protocol hides the social graph from both an external adversary as well as the service provider itself.

The idea is to let Alice use our point-to-point message transmission protocol to inform Bob of her current status, by simply sending him an update using our protocol whenever her status changes. The message complexity is constant in this case. The privacy guarantees provided by this approach are much lower than that offered by DP5 however: if only Alice and Bob are using the system, and no one else is using the underlying mixing network, their relationship is easily exposed. On the other hand, if there simultaneously many other users using the mixing network for other purposes, this does also give Alice and Bob strong privacy guarantees. Perhaps even stronger than provided by DP5 if the user base is very small. This is the risk of a dedicated system to provide anonymity.

This simple approach has two drawbacks. First of all, Alice needs to inform all of her friends of a status change separately. Moreover, the approach taken informs Bob of *all* of Alice's past status changes, even if he was offline for a long time. He can, for example, tell that Alice was online several times last night even he himself was vast asleep... This is perhaps counter intuitive but hard to strongly protect against in practice, because even a system like DP5 cannot prevent a malicious 'friend' to keep the application alive and log all status changes.

Yet, Alice can prevent that this information is leaked to honest users if we slightly tweak the protocol. The idea is to update the cell in which Alice's status is kept if Bob has not read that cell yet. This is possible because Alice uses independent messages to each of her friends to inform them of her status change. To implement this idea, the bulletin board needs to offer an update instead of a write primitive that swaps old values with new values if possible.

To remove the overhead of having to informing each friend separately, a broadcast like protocol needs to be developed. Let Alice's friends be a fixed group of users. The essential idea is that all of them initially share the same state K_{AB}, idx_{AB}, and tag_{AB}. Each tuple stored on the bulletin board also gets a reference counter, indicating how many of the friends have received the associated message. The bulletin board only deletes the tuple if the reference counter becomes zero. Alice notifies a change in state by writing the new state as before, but now including the number of friends as the initial value for the reference counter. A drawback of this approach is that it leaks the number of friends Alice has to the bulletin board. We note that this idea is incompatible with the idea presented in the previous paragraph. In other words, there is not yet an efficient protocol that prevents a friend of Alice to learn all her state changes.

8. RELATED WORK

Several other papers have recently been published that address the protection of metadata in private messaging protocols.

Dissent [22, 9] is based on DC-nets [6]. It distinguishes between clients and servers to increase efficiency, under the assumption that at least one of the servers is trusted. The communication costs is not negligible however: it depends on both the number of users and the number of servers. In 2012 the system supported anonymity sets of up to 5.000 users. The system is synchronous, operates in rounds, and assumes all communicating users are online. Our protocol is asynchronous, and assumes no trusted servers.

Riposte [8] is a system for anonymous broadcasting, and is using ideas from Private Information Retrieval (PIR) [7] in 'reverse' (i.e. to hide the location and the value being written). Like our protocol it uses a shared bulletin board, but in Riposte's case the bulletin board is maintained by a small set of servers at least one of which is assumed to be trusted. Senders need to submit $O(\sqrt{n})$ size encoded shares to each of the servers, where n is the size of the bulletin board. To read a message, users need to collect the shares at a particular index from all servers and combine the shares to retrieve the message. The system is asynchronous, like ours, but assumes some trust in servers and is less efficient.

Independent of our work, Van den Hooff *et al.* [21] recently proposed 'Vuvuzela', a private messaging system. The techniques they use are similar to ours, however their system is synchronous (operating in rounds), bidirectional, and assumes all senders and receivers are online. These assumptions do allow them to achieve stronger privacy guarantees, for example by introducing cover traffic.

Ricochet[9] is an anonymous instant messaging system built using Tor hidden services, similar to the idea explained in Section 2. Ricochet however assumes a P2P model of communication where each user hosts his own hidden service. This makes it less easy to deploy in practice. Also users need to be online to receive messages.

Another interesting approach to hide the social graph in online communication protocols is used in DP5 [3], that uses private information retrieval (PIR) [7] as a building block to implement a private presence service. Such a service allows Alice to inform Bob whether she is online, without revealing the fact that Alice and Bob are even connected to an external observer. The privacy guarantee given by DP5 is strong: even if only Alice and Bob are communicating, an observer does not learn whether Alice and Bob are connected. This privacy comes at a cost however. Using a particularly efficient instantiation of PIR [16], the message

[9] https://ricochet.im/

complexity of their protocol is roughly $O(\sqrt{ns})$, where n is the number of users and s the (fixed) message size.

9. DISCUSSION

From the analysis is may seem that the second mixing network (between the bulletin board and the recipient Bob) is not necessary. We note however that omitting the second mix first and foremost leaks (to the bulletin board) the fact that Bob is using the service, leaks how many messages he is receiving, and when he prefers to receive them. Moreover, the second mix is important to protect Bob's identity whenever Alice becomes compromised and the adversary tries to identify her contacts (with the help of the bulletin board service, who may be legally compelled to cooperate).

Lemma 6.11 and its proof are unsatisfactory. The proof does not use the fact that the receiver is connected to the bulletin board through a mixing network at all. Intuitively it seems logical that adding the mixing network there increases the protection offered by the system overall. But this is hard to quantify, especially because it is hard to model the privacy protection offered by a mixing network in a way that helps in our analysis here. We are aware of the work by Diaz *et al.* [12] for example, but believe additional research would be beneficial in this area.

To increase the level of privacy protection offered it would be great if there was a way to make it impossible for an external adversary to distinguish senders and receivers. Currently they can easily be told apart because senders use one call to a write function that does not return a value, whereas a receiver uses a call to a get function that does return a value. Obviously, they could be made indistinguishable by letting the write function return an arbitrary value. This does however break the proof of Lemma 6.11. By returning a value, you give the adversary more chances to trace the sender.

Our protocol is susceptible to the following active attack. If an adversary suspects that Alice and Bob are communicating, it can block all traffic from other users. If the bulletin board server cooperates, they can observe the access patterns on the cells of the bulletin board to see whether the same cell is accessed by both a write and a get operation. If that is the case, Alice and Bob are indeed communicating. For this attack to work the (honest but curious) bulletin board must cooperate with the external adversary blocking all traffic. A possible countermeasure would be to let clients detect whether they are blocked, and inform all other clients of this fact.

10. CONCLUSIONS

We have shown that it is possible to construct a secure and privacy friendly asynchronous unidirectional point-to-point message transmission protocol using a public bulletin board. Our construction is built on top of an arbitrary mixing network, and provides the same privacy guarantees as this underlying mixing network. However, our construction allows messages to be transmitted asynchronously (which is impossible with a bare mixing network). So another way to interpret our results is to say that we show how to use mix networks asynchronously.

Our construction is efficient and scalable. The public bulletin board can be hosted 'in the cloud' without any security or privacy consequences.

We thank AVG[10], and in particular Jessica, Carolien and Maurice, for organising and hosting a design sprint in March 2015 for an anonymous messaging app (Burnrchat), in which the ideas presented in this paper were developed. Jeroen de Knijf is especially thanked for contributing to the initial ideas explored in this paper. We thank David Lazar, the members of the CYRCLE[11] group and in particular Tommy Koens for discussions on earlier ideas and drafts of this paper.

[10] http://www.avg.com

[11] https://cryptocyrcle.wordpress.com/

11. REFERENCES

[1] Adam Back. Hashcash - a denial of service counter-measure. http://www.cypherspace.org/hashcash, March 1997.

[2] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *J. Cryptology*, 21(4):469–491, 2008.

[3] Nikita Borisov, George Danezis, and Ian Goldberg. DP5: A private presence service. In *Proceedings on Privacy Enhancing Technologies 2015*, 2015.

[4] Nikita Borisov, Ian Goldberg, and Eric A. Brewer. Off-the-record communication, or, why not to use PGP. In Vijay Atluri, Paul F. Syverson, and Sabrina De Capitani di Vimercati, editors, *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society, WPES 2004, Washington, DC, USA, October 28, 2004*, pages 77–84. ACM, 2004.

[5] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

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

[7] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *36th Annual Symposium on Foundations of Computer Science, Milwaukee, Wisconsin, 23-25 October 1995*, pages 41–50. IEEE Computer Society, 1995.

[8] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. Riposte: An anonymous messaging system handling millions of users. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 321–338. IEEE Computer Society, 2015.

[9] Henry Corrigan-Gibbs and Bryan Ford. Dissent: accountable anonymous group messaging. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 340–350. ACM, 2010.

[10] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type III anonymous remailer protocol. In *2003 IEEE Symposium on Security and Privacy (S&P 2003), 11-14 May 2003, Berkeley, CA, USA*, pages 2–15. IEEE Computer Society, 2003.

[11] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009, Oakland, California, USA*, pages 269–282. IEEE Computer Society, 2009.

[12] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In Roger Dingledine and Paul F. Syverson, editors, *Privacy*

Enhancing Technologies, Second International Workshop, PET 2002, San Francisco, CA, USA, April 14-15, 2002, Revised Papers, volume 2482 of *Lecture Notes in Computer Science*, pages 54–68. Springer, 2002.

[13] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the cbc, cascade and HMAC modes. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2004.

[14] William Feller. *An Introduction to Probability Theory and Its Applications*. Wiley & Sons, New York, 2nd edition, 1957.

[15] David Galindo and Jaap-Henk Hoepman. Non-interactive distributed encryption: A new primitive for revocable privacy. In *Workshop on Privacy in the Electronic Society (WPES)*, pages 81–92, Chicago, IL, USA, October 17 2011.

[16] Ian Goldberg. Improving the robustness of private information retrieval. In *2007 IEEE Symposium on Security and Privacy (S&P 2007), 20-23 May 2007, Oakland, California, USA*, pages 131–148. IEEE Computer Society, 2007.

[17] Sanja Kelly, Madeline Earp, Laura Reed, Adrian Shahbaz, and Mai Truong. Tightening the net: Governments expand online controls. Technical report, Freedom House, 2014. Freedom on the Net 2014 report.

[18] Alan Mislove, Bimal Viswanath, P. Krishna Gummadi, and Peter Druschel. You are who you know: inferring user profiles in online social networks. In Brian D. Davison, Torsten Suel, Nick Craswell, and Bing Liu, editors, *Proceedings of the Third International Conference on Web Search and Web Data Mining, WSDM 2010, New York, NY, USA, February 4-6, 2010*, pages 251–260. ACM, 2010.

[19] Vinnie Moscaritolo, Gary Belvin, and Phil Zimmermann. Silent circle instant messaging protocol. protocol specification. Technical report, Silent Circle, December 5 2012. Version 1.0.

[20] NIST 800-108. Nist special publication 800-108 recommendation for key derivation using pseudorandom functions (revised). Technical Report NIST Special Publication 800-108, National Institute of Standards and Technology, U.S. Department of Commerce, October 2009.

[21] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *HotPETs 2015*, Philadelphia, PA, USA, July 2 2015. (full version to appear).

[22] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In Chandu Thekkath and Amin Vahdat, editors, *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 179–182. USENIX Association, 2012.