ReClaim: a Privacy-Preserving Decentralized Social Network

Niels Zeilemaker Johan Pouwelse
Delft University of Technology, The Netherlands
niels@zeilemaker.nl

Abstract

The privacy concerns associated with the popularity of online social networks have given rise to numerous research papers which focus on either storing the data in encrypted form, or decentralizing the network. However, without the incentive of being able to read private data, a centralized system cannot be sustained. A decentralized network does not rely on such incentives and can continue to operate based solely on user contributed resources. Unfortunately, current proposals either still rely on centralized components, a pre-existing friend-to-friend network, or incur substantial bandwidth overhead. This renders them useless for actual deployment.

ReClaim employs an existing technique to the PSI problem to build a semantic network wherein peers have connections to friends and friends-of-friends. This is achieved by a handshake which uses homomorphic encryption to privately determine which friends are shared amongst two peers. Afterwards, peers synchronize messages destined for friends, providing them with easy access to replicas. Moreover, storing those messages allows ReClaim to handle excessive churn (peers joining and leaving the network at a high rate), overcome NAT-firewalls (by allowing for indirect communication), and allow friends to communicate without being online at the same time.

After describing the design of ReClaim, we implement a proof-of-concept application and show during extensive emulation that peers can connect to more than 60% of their friends within 10 minutes in a network consisting of 4000 peers. Moreover, within 10 minutes, peers using ReClaim can establish an indirect connection to 95% of their friends.

1 Introduction

Online social networks (OSNs) have become hugely popular over the last ten years. Facebook, with more than 1.23bn monthly users, is the largest. Its network consists of more than 200bn friendships, 400bn photographs, and generates more than 6bn "likes" every day[1]. However, the success of Facebook has also given rise to some substantial privacy concerns, as a single company is now responsible for managing a wealth of privacy sensitive data.

Although Facebook is committed to protect our privacy[2] it can be questioned if such a big collection of data *can* be secured. Without questioning the level of expertise of Facebook's employees, the Lavabit case has revealed that any company can be asked to hand over its private keys in order to comply with a court order, thereby exposing all data stored in its cloud.

Centralized storage can therefore be considered compromised, even if stored in encrypted form using a key managed by the storage provider. Therefore, managing keys locally is a more sensible solution. However, as social networks depend on being able to read your private data, there is a strong incentive for them against implementing this. Moreover, freeriding on top of an existing social network and using it to send your friends pointers to encrypted data stored elsewhere is likely to be actively blocked and although your data will be protected in this case, the social network graph is still accessible.

In this paper we present a fully decentralized solution for online social networks which does not depend on any centralized component. We use friends of friends to replicate data, and therefore do not need any server for storage. The resulting running costs are negligible, as there are no costs associated to storing data. Friendships are established by exchanging public keys out-of-band, allowing all messages to be encrypted, making them unreadable for everyone but the user who has the private key.

Discovering friends which are currently online is achieved by employing an existing approach to the

[1] `http://blogs.ft.com/tech-blog/2014/02/facebook-turns-ten-likes-and-lows`

[2] `https://www.facebook.com/notes/10150378701937131`

Private-Set-Intersection (PSI) problem. This allows two peers to discover which friends they have in common without disclosing those they do not. After discovering common friends, peers synchronize missing messages destined for those friends using Bloom filters (compact hash representations of their local message database). Bloom filters allow peers to efficiently receive all messages destined for themselves and their friends, as they prevent duplicate message transfers.

Contributions

1. The design and implementation of a protocol which allows peers to construct a decentralized social network.
2. The means of finding friends which are online without exposing all friends of a peer.
3. A method to synchronize with and store messages at friends, allowing the network to operate without centralized storage servers.
4. An extensive evaluation, testing the speed with which friends can be found.

2 Related Work

Sun et al. [10] describe an efficient revocation mechanism for a private OSN. Their approach is based on *broadcast encryption*. Broadcast encryption is a technique in which you encrypt a file/message once with a symmetric key, and let a subset of privileged users decrypt this symmetric key with their private key. Hence, a file can be stored on a central server once, but can be decrypted by multiple users. Revoking the privileges of a single user does not require any messages to be send, it will only cause the next encrypted message to not be decryptable by the revoked user.

Additionally, Sun et al. discuss how to query for encrypted files by employing *public key encryption with keyword search* (PEKS). PEKS allows a storage site, which is only storing encrypted files, to return only those files matching a query. The storage site cannot determine content of the query or the contents of the returned files. A secure index is used to allow for this, which, although a costly to build enables storage sites to efficiently respond to such queries.

Beato et al. [1] describe a browser plugin which employs broadcast encryption to be able to post and retrieve content stored in encrypted form at a central server. In contrast to Sun et al., their solution (Scramble) uses existing social networks to send links to encrypted content and hence does not require a secure index. Scramble encrypts/decrypts content in a transparent manner, hiding the complexity from its end users, e.g. users of Scamble do not see any difference between encrypted and non-encrypted files.

Safebook is a *decentralized online social network* (DOSN) [3] based around the concept of matryoshkas. Using the trust relationships of the social graph, matroyshkas relay messages from a peer in their center. The matryoshkas directly connected to the peer they are protecting are called mirrors and store encrypted data destined for the peer. The outermost matryoshkas are called entrypoints, and register themselves in a DHT as the innermost peer. By varying the pathlength between the inner and outer matryoshkas, the actual identity/location of the peer is hidden from the DHT.

Cachet [8] is also built around a DHT. However, it does not incorporate matryoshkas, but uses the DHT as a resilient data store. Users of Cachet store their latest update in the DHT, and point towards this update on their wall. All updates are encrypted using attribute-based encryption, which allows users to target groups of users with each update. Additionally, as a performance enhancement, friends are used to cache decrypted updates in order to reduce the number of lookups in the DHT/decryptions.

Mega et al. [7] describe how they envision efficient dissemination of messages in a DOSN. Instead of relying on a DHT, they build a friend-to-friend network in which messages are distributed using rumor mongering. Herein, peers proactively push a message to others while it is hot. After getting one or more replies from peers that they already received this rumor, it goes cold and a peer stops forwarding it. By assuming a prebuilt F2F network and incorporating history in the rumors, they claim that their protocol significantly improves over the mainstream gossip protocols and direct mailing.

3 Goals and Security model

In this paper we design, implement, and evaluate a decentralized online social network which does not rely on a central server. We employ an existing approach to the PSI problem to locate friends and friends-of-friends which are online in a privacy preserving manner. This results in a system that does not rely on any existing network, and one which runs without any centralized component/oversight.

Messages between peers are synchronized in a push-pull manner, and can contain wallposts, likes, images, etc. After creating a new message, it is pushed to all connected peers which have the destination peer in common. Older or missed messages are synchronized using Bloom filters, allowing peers to request messages which were sent when it was offline.

All peers in our DOSN, named ReClaim, store encrypted messages destined for their friends. This allows ReClaim to continue to operate reliably in the presence of NAT-firewalls, intermittent network connectivity, and

significant link delays.

In contrast to related work, we do not employ broadcast encryption. While broadcast encryption will reduce both bandwidth and storage costs, it requires us to either annotate each encrypted message with the destination peers, or implement a secure indexing scheme. The former will expose the social graph to our friends (something we want to prevent), the latter will require peers to build and maintain costly secure indexes. Moreover, the use of attribute based encryption, as in Cachet, is non-trivial. Managing the access policies of cached objects in a decentralized setting is hard, and the latency introduced by the ABDecryption [8] could only be overcome in Cachet by adding unencrypted social caching.

Therefore, in order to simplify our system and the resulting protocol, a ReClaim peer creates an encrypted message for each peer it wants to send an post to.

We assume a semi-honest setting [4] also known as honest but curious. This setting assumes that peers do not deviate from the protocol, but might attempt to analyze messages in order to infer additional information. In this setting, ReClaim will protect the social graph from peers which do not have any friends in common, i.e., any two peers which do not share friends will never be able to determine the friends of the other. Moreover, such peers will never receive an encrypted message destined for the other.

Peers which do have overlap in friends are able to detect which friends overlap, and hence construct a partial social graph. However, if these two peers are not friends themselves they will not be able to discover their true identities, as these are kept hidden. Finally, the two peers will still never receive a message destined for the other, as peers will only be able to receive messages destined for their friends.

Peers which are friends will receive and store messages destined for each other. However, it is intractable to read these messages as they are encrypted using the public key of their friend. Moreover, friends are not able to read the sender-id of a message as this is encrypted as well.

Finally, peers with overlapping friends will expose which messages they create. This is due to the push based synchronization. Whenever a peer receives an update message without sending a synchronization request, it can assume that the sender created it. Disabling the push based synchronization will resolve this. However, at a latency cost, as synchronization requests are send in a periodic manner. As ReClaim aims to be an alternative to a centralized OSN, we choose latency over privacy as this only impacts privacy when dealing with peers which share at least one friend.

4 Features

Our privacy preserving decentralized social network will provide users with all the functionality found in current centralized solutions with the added benefit of remaining in control of your private data and hiding the social graph. However, this requires a method that allows a user to determine if any of its friends are online, allow him make new friends, and unfriend friends after a relationship has gone sour.

4.1 Establishing a new Friendship

In order to establish a new friendship, two users are required to exchange their identities. We use a public key to *identify* a peer, which is only shared with friends whom we want to communicate with. As there is no central public key infrastructure (PKI), we rely on out-of-band communication in order to send a friend-request. The public/private keypair is generated locally, but both parts of the key are kept secret.

Steps required to establish a new friendship

1. Using out-of-band communication Alice obtains the public key of Bob. E.g. Bob sends her an email with his public key inviting her to befriend him.
2. Next, Alice adds Bob's public key to her handshake message and starts looking for a peer which has this key in common.
3. After finding Bob or one of his friends, Alice encrypts her public key using Bob's public key and sends it to him/stores this message at his friend.
4. Finally, after receiving the public key of Alice, Bob can start sending her messages.

4.2 Locating Friends

Without a centralized lookup service, peers need another method to locate friends which are currently online. In ReClaim we achieve this with a method which determines if the peer we have just connected to is a friend or has a friend with us in common. We call this method $FSF_{A,B}$.

However, as we are trying to locate friends in a privacy preserving manner, we require that the method must be able to function without disclosing any of the friends Alice and Bob do not share. Moreover, it should be possible for Bob to forward the request of Alice, and should not be able to determine if any of his neighbors share friends with Alice.

The input of the FSF method is a friendset $\mathscr{F}$ which contains the identifiers of all friends of a user and its own identifier.

4.3 Posting a Message

When posting a message, Alice first signs the message using her private key. Next, Alice uses the public keys of each friend, to create separate encrypted copies. This allows for a very flexible manner of distributing messages, as Alice can decide per post whom to send this particular update to, e.g. it allows Alice to define groups whom to send a particular post to.

After creating the encrypted messages, Alice determines which of her current connections accepts messages for the friends she is sending the post to, and pushes the messages.

Peers additionally store the encrypted updates locally in order to reply to synchronization requests. Hence, if Alice does not have active connection for a particular friend, she will synchronize the message at a later time when replying to a synchronization request. In Section 5.2 we will describe this pull based synchronization mechanism in depth.

4.4 Unfriending

If Alice unfriends Bob, she can simply decide locally to stop sending messages to him. However, this will still allow Bob to locate Alice and her friends, as her public key (her identity) has not changed.

In order to permanently unfriend Bob, Alice changes her identity (public/private keypair), and sends all her friends (which now does not include Bob) her new public key, and she can switch identities. However, as changing identities can cause Alice to miss messages addressed to her old identity, we implement a time period in which Alice uses both her new and old identity in order to minimize this risk.

5 Protocol Details

In the following paragraphs we will introduce the FSF method and the synchronization protocol we developed and implemented for ReClaim.

5.1 PSI protocol

As mentioned in Section 4.2, ReClaim requires a $FSF_{A,B}$ method which allows two peers to determine which of their friends overlap. In our implementation we use an existing approach to the PSI problem as defined by Freedman *et al.* [4]. We define the *friendset* of Alice as $\mathscr{F}_A$, and the friendset of Bob as $\mathscr{F}_B$.

Protocol 1: PSI protocol, communication between Alice and Bob proceeds as follows:

1. Alice builds a polynomial having roots in each of her friends contained in her set $\mathscr{F}_A$. That is, she computes the $n+1$ coefficients $\alpha_0,\ldots,\alpha_n$ of the polynomial
$$f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_n x^n \qquad (1)$$
for which $f(\mathscr{F}_{A,i}) = 0$ for any friend in her set.
2. Next, Alice encrypts the coefficients and sends them to Bob, e.g. sending him $\mathscr{E}_{pk_A}(f(x))$.
3. Bob uses the homomorphic properties of the encryption scheme to evaluate the polynomial for each item in his set $\mathscr{F}_B$, obtaining $\mathscr{E}_{pk_A}(f(\mathscr{F}_{B,i}))$.
4. Bob multiplies each result with a fresh random number r_i, and then adds the original input to the result $\mathscr{E}_{pk_A}(f(\mathscr{F}_{B,i}) \times r_i + \mathscr{F}_{B,i})$
5. Bob appends all evaluated polynomials to a list, permutes the order, and sends it back to Alice.
6. Alice decrypts each ciphertext and verifies if any of the items occur in her friendset $\mathscr{F}_A$. The items which overlap are the shared friends between her and Bob.

We encrypt the coefficients of the polynomial with the Paillier cryptosystem [9]. This allows Bob to compute the output of the polynomial, but prevents him from knowing if the output is 0 or a random value. After sending the value back to Alice, she can decrypt it, allowing her to see which friends are shared between her and Bob. However, as Bob is multiplying the value of each evaluated polynomial with a fresh random value, Alice cannot know which friends are in his set and not in hers.

Moreover, after performing an handshake Alice cannot differentiate between Bob being a friend, or Bob having some friends in common due to him permutating the order of the evaluated polynomials.

We can obtain the polynomial required in step 1 by simple multiplication of the factors $(x - \mathscr{F}_{A,i})$. This way, the resulting polynomial will have the most significant coefficient take the value 1 and a degree equal to the number of items in Alice's set.

Finally, we use hashing and partitioning to reduce the bandwidth required for the FSF handshake. This reduces the size of the Paillier key required to encrypt the polynomial as that depends on its degree and the size of the identifiers. For more information we refer to [12] wherein additionally a reduction in CPU overhead is described.

5.2 Message Distribution

In our implementation we build upon a fully decentralized message synchronization platform which provides us with semi-random peer selection and a synchronization technique based upon Bloom filters.

Bloom filters are compact hash-representations which allow for membership testing [2]. A peer uses Bloom

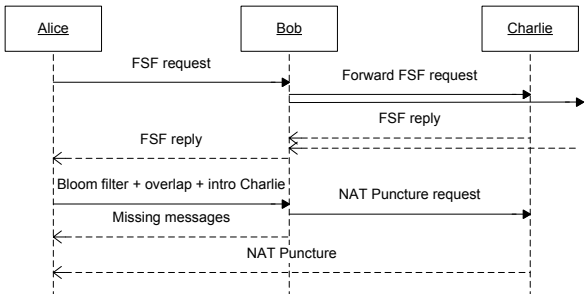

Figure 1: Synchronizing messages in the social network

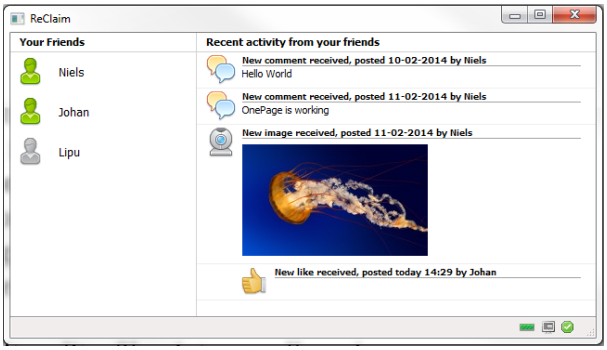

Figure 2: ReClaim proof-of-concept application

filters to tell another peers which messages it already received without having to send a complete copy of its database. Upon receiving a request, a peer will reply with all messages it has locally but the requesting peer has not. In our system, we modify synchronization approach, as we only want to synchronize messages of friends overlapping between us and the peer sending the request.

Figure 1 gives an overview of the synchronization steps. Bob forwards the *FSF* request of Alice to Charlie (and others), and waits for their replies before replying to Alice. This allows Alice to request Bob to send a message to one of Bob's neighbors which is most interesting to Alice (in this case Charlie) and instructing him to puncture/prepare his NAT-firewall. Puncturing a NAT-firewall will allow Alice to connect to Charlie in a next handshake round.

The synchronization technique will eventually cause all peers to store all messages addressed to their friends as well as those addressed to themselves. E.g. adding an identity of a friend to your friendset, implies that you as a peer are willing to store all messages addressed to this friend in order to assist him to receive them.

5.3 Peercache

In ReClaim, peers themselves are responsible for establishing connections to their friends, and friends of friends. However, by reducing the time it takes to establish those connections, we will improve the user experience substantially. Therefore, a ReClaim peer will maintain a peercache containing the ip/port addresses of peers which share at least one friend and are directly connected to the Internet. These peers can assist in reconnecting to friends, as messages sent these peers are not blocked by a NAT-firewall, and hence we can easily verify if they are online.

Halkes et al. [5] measured that up to 64% of peers in a P2P network are behind a NAT-firewall, hence we expect to be able to create a peercache consisting of 36% of peers which share at least one friend. In a social network, this results in a substantial amount of peers, as users typically have a high number of friends, and hence an even larger number of friends of friends.

6 Application Design

We implement ReClaim using the core of Tribler [11]. Tribler is the product of a research project funded by multiple European research grants and provides its users with the ability to download .torrents, remote search, video-on-demand, live streaming, etc. Using only the core allows us to cherrypick some of its components for our social network.

We implement our *FSF* method using Dispersy [13] and modify its core to allow for subsets of messages to be synchronized based on the shared friends between two peers. Moreover, in the network peers maintain a list of active connections to peers which share friends. After detecting the NAT-type of these peers, we create a peercache containing peers which share friends and have a direct connection to the Internet. These peers are used to quickly bootstrap into the social network after starting the application.

A screenshot of the proof-of-concept ReClaim application is shown in Figure 2. In the next section, we will evaluate the performance of this proof-of-concept application when emulating a network consisting of 1000, 2000, and 4000 peers, running this exact code.

7 Evaluation

In this section we will evaluate the performance of ReClaim using a Facebook dataset collected by McAuley et al. [6]. This dataset contains just over 4000 users and their relationships. We use this dataset to visualize the speed of bootstrapping the P2P network (the speed at which you connect to your friends), and the impact of the network size by creating two additional subsets consisting of 1000 and 2000 users by selecting random edges.

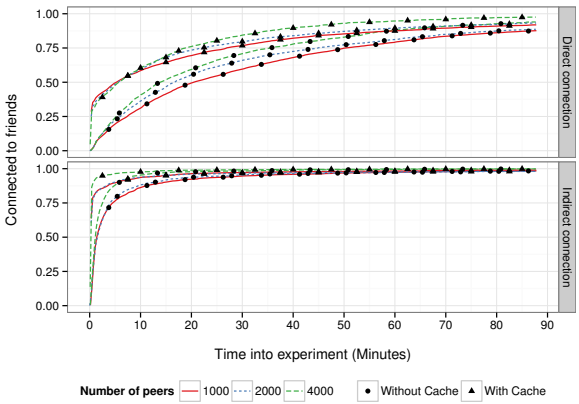

Figure 3: Bootstrapping speed of peers in a ReClaim overlay

7.1 Setup

To emulate ReClaim we deploy a separate process per peer using 20 nodes of our supercomputer. Each process is running the code from the proof of concept application introduced in the previous section with some additional instrumentation to be able to monitor the connections of each peer.

7.2 Boostrapping

Using the dataset we show the speed of locating friends in an ReClaim network. In this experiment 80% of the peers are online, and 20% of peers join/start locating their friends. We perform the experiments with and without the peercache, showing the speedup of being able to connect to 36% of your friends at startup. Moreover, we show the speed at which peers find an indirect connection to their friends. These indirect connections are used to store replicas of messages destined for your friends.

We expect any peer to be able to connect to 36% of its friends because of the NAT-firewall is preventing a direct connection to the other 64% of your friends (Section 5.3).

Figure 3 shows the result of the experiments. In total we have run six experiments by varying the number of peers and enabling/disabling the peercache. Each experiment took 1.5 hours.

From the graph we can observe three things, first using the peercache substantially improves bootstrapping into the network as we can connect to 36% of our online friends. Second, doubling or quadrupling the size of the overlay does not have a substantial impact on the speed of bootstrapping. Third, although the initial speed of bootstrapping is quite rapid (connecting to 60% of online friends in 10 minutes) it drops off and slows down.

The slowdown is an indication that the process of locating friends is being hindered by clustering. Although, a social graph has a high clustering coefficient, peers are stuck in a local optima which slows down the speed of locating friends which are not currently connected. Adding more randomness while forwarding the similarity request created by Alice to the selection of neighbors by Bob will improve this. However as the degree of randomness depends on the structure of the social graph this optimization is out the scope of this paper.

Moreover, ReClaim does not require a direct connection to a friend to be able to send messages to him. If no direct connection is available, it will use the friends of a user to make sure that a message will still arrive. This is visualized in the Figure 3 using the indirect connection lines. Peers are able to connect to more than 95% of their friends indirectly within 10 minutes.

8 Conclusion

In this paper we have shown that it is possible to build a fully decentralized social network which does away with the requirement of a centralized server. Using a realistic workload and churn model, we have evaluated the performance of ReClaim and shown that it can effectively synchronize messages in a network consisting of 4000 peers. Thereby, removing the need of storing privacy sensitive information in a single compromised location, and instead making users themselves are responsible for their data.

ReClaim is built on top of existing cryptographic primitives which allow us to limit the exposure to peers which have at least one friend in common. We have shown that using the *FSF* handshake a peer can connect to more than 60% of its friends in a system consisting of 4000 peers within 10 minutes. Moreover, it can create an indirect connection to 95% of its friends within 10 minutes.

Aside from experimental evaluation, we build a proof-of-concept application which showcases ReClaims basic functionality. The application allows users to communicate with their friends, exchange pictures, post likes, etc. in a fully decentralized manner. All traffic is encrypted, and so are the messages stored at friends.

References

[1] BEATO, F., KOHLWEISS, M., AND WOUTERS, K. Scramble! your social network data. In *PETS '11*, vol. 6794 of *Lecture Notes in Computer Science*. 2011, pp. 211–225.

[2] BLOOM, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM 13*, 7 (July 1970), 422–426.

[3] CUTILLO, L., MOLVA, R., AND STRUFE, T. Safebook: A privacy-preserving online social network leveraging on real-life trust. *Communications Magazine, IEEE 47*, 12 (2009), 94–101.

[4] FREEDMAN, M. J., NISSIM, K., AND PINKAS, B. Efficient private matching and set intersection. In *EUROCRYPT '04* (2004), pp. 1–19.

[5] HALKES, G., AND POUWELSE, J. Udp nat and firewall puncturing in the wild. In *Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part II* (Berlin, Heidelberg, 2011), NETWORKING'11, Springer-Verlag, pp. 1–12.

[6] MCAULEY, J., AND LESKOVEC, J. Learning to discover social circles in ego networks. In *Advances in Neural Information Processing Systems 25* (2012), pp. 548–556.

[7] MEGA, G., MONTRESOR, A., AND PICCO, G. Efficient dissemination in decentralized social networks. In *Peer-to-Peer Computing (P2P), 2011 IEEE International Conference on* (2011), pp. 338–347.

[8] NILIZADEH, S., JAHID, S., MITTAL, P., BORISOV, N., AND KAPADIA, A. Cachet: A decentralized architecture for privacy preserving social networking with caching. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies* (New York, NY, USA, 2012), CoNEXT '12, ACM, pp. 337–348.

[9] PAILLIER, P. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology — EUROCRYPT '99* (May 2-6, 1999), vol. 1592 of *LNCS*, Springer, pp. 223–238.

[10] SUN, J., ZHU, X., AND FANG, Y. A privacy-preserving scheme for online social networks with efficient revocation. In *INFOCOM, 2010 Proceedings IEEE* (2010), pp. 1–9.

[11] ZEILEMAKER, N., CAPOTĂ, M., BAKKER, A., AND POUWELSE, J. Tribler: P2p media search and sharing. In *Proceedings of the 19th ACM international conference on Multimedia* (New York, NY, USA, 2011), MM '11, ACM, pp. 739–742.

[12] ZEILEMAKER, N., ERKIN, Z., PALMIERI, P., AND POUWELSE, J. Building a privacy-preserving semantic overlay for peer-to-peer networks. In *Information Forensics and Security (WIFS), 2013 IEEE International Workshop on* (Nov 2013), pp. 79–84.

[13] ZEILEMAKER, N., SCHOON, B., AND POUWELSE, J. Large-scale message synchronization in challenged networks. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC)* (Gyeongju, Korea, March 2014).