Privacy-Preserving Peer Discovery for Group Management in p2p Networks

Tommi Meskanen, Valtteri Niemi
University of Helsinki
Helsinki, Finland
{tommi.meskanen, valtteri.niemi}@helsinki.fi

Jarkko Kuusijärvi
VTT Technical Research Centre of Finland Ltd
Oulu, Finland
jarkko.kuusijarvi@vtt.fi

***Abstract*—The necessity for peer-to-peer (p2p) communications is obvious; current centralized solutions are capturing and storing too much information from the individual people communicating with each other. Privacy concerns with a centralized solution in possession of all the users data are a difficult matter. HELIOS platform introduces a new social-media platform that is not in control of any central operator, but brings the power of possession of the data back to the users. It does not have centralized servers that store and handle receiving/sending of the messages. Instead, it relies on the current open-source solutions available in the p2p communities to propagate the messages to the wanted recipients of the data and/or messages. The p2p communications also introduce new problems in terms of privacy and tracking of the user, as the nodes part of a p2p network can see what data the other nodes provide and ask for. How the sharing of data in a p2p network can be achieved securely, taking into account the user's privacy is a question that has not been fully answered so far. We do not claim we answer this question fully in this paper either, but we propose a set of protocols to help answer one specific problem. Especially, this paper proposes how to privately share data (end-point address or other) of the user between other users, provided that they have previously connected with each other securely, either offline or online.**

I. Introduction

Privacy concerns and collection of data by the centralized social media services are driving the need and development towards decentralized and distributed solutions. Numerous cases of data breaches by centralized platforms have been reported thus far [7], [12]. In the premise of these requirements, a decentralized solution is needed, in order to fulfill the end-users needs and wants, the end-user requirements. In order to achieve a decentralized social media platform, a p2p solution is required. A number of decentralized solutions have been proposed, such as [2], [4], that do address the base for achieving the goals towards having a decentralized solutions (and in some cases taking into account privacy aspects at some level), but they do lack the solution of achieving un-tractability in the networks.

The p2p networks are open in terms of connections and data exchange between peers, including all the communicating nodes seeing the end-point addresses of the other connected nodes, obviously. This cannot be fully avoided with any generic p2p solution as such (at least without onion routing), as peers communicating with each other will have to create connections with each other in order to exchange data and function as a true p2p network. This is the case when the communicating nodes cannot directly connect to each other, e.g., they may be behind distinct Network Address Translation (NAT). The problem and concern we address in this paper is that we do not want our identification data (the user ID/IP address) to be shared with non-relevant other person(s) involved in the p2p network. In order to achieve this, we propose a set of protocols to share identification data in the environment to a set of users to whom we want to share these data. To exchange data with the other users, we need to know the actual end-point addresses of our friends, i.e., IP addresses and the ephemeral IDs. The premise of HELIOS [6] platform is that users can communicate with each other in a p2p network, while taking into account the overall security and privacy of the user. The restrictions/technical problems explained above need to be solved, in order to make the user untraceable. One solution to achieve some level of untraceability is to change the used end-point address and the ID of the user at certain intervals.

The libp2p [11] can connect to other nodes using a Circuit Relay (or relay in networking terms) in case the nodes are behind a NAT network and cannot connect to each other directly. The relay feature can also be used to connect to the p2p network via a relay and exchange data, without connecting directly to the other nodes. Another way to achieve this would be to use another gateway outside to connect to the p2p network (of course that gateway would see the IP address of the user in this case also).

Peer-to-peer networks have many advantages compared to networks with centralized control. One of the advantages is privacy. Nobody in the network has a full view about what is going on. Sensitive data can also remain local when p2p paradigm is in use. On the other hand, one of the advantages of a centralized system over a p2p system is easier management. For example, nodes in the network can easily find each other with the help of a centralized database that contains up-to-date information about the network endpoints of all nodes. In this paper we show how nodes can find each other (i.e., peer discovery and/or sharing of data) in a privacy-preserving manner.

A user wants to share their IP address (or other identification information) to their friends but wants to keep it secret from the other users. The address is changing over time and a new address needs to be distributed to the friends. Some user

ISSN 2305-7254

may be removed from the set of friends also, so they should not learn the new address after they have been removed. In addition, some new friends may be added to the set of friends and they should also learn the new address after the addition.

We describe the concept with group of friends but, of course, other groups can be handled similarly.

In this paper we will describe how we will accomplish sharing data (e.g., IP address) to the people we have connected with in the past (have shared keys or such), while keeping this information as secure and private as possible in terms of the p2p network premise.

II. Related work

Different security issues related to p2p networks can be found in [9], [18]. The paper [3] presents a survey of different threats and security models for p2p streaming applications. In the paper [22] Urdaneta et al. consider the attacks against DHT (distributed hash table, see next section) based systems and protection techniques against these attacks. In particular, Sybil attacks are considered in [17]. Data privacy in p2p systems is extensively discussed in [8]. Privacy issues in p2p networks and the solutions for these issues are also discussed in [20] by Touceda et al.

A well-known technique for anonymous Internet communication is onion routing and the TOR network [21]. In their paper [1], Brack et al. use DHT to track covid-19 contacts in a privacy preserving way. One method for two machines to share session configuration information using DHT can be found in [14]. A different approach of using blockchain instead of DHT to share data is presented in [10]. Octopus [24] is a system developed for the purpose of secure and anonymous DHT lookups. The paper by Guidi et al. [5] presents the novel ideas of the HELIOS platform [6] as well as compares it to other social media platforms. This paper can also be seen as the motivation for issues in our study.

III. Preliminaries

We assume that there is an out-of-band channel for the user to change information with the new friend when they contact each other for the first time. It is difficult to make sure that people are what they claim to be in the Internet. Thus, it is better if people meet in person or have some other method to make sure that they are what they claim to be. One example of an other method is that they use some trusted third party to authenticate themselves. They may also have certificates granted for themselves to show that they are who they claim to be.

In addition to mutually authenticating each other, two users can share their public keys that are used for verifying digital signatures that they generate or self-signed certificates that contain public keys and identities by which the two would know each other.

One of the key components of the solution in this paper is a *cryptographic hash function [16].* A cryptographic hash function takes as an input a bit string of any length and returns a bit string of fixed length. It is assumed that evaluation of the hash function can be done efficiently. If $Hash$ is a cryptographic hash function then it is computationally infeasible to find two different inputs x and x' such that $Hash(x) = Hash(x')$ or to find x such that $Hash(x) = y$ when output y is given. SHA-256 is an example of a cryptographic hash function [23].

A cryptographic hash function is needed when using a distributed hash table.

A *distributed hash table* (DHT) is a lookup table that is distributed between several nodes. The table stores (key, value) pairs. The key is the hash value of value or the hash value of something else that is used to determine the location for storing the value. Typically the value is stored to several nodes where it can be retrieved using the same key. There are several methods to implement a DHT, for example Chord [19] and Kademlia [13].

In this paper, DHTs are used to share information utilizing a specific protocol proposed here in a secure way, without any third-party participant being capable of capturing/deducing/calculating the information at real-time with currently available methods. Naturally, when DHT is utilized to retrieve some content, other nodes in the network path will see a specific node requesting that content and can take use of this information.

In this paper, we assume that the user can write to, and the friends can read from, a distributed hash table. The cryptographic hash function, $Hash$, in this paper is the hash function that the chosen implementation of DHT uses.

A digital signature for a message or a bit string is calculated using the signing algorithm and a key that only the signer knows. This key is called the private key and it is closely related to another key that is called the public key. Using the verification algorithm anybody who has the public key can make sure that the message was signed using the private key. Therefore, only the person who has the possession of the private key could have created the signature. A well known method for generating electronic signatures is based on the RSA cryptosystem [15].

A logical key hierarchy [25] is a method to distribute a group key to a set of users. It is based on a (binary) tree. Each member of the group has a personal key and is assigned a leaf in the tree by the administrator of the group. The administrator picks a random key for all nodes of the tree. The encrypted values of the keys are then stored in the tree. The administrator, for each node except the root, encrypts the key in the parent node of the node with the key of the node and stores it in the node. For the leaves the personal keys of the members of the group are used for encryption. The group key is stored in the root of the tree and encrypted by the key picked for the root.

If any member of the group now has access to the encrypted keys in the tree and they have their personal key, they can find out the group key. The member can first decrypt the key located in their leaf by decrypting it by their personal key. Using the key they decrypted they can again decrypt the next key on the path from their leaf to the root. They can repeat the process until they reach the root and are able to learn the group key. Someone who does not have a personal key, or

any other key in the hierarchy, is unable to decrypt any of the leaves or any other key in the tree. A logical key hierarchy tree for seven members is presented in Fig. 1.

There are several benefits for using the logical key hierarchy. All the encrypted values in the tree can be broadcasted to all the members of the group. It does not matter if other persons than the members of the group are able to receive these encrypted values. They are anyway unable to decrypt these without a personal key. Thus there is no need for the administrator to communicate with the individual members.

In addition, it is relatively easy to add members to the group and deliver the group key to them. The administrator just adds a node to the tree such that there is one more leaf for the new member. For removing a member the following steps are needed:

1) The leaf of the removed member is removed from the tree.
2) New keys are picked to replace those that the removed member was able to learn.
3) These new keys are encrypted and stored the same way as in the original tree.
4) A new group key is picked, encrypted with the new key picked for the root and stored at the root of the table.

The personal keys of the remaining members need not to be changed, but the personal key of the removed member is no longer useful for learning the group key.

IV. The protocols

In this chapter, we present our protocols for sharing the IP address of user A to a group of their friends. In the first protocol, we assume that there exists a separate method to distribute a shared key to all friends in the group.

A. Protocol 0

Sharing phase:

1) A encrypts its address using the shared key that is distributed to all members of the group.
2) A stores the encrypted address to location $Hash(ID_A)$ in the DHT.

Here ID_A is an identity of A in bit string format that is assumed to be known by everyone who needs to learn the address of A.

When B needs to find out the address of A, she performs the following steps:

Retrieving phase:

1) B retrieves the encrypted key from the location $Hash(ID_A)$
2) B decrypts the address using the key that is distributed to all members of the group.

Only the persons who have the key are able to decrypt the address. When the address changes, A performs the sharing phase again. When a new friend is added, the key is distributed to her. If friends are removed from the set of friends, a new key must be distributed to all the remaining friends. In this solution, the DHT is only used to store the encrypted value of the address. The non-trivial problem of distributing the key is left out of this solution.

B. Protocol 1

In this solution, we assume that there is a way for A and B to authenticate each other and share their public keys. They do not need to tell their true identities to each other, but there must be some kind of identity, ID_A that B will associate to this person. User A can use the same identity with several other users and have other identities to use with the same or other users.

Setup phase:

1) A and B authenticate themselves to each other in some way.
2) A and B agree on a $sharedkey_{AB}$. They also share their public signature verification keys.

This setup phase needs to be done between A and all the friends of A. A different $sharedtkey_{AB}$ is chosen for each friend of A.

Sharing phase:

1) A chooses a $saltkey_A$, encrypts it with $sharedkey_{AB}$, and stores it to the DHT location $Hash(sharedkey_{AB}||ID_A)$. The same is repeated for every friend of A using the same $saltkey_A$.
2) A encrypts the address using $saltkey_A$.
3) A stores the encrypted address to location $Hash(saltkey_A||ID_A)$

Again, A needs to perform this step for all friends of A. When new friends are added, the same $saltkey_A$ is encrypted and stored for them. When some friends are removed, a new random $saltkey_A$ is chosen and updated to DHT for all remaining friends and the encrypted (preferably changed) address is stored to the new location in the DHT.

Retrieving phase:

1) B retrieves the encrypted $saltkey_A$ from the DHT location $Hash(sharedkey_{AB}||ID_A)$.
2) B decrypts the $saltkey_A$ with $sharedkey_{AB}$.
3) B retrieves the encrypted address from the DHT location $Hash(saltkey_A||ID_A)$.
4) B decrypts the address with $saltkey_A$.

If the $saltkey_A$ has not changed and B remembers it, it is enough for B to perform the last two steps. However, B cannot be sure of this unless they perform the first two steps, or notice that the address they get in Step 4 is no longer the correct one.

For added security, A could add his digital signature to everything he stores in the DHT. Then B could verify these signatures when she retrieves the values.

Compared to the previous protocol, removing friends is now more efficient. Also, the DHT is now used to distribute the group key ($saltkey_A$) for each friend.

t

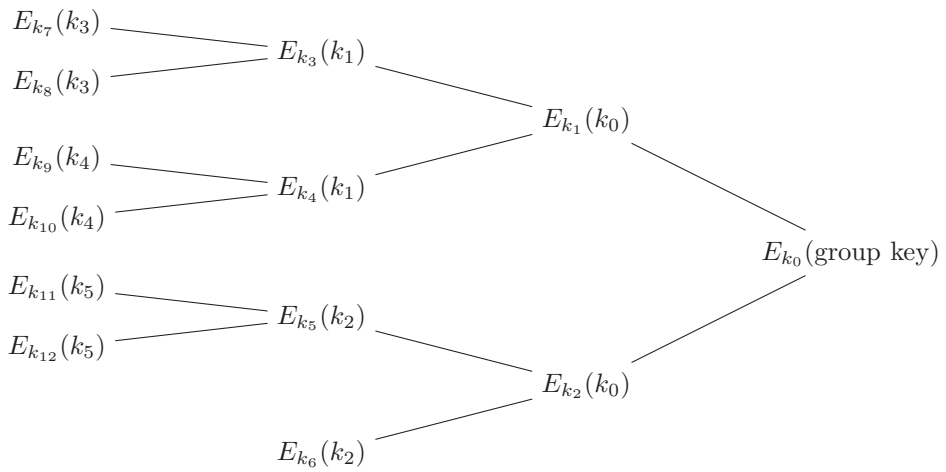

Fig. 1. Logical key hierarchy for seven members. Keys $k_6, k_7, \ldots, k_{12}$ are distributed to the members With a personal key a member can decrypt another key and finally the group key as long as all the encrypted values are made available.

C. Protocol 1.1

We could modify the previous protocol such that the locations in the DHT change over time. For example, the location could change every day and the TIME in the following could be the current date.

Setup phase:

1) A and B agree on a $sharedkey_{AB}$. They also share their public signature verification keys.

Sharing phase:

1) A chooses a $saltkey_A$, encrypts it with $sharedkey_{AB}$, and stores it to the DHT location $Hash(sharedkey_{AB}||ID_A||TIME)$. The same is repeated for every friend of A using the same $saltkey_A$.
2) A encrypts the address using $saltkey_A$.
3) A stores the encrypted address to location $Hash(saltkey_A||ID_A||TIME)$.

User A could repeat this sharing phase every time the value of the parameter TIME changes to make it easier for other users to locate the positions in the DHT. If A does not do this then another user B may need to repeat the following retrieving phase, using also earlier values of TIME as inputs.

Retrieving phase:

1) B retrieves the encrypted $saltkey_A$ from the DHT location $Hash(sharedkey_{AB}||ID_A||TIME)$.
2) B decrypts the $saltkey_A$ with $sharedkey_{AB}$.
3) B retrieves the encrypted address from the DHT location $Hash(saltkey_A||ID_A||TIME)$.
4) B decrypts the address with $saltkey_A$.

If the $saltkey_A$ has not changed and B remembers it, it is enough for B to perform the last two steps. But, again, B cannot be sure of this unless they perform the first two steps or notice that the address they get in Step 4 is no longer the correct one.

Changing the location in the DHT over time makes it more difficult for the nodes in charge of the DHT to track when the address changes and who are reading the location where the address is stored.

This solution has advantages over the previous one, but the whole structure needs to be stored to the DHT every time TIME changes. Even through it would be enough to just change the information in location $Hash(saltkey_A||ID_A||TIME)$ if all the friends know the $saltkey_A$, there is no method in the protocol for A to be sure that everybody knows the current $saltkey_A$.

D. Protocol 2

The next solution is based on the ideas of LKH [25].

Setup phase:

1) A and B agree on a $sharedkey_{AB}$. They also share their public signature verification keys.

This phase is the same as before.

We assume that A has constructed a binary tree where the leaves are all his friends. Each node in the tree has a random saltkey associated to it, except the leaves. The leaf that corresponds to B has the $sharedkey_{AB}$ associated to it. The situation is presented in Fig. 2.

Sharing phase:

1) For every non-root, non-leaf node of the binary tree, A encrypts the saltkey associated with its parent ($saltkey_{parent}$) with the saltkey associated with the node ($saltkey_{node}$), and stores the result to the DHT location $Hash(saltkey_{node}||ID_A)$.
2) For every leaf node B of the binary tree, A encrypts the saltkey associated with its parent ($saltkey_{parent}$) with the sharedkey associated with the node ($sharedkey_{AB}$), and stores the result to the DHT location $Hash(sharedkey_{AB}||ID_A)$.

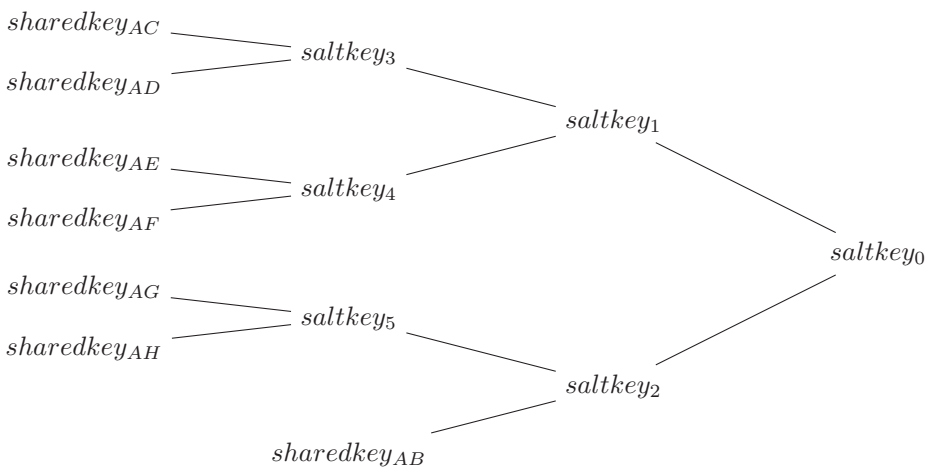

Fig. 2. Binary tree of saltkeys for seven friends B,C,...,H of A in Protocol 2.

3) A encrypts the IP address using the saltkey associated with the root ($saltkey_{root}$).
4) A stores the encrypted address to location $Hash(saltkey_{root}||ID_A)$

The resulting tree of encrypted saltkeys is presented in Fig. 3.

Retrieving phase:

1) B retrieves the encrypted $saltkey_{parent}$ from the DHT location $Hash(sharedkey_{AB}||ID_A)$.
2) B decrypts the $saltkey_{parent}$ with $sharedkey_{AB}$.
3) B continues recursively retrieving node's parent's saltkeys and decrypting them until the $saltkey_{root}$ is reached.
4) B retrieves the encrypted address from the DHT location $Hash(saltkey_{root}||ID_A)$.
5) B decrypts the address with $saltkey_{root}$.

If the retriever remembers some of the saltkeys, she does not need to start retrieving from the leaf, but somewhere closer to the root. It may be enough to just retrieve the value in the root. However, again, B cannot be sure whether the saltkeys have changed unless they check them from the DHT or the address is found to be old.

A friend B can be added by adding an inner node to the tree. Let $sharedkey_{AC}$, be the sharedkey associated with the leaf C, let $saltkey_P$ be the saltkey associated with the parent of L let B have the saltkey $sharedkey_{AB}$.

Adding phase:

1) A chooses a new $saltkey_N$ for the new leaf N.
2) A encrypts the $saltkey_P$ with the $saltkey_N$ and stores it to the DHT location $Hash(saltkey_N||ID_A)$.
3) A encrypts the $saltkey_N$ with the $sharedkey_{AL}$ and stores it to the DHT location $Hash(sharedkey_{AL}||ID_A)$.
4) A encrypts the $saltkey_N$ with the $sharedkey_{AB}$ and stores it to the DHT location $Hash(sharedkey_{AB}||ID_A)$.

A friend B can be removed as follows: By changing its sibling to the place of its parent, changing all the saltkeys of its grandparent, grand grandparent and so on till the root, re-encrypting them with the saltkeys associated with their siblings and storing the new values to the DHT as above.

Removing phase:

1) A chooses a new $saltkey_G$ for the grandparent G of removed leaf.
2) The sibling S of the removed leaf is now replacing the parent of the deleted leaf.
3) A encrypts the $saltkey_G$ with the $saltkey_S$ and $saltkey_{S'}$ associated with the children of G and stores them to the DHT location $Hash(saltkey_S||ID_A)$ and $Hash(saltkey_{S'}||ID_A)$. (If S is a leaf then $sharedkey_{AS}$ replaces the $saltkey_S$ here. The same is done for S'.)
4) A chooses a new $saltkey_{parent}$ for the parent of G.
5) A encrypts the $saltkey_G$ with the $saltkey_G$ and $saltkey_{G'}$ associated with the sibling of G and stores them to the DHT location $Hash(saltkey_G||ID_A)$ and $Hash(saltkey_{G'}||ID_A)$. (Again, if G' is a leaf then $sharedkey_{AG'}$ replaces the $saltkey_{G'}$ here.)
6) A continues recursively towards the root until the root is reached.

Removing a friend from the set of friends is even more efficient in this protocol than in the previous ones, because only $2\log_2 n$ entries in the DHT needs to be changed. Here n denotes the number of friends. When adding and removing friends the binary tree may sometimes need some restructuring to keep it as close to balanced as possible.

After adding and removing friends, the binary tree may need to be restructured to keep it as close to balanced as possible. This will require additional encryption and store operations.

If somebody is removed from the set of friends, some of the remaining friends need to start retrieving the saltkeys far away from the root because the saltkeys have changed during the removal. For about half of the remaining friends only the

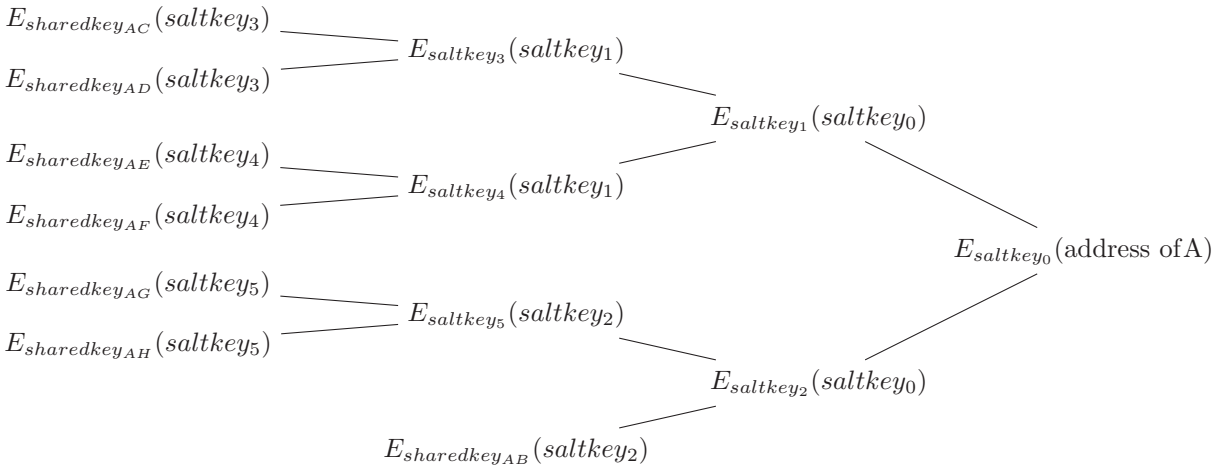

Fig. 3. Encrypted saltkeys for seven friends of A.

TABLE I. COMPARISON OF DHT OPERATIONS IN PROTOCOLS. HERE t IS THE NUMBER OF DIFFERENT VALUES FOR TIME AND n IS THE NUMBER OF FRIENDS

	sharing	adding friends	removing friends	retrieving
Protocol 1	$n+1$	1	n	≤ 3
Protocol 1.1	$(n+1)t$	1	n	≤ 3
Protocol 2	$2n$	3	$2\log_2 n$	$\leq \log_2 n + 1$
Protocol 2.1	$2nt$	3	$2\log_2 n$	$\log_2 n + 1$

saltkey in the root has changed. For about one fourth of the friends two saltkeys closest to the root have changed, and so on.

E. Protocol 2.1

It is possible to modify Protocol 2 in the same way as Protocol 1 and make the locations in the DHT change over time. In this case, the whole tree needs to be stored to the DHT every time the TIME changes. Otherwise, the retrievers need to guess when the new saltkeys were stored.

V. PERFORMANCE ANALYSIS

In Protocol 0 only one store by A is needed and one retrieve for each friend is needed. However, the matter of sharing the key is not addressed. Therefore, we do not compare Protocol 0 to the other protocols in the following.

Let us assume that A has n friends. In practical applications, typical value for n could be from a dozen to a few hundreds. In Protocol 1, $n+1$ store operations are needed in the sharing phase, Adding a friend requires one store operation by A and removing a friend requires n stores, because a new $saltkey_A$ must be distributed to all remaining friends. Retrieving the address requires one, two or three retrieves depending on whether the $saltkey_A$ has been changed and whether the retriever remembers the $sharedkey_{AB}$. In all of these protocols, if the address changes, one store operation is enough to update it.

In Protocol 1.1, the number of operations is the same as in Protocol 1 except that the sharing phase needs to be repeated every time the TIME changes. That is, $n+1$ store operations are needed at fixed predictable times.

In Protocol 2, the sharing phase requires $2n$ store operations. If the binary tree is balanced, $\log_2 n + 1$ retrieve operations is enough for the retrieving phase. Three store operations are needed for adding a friend from the group and $2\log_2 n$ operations are needed for removing a friend, if the binary tree is balanced.

Again, the Protocol 2.1 is similar to Protocol 2 in efficiency, except that $2n$ store operations need to performed every time a new TIME is met.

This analysis is summarized in Table 1.

VI. CONCLUSION AND FUTURE WORK

The main contribution of this paper is the proposed set of protocols to share information securely in p2p networks with other users after an initial contact with them has been established. Our protocols utilize cryptographic functions for the purpose of protecting privacy of the group members. The goal is to prevent outsiders from learning identities of the group members and from tracking activities of the group members.

Our goal is to be able to use the p2p network also for the discovery of other nodes (and their end-point addresses) with whom we intend to communicate. Realizing a privacy-aware p2p platform that can share the identification information/end-point address only to a restricted set of users, while being a part of a public p2p network, is crucial. Being able to hide the top-level identity of the user and to replace it with an

ephemeral ID is something that is also required to improve the users privacy in a p2p network.

We are planning to implement the most promising protocols defined here and evaluate them in a simulation setting or a proof of concept implementation. Especially, we are going to utilise the protocols to share current ephemeral data of a user to a set of other users who are allowed to receive it.

ACKNOWLEDGMENT

This work is supported by the HELIOS H2020 project. HELIOS has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 825585.

REFERENCES

[1] S. Brack, L. Reichert and B. Scheuermann, "Decentralized Contact Tracing Using a DHT and Blind Signatures", IACR Cryptol. ePrint Arch., 2020, 398.

[2] L. A. Cutillo, R. Molva and T. Strufe, "Safebook: A privacy-preserving online social network leveraging on real-life trust", IEEE Communications Magazine, 47(12), IEEE, 2009, pp. 94-101.

[3] G. Gheorghe, R. L. Cigno and A, Montresor, "Security and privacy issues in P2P streaming systems: A survey", Peer-to-Peer Networking and Applications 4.2, 2011, pp. 75-91.

[4] K. Graffi and N. Masinde, "LibreSocial: A Peer-to-Peer Framework for Online Social Networks", arXiv preprint arXiv:2001.02962, 2020.

[5] B. Guidi, K. G. Kapanova, K. Koidl, A. Michienzi and L. Ricci, "The Contextual Ego Network P2P Overlay for the Next Generation Social Networks", Mobile Networks and Applications, 2020, pp. 1-13.

[6] HELIOS project homepage, Web: https://helios-h2020.eu/.

[7] M. Isaac and S. Frenkel, "Facebook Security Breach Exposes Accounts of 50 Million Users", Web: https://www.nytimes.com/2018/09/28/technology/facebook-hack-data-breach.html.

[8] M. Jawad, P. Serrano-Alvarado and P. Valduriez, "Supporting data privacy in p2p systems", in Security and Privacy Preserving in Social Networks, Springer, Vienna, 2013, pp. 195-244.

[9] J. Li, "A Survey of Peer-to-Peer Network Security Issues", 2007, Web: https://www.cse.wustl.edu/ jain/cse571-07/ftp/p2p/.

[10] G. Li and H. Sato, "A Privacy-Preserving and Fully Decentralized Storage and Sharing System on Blockchain", in 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC) Vol. 2, IEEE, 2019, pp. 694-699.

[11] Libp2p homepage, Web: https://libp2p.io/.

[12] D. Lynskey, "'Alexa, are you invading my privacy?' the dark side of our voice assistants", Web: https://www.theguardian.com/technology/2019/oct/09/alexa-are-you-invading-my-privacy-the-dark-side-of-our-voice-assistants.

[13] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric", in International Workshop on Peer-to-Peer Systems, Springer, Berlin, Heidelberg, March 2002, pp. 53-65.

[14] R. Moore, C. Morrell, R. Marchany and J. G. Tront, "Utilizing the BitTorrent DHT for blind rendezvous and information exchange", in MILCOM 2015-2015 IEEE Military Communications Conference, IEEE, 2015, pp. 1560-1565.

[15] R. L. Rivest, A. Shamir, and L. Adleman. "A method for obtaining digital signatures and public-key cryptosystems", Communications of the ACM, 21(2), 1978, pp. 120-126.

[16] P. Rogaway and T. Shrimpton., "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance", in International workshop on fast software encryption, Springer, Berlin, Heidelberg, Feb. 2004, pp. 371-388.

[17] H. Rowaihy, W. Enck, P. McDaniel and T. La Porta, "Limiting sybil attacks in structured p2p networks", iIn IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications, IEEE, 2007, pp. 2596-2600.

[18] E. Sit and R. Morris, "Security considerations for peer-to-peer distributed hash tables", in International Workshop on Peer-to-Peer Systems. Springer, Berlin, Heidelberg, 2002, pp. 261-269.

[19] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications", ACM SIGCOMM Computer Communication Review, 31(4), 2001, pp. 149-160.

[20] D. S. Touceda, J. M. S. Cámara and J. T. Isaac, "Privacy in Peer-to-Peer Networks", in Privacy in a Digital, Networked World, Springer, Cham, 2015, pp. 111-139.

[21] Tor Rendezvous Specification - Version 3, Web: https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt.

[22] G. Urdaneta, G. Pierre and M. V. Steen, "A survey of DHT security techniques", ACM Computing Surveys (CSUR), 43(2), 2011, pp. 1-49.

[23] US National Institute of Standards and Technology, "Federal Information Processing Standards Publication 180-4: Secure Hash Standard", Web: http://www.csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf, 2012.

[24] Q. Wang and N. Borisov, "Octopus: A secure and anonymous DHT lookup", in 2012 IEEE 32nd International Conference on Distributed Computing Systems, IEEE, 2012, pp. 325-334.

[25] C. K. Wong, M. Gouda and S. S. Lam, "Secure group communications using key graphs", ACM SIGCOMM Computer Communication Review, 28(4), 1998, pp. 68-79.