Privacy-friendly Discovery of Common Friends in P2P Networks

Tommi Meskanen*, Jarkko Kuusijärvi†, Sara Ramezanian*, Valtteri Niemi*
*University of Helsinki and Helsinki Institute for Information Technology, Helsinki, Finland
{tommi.meskanen, sara.ramezanian, valtteri.niemi}@helsinki.fi
†VTT Technical Research Centre of Finland Ltd, Oulu, Finland
jarkko.kuusijarvi@vtt.fi

***Abstract*—In this paper we study the problem of comparing a set of data between two parties in a peer-to-peer network to determine the number of common friends. Several protocols for private set intersection are presented in the literature. When the sets are large these tend to be too slow for many purposes. We consider the problem of two parties finding out how many common friends they have in a privacy preserving way. This problem has arisen in designing a peer-to-peer platform called HELIOS. We present our solution for the problem that is more efficient than older protocols but still sufficiently privacy-friendly for our purposes. The solution is based on iteratively revealing information about the hash values of friends' identities in small increments.**

I. Introduction

We study the problem where two individuals, Alice and Bob, are interested in how many common friends they have. They do not want to reveal how many friends they have or who are their friends. However, they may be willing to reveal to each other who are the common friends. This is a problem that arises when trying to update the social networks of users in peer-to-peer networks in a privacy preserving way.

This problem can be solved, for example, by applying a private set intersection protocol. Typically, private set intersection protocols are complex and require a lot of computations and data exchange between the parties [10]. Thus there is a need for a more efficient solution. The problem can also be solved by using a trusted third party. However, peer-to-peer networks are usually built under the premise that no party needs to be trusted by everyone. We consider the problem in the case there is no trusted party available. We also assume that Alice and Bob have limited computation capabilities. We assume that in the context of friendships, the solution to the problem may expose a moderate amount of information about the friends of one user to another user if in trade-off the protocol becomes more efficient in practice.

In our solution Alice calculates the hash values of her friends (from their names, user accounts, email addresses or some other attributes) and reveals to Bob some prefixes that none of Alice's friends have in their hash values. Bob does the same and reveals to Alice some prefixes that none of his friends have. We estimate how much information Bob learns about the friends of Alice that are not friends of Bob and conclude that this is a fair trade-off between privacy and efficiency in this context. In addition to finding common friends, the protocol can be used in other contexts where two parties need to find out intersection of two sets of input data, and this should be done by revealing only minimal amount of data about the elements that are not in the intersection.

HELIOS is a new distributed social media platform. The HELIOS platform is designed to protect users' privacy and users have control over how much information they share about each other [5]. Because of this the users may wish not to share their contacts to other users. However, this information could be useful when users want to find other users with whom they may share common interests. Users also tend to trust other users more, even if previously unknown to them, when they share a fair amount of common friends. The number of common friends can be used as one measure in building a trust value between two actors in the social graph. The trust value is useful when adding new actors to one's social graph, as well as later on, when building the trust further. Thus it is beneficial to develop methods for users to share information in a privacy preserving way. [4]

In [6], a trust metric is used to calculate a composite of individual security metrics that focus on the trustworthy behaviour of the other party through a trusted third party. This behaviour is measured with passive or active measurements of the other party, focusing on the cybersecurity related technical measurements. In this paper, we focus on developing a protocol to discover the common data items between two peers, specifically the number of common friends. The result of the proposed protocol can be used as one value, or metric, to compose a trust value combining also other metrics, such as, technical [6] or interaction, distance or information truthfulness [4] based ones. It can also be used solely to calculate the number of common friends in a more privacy-preserving manner without other metrics.

The paper is organized as follows: In Section 2, we present the mathematical and cryptographical building blocks that are needed later in the paper. In Section 3, we list some alternative solutions to solve the problem we are studying. The algorithm that we propose is presented in Section 4. After that, in Section 5, we analyse the accuracy and the complexity of the protocol. In Section 6 we explain some improvements of the protocol that could be achieved using probability theory. The paper is concluded with Section 7.

II. Preliminaries

The algorithm presented in this paper is based on a *cryptographic hash function* [12]. A cryptographic hash function takes as an input a bit string of any length and returns a bit

ISSN 2305-7254

string of a 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 [13].

In a *keyed hash function* a secret key is used together with the input string to generate the hash value [1].

In a *secure multi-party computation* there are several parties, at least two. The parties follow a protocol to generate a result of the computation. Each party may have an input. Typically the parties send messages to each other and do calculations with the data they have received. [14], [15]

A *trusted third party* is an entity that other parties can employ in a secure multi-party protocol. The other parties trust that the trusted third party always behaves as instructed by the protocols and the only information the third party reveals to anybody else is the information described in the protocol. The trusted third parties are very useful but often it is difficult to find entities that all other parties trust and that are also worthy of this trust.

A *private set intersection* (PSI) protocol involves two or more parties that all have some finite and private set of elements in their possession. A PSI protocol is a method to compute the intersection of these sets and to reveal the intersection or the size of the intersection to all parties. This method usually has the requirement that nothing else about the private sets of the parties is revealed to other parties.

Several solutions are presented in the literature for the PSI problem but all the general purpose solutions tend to require several rounds between the parties and considerable amount of computing resources [8], [10].

Our solution is motivated by a use case in peer-to-peer networks. Details of such networks, including security aspects, are explained, e.g., in the book by Korzun and Gurtov [7].

III. Related work

A trivial way to solve our problem could be: Alice sends list of her friends to Bob and Bob sends back those who are also his friends. Of course, then only Bob's non-common friends remain private.

A straight-forward way to try to solve our problem with hash functions is as follows. Alice calculates the hash values of all her friends individually and sends the values to Bob. Bob also calculates the hash values of all his friends individually and sends the values to Alice. Both of them can now count how many of their hash values match with the hash values of the other party. This is the number of their common friends.

This approach has several shortcomings. If Alice has the interest to know if some person that is not one of her friends is a friend of Bob, she may calculate the hash value of that person and compare it with the hash values Bob sent. She can do this even long after the protocol is finished.

Also, if Alice performs this protocol also with David, she can compare the hashes Bob and David sent and find out how many common friends Bob and David have between them.

Still another problem with this approach is that Alice herself can use her correct hash values when calculating the result but could send Bob some random hashes and thus Bob is not able to calculate the correct result.

One improvement for this protocol would be that Alice and Bob use a keyed hash function and the random key is agreed together between Alice and Bob. Now Alice cannot use the hash values she has received from Bob and David to find out how many common friends Bob and David have. This is because Bob and David use different keys when computing the hashes. Although Alice knows both keys, she cannot find out whether two hash values result from the same input.

The protocol could be made more secure by using *blind signatures* [2]. Bob signs identity of each of his friends with his private signing key and sends the hash values of the signatures to Alice. On the other hand, Alice asks a blind signature from Bob for identities of all her friends and then calculates the hash values of these signatures. By doing comparisons between two sets of hash values Alice can find out the number of their common friends.

Alice cannot test if a person that is not her friend is a friend of Bob because she does not know Bob's signature for this person. Here, and later in the paper, we assume that Alice's friends are the individuals that she uses as the input to the protocol. She could, of course, pretend that she has more friends than she has and use the identities of individuals she has seen in TV as her friends. Protocols like the ones discussed in this paper cannot protect against such false input values.

On the other hand, Bob cannot test if a person that is not his friend is a friend of Alice because of the blindness of the signature. He does not know the hash values of the signatures of Alice's friends.

For extra privacy, Alice and Bob could still use a keyed hash function, together with blind signatures.

As mentioned already, another solution for the problem is to use any protocol for private set intersection [8]. Usually, these protocols reveal also what elements belong to the intersection. In our setting this means that Alice and Bob also learn who the common friends are. There are also multiparty protocols for only finding out how many elements there are in the intersection.

And lastly, if Alice and Bob have access to a trusted third party, they can just send the names of their friends to the trusted third party and the trusted third party will tell them who are their common friends or how many common friends they have. This method can also be used by sending keyed hash values of identities instead of identities themselves. Then the trusted party would not find out who the common friends are [11].

A comparison of these solutions is presented in Table I. We have compared our proposed solution against four solutions mentioned above: the naive solution where Alice sends hashes of her friends to Bob, the solution where Alice and Bob together agree on a key and use keyed hash function, the solution where Bob signs Alice's friends before hashing and a general PSI solution. We have listed if Bob can use exhaustive

search to find all friends of Alice and if Alice can find out common friends of Bob and Carol after running the protocol with both of them. We have also compared the relative speeds of the solutions and listed what is the operation that has the largest impact on the running time.

IV. Our protocol

In the following we present our protocol for Alice and Bob to find (the number of) their common friends. Note that this protocol can be used for other purposes as well. The protocol can be modified for problems where the sets of parties are something else than their friends. To protect the privacy of the parties against third parties, we assume that communication between Alice and Bob is done using a secure channel.

We assume that both Alice and Bob have 1024 friends and each has calculated the hash values (of 160 bits) of names of their friends. If they have less than 1024 friends, then they can use random dummy values to expand their number of friends to 1024. Note that it is extremely unlikely that two users would choose same dummy value, if the selection is done randomly.

Algorithm 1 Protocol for discarding prefixes of length k that the common friends do not have

There are 2048 different hash value prefixes of length k bits left.

0: Step 1; Initiator chooses 512 of these that are not prefixes of the hash values of their friends and tells Responder to discard all hash values that have these prefixes.

0: Step 2; Responder chooses another 512 of these that are not prefixes of the hash values of their friends and tells Initiator to discard all hash values starting with these prefixes.

There are 2048 different hash value prefixes of length $k + 1$ bits left.

Algorithm 2 Protocol for finding the common friends

0: Initiator := Alice; Responder := Bob;

0: For k = 11 to 30;

- Run one round of Algorithm 1 with parameter k;
- Switch roles of Initiator and Responder

0: Alice checks whom of her friends do not have discarded prefixes

0: Bob checks whom of his friends do not have discarded prefixes

Our protocol is described in Algorithm 1 and Algorithm 2. The information flow between the two parties is described in Fig 1. The parties continue running the algorithm for, e.g., 20 rounds.

The protocol can be adapted to the case where the maximum number of friends for each party is larger or smaller than 1024. It is important that there are always enough prefixes left so that Alice and Bob can discard their shares of them. If we assume that there are 2048 prefixes in the beginning. Alice can discard 512 them and still leave all 1024 prefixes that correspond to the hashes of her friends. After this there are 1536 prefixes remaining and Bob can discard 512 them and still leave all 1024 prefixes that correspond to the hashes of his friends. Then these 1024 prefixes can be expanded to 2048 longer prefixes and the algorithm can continue.

If we allow the parties to have maximum of 128 friends then we could start from prefixes of length 8. There are 256 of these. Alice first discards 64 of these and Bob then discards another 64. There are 128 remaining and these can be expanded to 256 prefixes of length 9.

If we allow the parties to have maximum of 1'048'576 friends then we could start from prefixes of length 21. There are 2'097'152 of these. Alice first discards 524'288 of these and Bob then discards another 524'288. There are 1'048'576 remaining and these can be expanded to 2'097'152 prefixes of length 22.

When the protocol is finished, Alice can count how many of her friends have hash values that have prefixes that have not been discarded. This is the number of the common friends that the protocol gives. Note that Alice can also learn who the common friends are. Also Bob can count how many of the hash values of his friends have prefixes that have not been discarded and who these friends are.

We have pictured the progress of the algorithm in Fig 2. In this figure the algorithm starts from the prefixes of length 5. This means that there are 32 possible prefixes. These are the sectors on the innermost circle. We assume that Alice and Bob both have 8 friends. The hash values of their three common friends are pictured as black lines. The five friends of Alice that are not friends of Bob are pictured by dashed lines and the five friends of Bob that are not friends of Alice are presented by dotted lines.

On the first round Alice has chosen eight prefixes to discard. These are the eight lighter grey sectors on the innermost circle. Note that three friends of Bob match three of these prefixes and Bob learns that they are not friends of Alice.

On the first round Bob discards eight prefixes. These are the eight dark grey sectors on the innermost circle. He also discards eight prefixes of length 6 on the second round. These are the sectors in the figure where the innermost circle is white but the next circle is dark grey. After this, Alice learns that four out of her friends are not friends of Bob.

The algorithm is continued until covering prefixes of length 9, after which three friends of Alice and also three friends of Bob remain. These are actually the common friends but, of course, after running the protocol to this point, Alice and Bob just know for sure that their common friends are among these three. If they would run the protocol for more rounds, they get more and more convinced that all three are indeed common friends.

V. Analysis

In this section we analyze the accuracy of the solution, its privacy properties and its efficiency, from both computation and communication points of view.

A. Accuracy analysis

In the following we estimate for how many rounds the protocol needs to be executed before the result is accurate enough. If the number of rounds is not sufficiently large there

TABLE I. COMPARISON OF PROPOSED SOLUTIONS

	Naive hash	Keyed hash	Blind signature	PSI	Our solution
Secure against exhaustive search	-	-	+	+	+
Secure against comparing two runs with different partners	-	+	+	+	+
Fast	+	+	-	-	(+)
Speed bottleneck	hash	hash	signing	varies	number of communication rounds

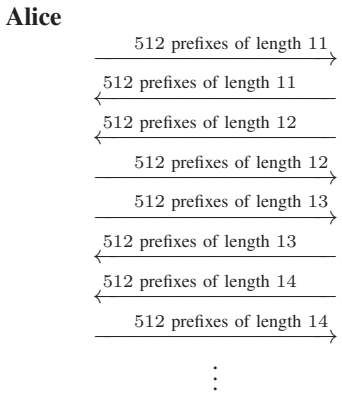

Fig. 1. The communication in the algorithm.

is a chance that the result is larger than the actual number of common friends. Alice could interpret this as some of her friends being common friends even if this is not the case. And, vice versa, it is possible that Bob would think that some of his friends are common friends even if this is not the case. Moreover, it can happen that the result Alice has is not the same as Bob has.

After the first round, for Bob, about three quarters of his friends that are actually not Alice's friends still remain as candidates for common friends. For Alice, about two thirds of her friends that are actually not Alice's friends still remain as candidates for common friends.

This is because, for any round with parameter k, the prefixes of all of Initiator's friends are among $2^k - 2^{k-2}$ bit strings. Responder chooses 2^{k-2} of those to be discarded. That means $\frac{2}{3}$ of Initiator's friends that are not Responder's friends remain as candidates for common friends.

But then the roles change for the next round which means reduction for Alice is with factor of $\frac{3}{4}$, while for Bob it is with $\frac{2}{3}$.

After the second round, for Bob, about $\frac{3}{4} \cdot \frac{2}{3} = \frac{1}{2}$ of his friends that are not Alice's friends remain as candidates for common friends. For Alice, still about $\frac{2}{3} \cdot \frac{3}{4} = \frac{1}{2}$ of her friends that are not Bob's friends remain as candidates for common friends.

After the fourth round, about $\left(\frac{1}{2}\right)^2 = \frac{1}{4}$ of Bob's friends that are not Alice's friends remain as candidates for common friends and about $\frac{1}{4}$ of Alice's friends that are not Bob's friends remain as candidates for common friends.

We conclude, that after $2s$ rounds, for Bob (resp., Alice), about $\left(\frac{1}{2}\right)^s$ of Bob's (Alice's) friends that are not Alice's (Bob's) friends still seem as candidates for common friends.

If we assume that both Alice and Bob have at most 1024 friends then after 20 rounds the error in the count of common friends is in average at most one friend for both Alice and Bob. From another point of view, after 22 rounds, the probability that the number of common friends is correctly calculated is over 60%.

There is a small probability that a hash of one of the friends of Alice has a very long common prefix with a hash of one of the friends of Bob even when these are not common friends In this case the algorithm would return that there is an extra common friend if there are not enough rounds to reach to longer prefixes than the common prefix.

If we consider the prefixes of length $k > 20$ then all (or almost all) 1024 friends of Alice have a unique prefix. This means that the probability of a any other friend of Bob having the same prefix is $\frac{1024}{2^k}$. The number of friends of Bob that are not friends of Alice is at most 1024. Thus the expected number of these having the same prefix than any friend of Alice is smaller than

$$1024 \cdot \frac{1024}{2^k} = 2^{20-k}.$$

For example, starting with prefixes of length 11 and continuing until discarding prefixes of length 30, the probability of any friend of Bob that is not a friend of Alice still having a same prefix as a friend of Alice is smaller than $2^{-10} \approx \frac{1}{1000}$.

B. Privacy analysis

In this section we discuss the amount of information that may be leaked while running the protocol.

On the first round Alice asks to discard $\frac{1}{4}$ of all possible hash values and reveals that none of her friends have these hash values.

On the second round Alice asks to discard $\frac{1}{8}$ of all possible hash values and reveals that none of her friends have these hash values.

On the third round Alice asks to discard $\frac{1}{16}$ of all possible hash values and reveals that none of her friends have these hash values.

After a few rounds Alice has asked to discard almost half of all possible hash values and has revealed that none of her friends have these hash values.

On the first round Bob asks to discard $\frac{1}{4}$ of all possible hash values and reveals that none of his friends have these hash values.

On the second round Bob asks to discard $\frac{1}{8}$ of all possible hash values and reveals that none of his friends have these hash values.

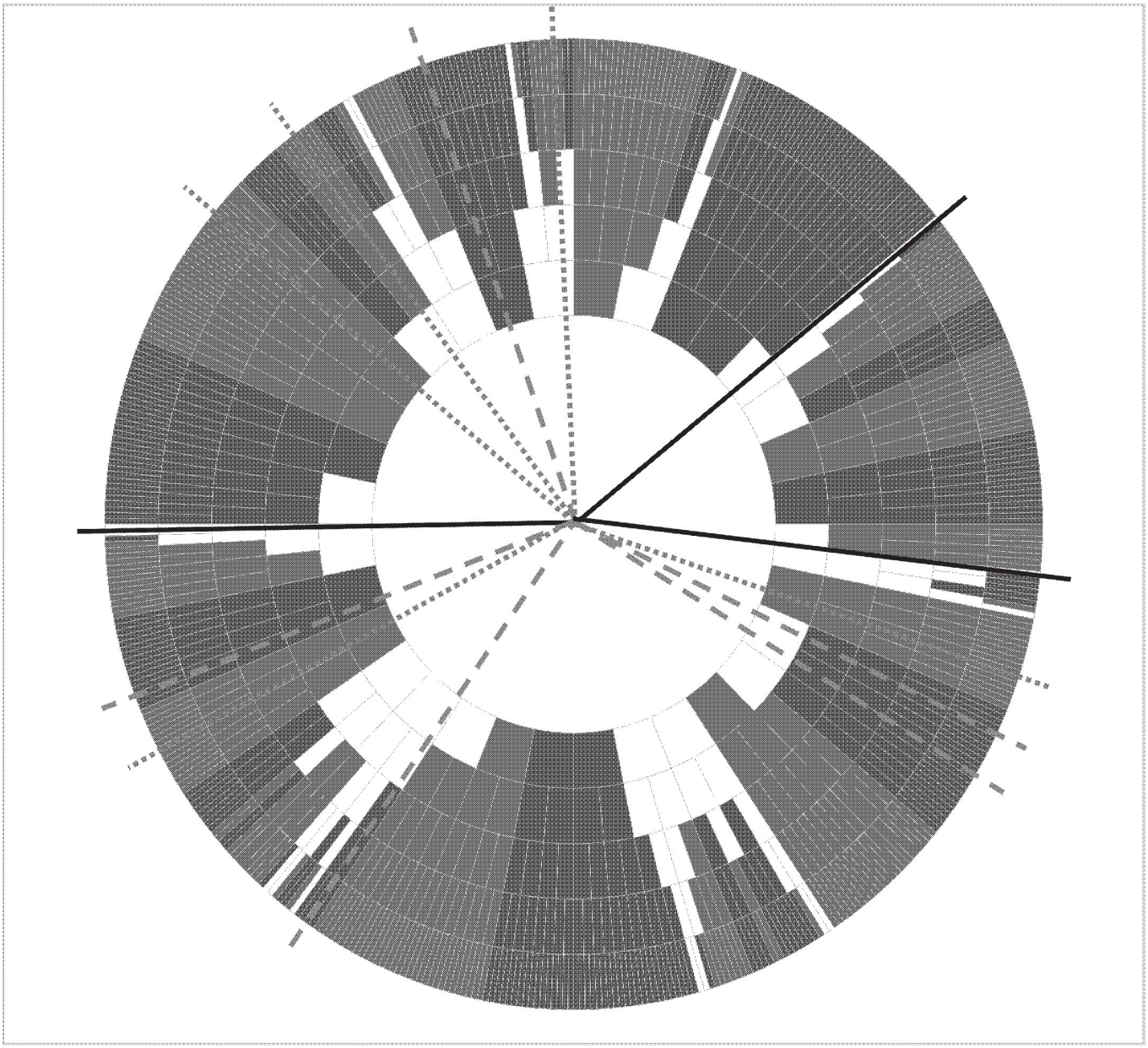

Fig. 2. The result of the algorithm. The range of hash function values is depicted as a compass rose In this picture Alice and Bob both discard 8 prefixes of each length.

After a few rounds Bob has asked to discard almost half of all possible hash values and has revealed that none of his friends have these hash values.

If the protocol is run for many rounds, Bob learns almost $\frac{1}{2} \cdot 2^{160}$ hash values that do not belong to Alice's friends, the remaining values may belong to friends of Alice. However, Bob does not learn which of the remaining hash values belong to the friends of Alice.

The same is true for Alice about the friends of Bob.

It could be said that, in principle, approximately half of all people in the world are revealed to Bob to not belong to the friends of Alice. Bob could be interested in finding whether a certain person is Alice's friend although that person of interest is not among Bob's friends. Bob could then avoid discarding that person's hash prefixes. Then eventually Alice is sort of "forced" to discard that person if the person is not her friend. But this setting means that Bob does not follow the protocol correctly. In other words, Bob's actions are equivalent to using false set of friends as input to the protocol. As mentioned before, a protocol of the sort we are discussing in this paper cannot protect against using false input values.

C. Communication cost

In this section we estimate how many bits of data Alice and Bob need to sent to each other during the execution of

the protocol. We also present a method to send only 2048-bit incidence vectors instead of 2048 prefixes of hashes.

On the first round, Alice has a list of all 2048 prefixes of length 11. She can communicate her choice of the prefixes to be discarded by sending an incidence vector of 2048 bits to Bob. She orders the prefixes in the lexicographical order and for each prefix sets the corresponding bit in the vector to be 1 if this prefix is to be discarded and 0 otherwise. For the next round she makes a new list of 1536 prefixes of length 11 that are available after her choice of 512 prefixes and forgets the old one.

Bob can also order the 2048 prefixes of length 11 in the lexicographical order and, after receiving the incidence vector from Alice, he learns her choice.

On the first round, Bob has 1536 prefixes left. He chooses 512 to discard and saves this choice to an incidence vector of length 1536, as described before. On the second round he considers the 2048 prefixes of length 12. As before, he orders them in the lexicographical order and saves his choice of prefixes to discard in an incidence vector of length 2048. Bob sends both of these vectors to Alice, who first learns which vectors of length 11 are to be discarded and thus, what are the 2048 prefixes of length 12 that remain. She can arrange these in the lexicographical order and learns from the second vector Bob sent what prefixes he wants to discard.

Alice and Bob can continue this way on the following rounds. Even when the prefixes are getting longer and longer, the incidence vectors they send to each other on each round are always of length 1536 and 2048. Thus, on each round Alice sends to Bob or Bob sends to Alice less than 4000 bits of information.

The ordering of the new, one bit longer prefixes in the lexicographical order is quite easy because all the prefixes are always in the lexicographical order. From each remaining prefix two longer prefixes are generated by appending 0 or 1 to the end.

D. Time cost and comparison with other solutions

We have implemented the protocol in the case that both Alice and Bob have at most 1024 friends and execute the protocol for 20 rounds. The calculations took about 3ms in total for Alice and Bob on an old Windows PC. The time that the communication between Alice and Bob takes depends, of course, on the properties of the network Alice and Bob are using. In order to estimate the delays of running the algorithm on Android devices and especially the HELIOS platform we have implemented a test scenario for this purpose. The tests were done on Android phones (Nokia 8.1) running a HELIOS platform (see https://github.com/helios-h2020/) test application on 4G connectivity. HELIOS p2p direct messaging feature was used by adding support to a new protocol for finding out common friends between two HELIOS peers that both support this new protocol. The implementation allows to configure the amount of rounds to be executed, thus allowing to adjust the confidence and privacy-level as wanted. The HELIOS messaging library adds its own internal message headers on top of the actual payload of the protocol, i.e., either 2048 or 4096 bits. The protocol payload messages were random in this test scenario, thus measuring the delays of the device, used network and the platform. The messages were end-to-end encrypted between the two applications with the messaging module of HELIOS. During the tests, the applications used a HELIOS relay to be able to communicate to each other, which obviously adds milliseconds to the execution time. Overall, combining the measured times, we estimate that using HELIOS platform and two Android phones the communication typically takes 1700ms.

We now compare the time cost against other solutions, in particular naive hashing solution, where hashes of friends of Alice are sent to Bob for comparison, and the Oblivious Transfer using Random Bloom Filter solution by Pinkas et al [9]. The latter seems to be the fastest of secure PSI protocols in the comparison made in another paper by Pinkas et al. [10]. We compare our protocol against the numbers in Table 9 of the latter paper.

We start from the LAN setting where communication is very fast. The naive hashing solution takes 1ms for 256 elements and 3ms for 4096 elements. For these set sizes the protocol by Pinkas et al. takes 95ms and 346ms. To make comparisons we continue by indirect reasoning about the potential speed of our protocol in a similar setting. From the numbers we can deduct that latency between Alice and Bob is smaller than 1ms and thus for our protocol the transfer of data would take less than 21ms. So, in total our protocol would finish in 24ms for sets of 1024 elements.

Next we consider the WAN setting where round trip of transmissions between Alice and Bob takes 97ms. It is mentioned in the table that naive hashing solution takes 51ms for sets of 256 elements and 119ms for sets of 4096 elements. The time requirement of the protocol by Pinkas et al. is 968ms for sets of 256 elements and 3863ms for sets of 4096 elements. Similarly as above, we continue by speculating how our protocol would perform in the WAN setting. Our protocol needs 10 and half round trips so the time for sets of 1024 elements would be 10.5 times 97ms, in addition to 3ms that is needed for computations on both sides. This is 1022ms in total.

Thus our solution seems to be faster than the solution by Pinkas et al. in both LAN and WAN cases. Our solution is slower than the naive hashing but it provides better privacy.

VI. IMPROVEMENTS

We can use probability theory to estimate how close the output of the protocol is to the actual number of common friends. In this way we can reduce the number of rounds in the protocol from 20 to 10 and still get a good estimate on the number of common friends. On the other hand, we lose the ability to accurately find out who the common friends are. For Alice, her friends are divided into two sets, one set consists of friends that are surely not common friends and another set in which the friends may be common friends. Bob also learns that his friends can be divided into two sets in a similar fashion.

Let us consider the situation from the Alice's point of view. On the second step of the first round Bob discards one third of the prefixes. Before this, none of the friends of Alice are discarded. There is a possibility of $\frac{1}{3}$ for discarding a hash

value belonging to a friend of Alice that is not a friend of Bob. In the first step of the second round Bob discards one quarter of the remaining longer prefixes. In total, after the second round, about

$$\frac{1}{3} + \frac{2}{3} \cdot \frac{1}{4} = \frac{1}{2}$$

of all Alice's friends that are not friends of Bob have discarded prefixes.

About half of those friends of Alice that are not friends of Bob but remain after first two rounds are discarded on third and fourth round.

We can calculate that after $2s$ rounds, the probability that a friend of Alice that is not a friend of Bob has a discarded prefix is

$$p = \sum_{i=1}^{s} \frac{1}{2^i} = 1 - \frac{1}{2^s}.$$

Alice knows how many of her friends have hash values with discarded prefixes. We denote by a_s the number of Alice's friends that have discarded prefixes after $2s$ rounds and by a the number of Alice's friends that are not friends of Bob and thus would be discarded if the protocol would be run long enough. We have

$$a_s \approx pa$$

and thus Alice can calculate that

$$a \approx \frac{a_s}{p} = \frac{2^s}{2^s - 1} a_s.$$

Alice can now calculate an estimation for the number of her common friends with Bob by subtracting this number from her total number of friends.

We can do a similar analysis from the point of view of Bob. On the first round one quarter of friends of Bob that are not friends of Alice are revealed not to be friends of Alice. And on the second round one third of the remaining friends of Bob that are not friends of Alice are revealed not to be friends of Alice.

Thus, after these transmissions, about

$$\frac{1}{4} + \frac{3}{4} \cdot \frac{1}{3} = \frac{1}{2}$$

of all friends of Bob that are not friends of Alice are revealed to Bob.

Therefore, if after $2s$ rounds Bob has learned b_s friends of his that are not friends of Alice, the total number of friends of his that are not friends of Alice is about

$$\frac{2^s}{2^s - 1} b_s.$$

We now give an error estimation for this approximation. The number of friends of Alice that are not friends of Bob but are not yet revealed to not be friends of Bob follow binomial distribution $B(n, p)$ [3]. Here n is the number of friends of Alice that are not friends of Bob, that we have denoted earlier by a, $p = 1 - \frac{1}{2^s}$ and $2s$ is still the number of rounds. The variance for the number of friends of Alice that are not yet revealed to not be friends of Bob is

$$np(p-1) = a\frac{2^s - 1}{4^s}$$

and the standard deviation is

$$\sqrt{np(p-1)} = \sqrt{a}\frac{\sqrt{2^s - 1}}{2^s}.$$

Thus, if Alice has 100 friends that are not friends of Bob, after 10 rounds of the protocol, the standard deviation is 1.7. This means, approximately, that with probability 95% the result of the algorithm is off by at most $2 \cdot 1.7 \approx 4$ friends.

Using this method the parties can reduce the number of rounds in the protocol and still get a good estimate on the number of common friends.

VII. Conclusion and future work

We have considered the problem of finding the intersection of two sets, in a privacy friendly manner. Our motivation for this is a case that has arisen in HELIOS social media platform. Two parties want to find out how many common friends they have in a privacy preserving way.

We have described our solution that is efficient and does not reveal too much information about the private sets and thus is more suitable for social media platforms than previous work. This paper includes the analysis of the privacy and efficiency of the protocol. We have also implemented the protocol and showed that our solution is faster than prior solutions meeting the same privacy level.

Several modifications to the protocol that are suitable for solving different versions of the problem are also discussed.

Future work includes implementing the full protocol for Android devices and making it eventually a part of the HELIOS platform.

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] Bellare, M., Canetti, R., & Krawczyk, H. (1996, August). Keying hash functions for message authentication. In Annual international cryptology conference (pp. 1-15). Springer, Berlin, Heidelberg.

[2] Chaum, D. (1983). Blind signatures for untraceable payments. In Advances in cryptology (pp. 199-203). Springer, Boston, MA.

[3] Clapham, C., Nicholson, J., & Nicholson, J. R. (2014). The concise Oxford dictionary of mathematics. Oxford University Press.

[4] Guidi, B., Kapanova, K. G., Koidl, K., Michienzi, A., & Ricci, L. (2020). The contextual ego network p2p overlay for the next generation social networks. Mobile Networks and Applications, 25(3), 1062-1074.

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

[6] Hiltunen, J., & Kuusijrvi, J. (2015, August). Trust metrics based on a trusted network element. In 2015 IEEE Trustcom/BigDataSE/ISPA (Vol. 1, pp. 660-667). IEEE.

[7] Korzun, D., & Gurtov, A. (2012). Structured peer-to-peer systems: fundamentals of hierarchical organization, routing, scaling, and security. Springer Science & Business Media.

[8] Meadows, C. (1986, April). A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In 1986 IEEE Symposium on Security and Privacy (pp. 134-134). IEEE.

[9] Pinkas, B., Schneider, T., & Zohner, M. (2014). Faster private set intersection based on OT extension. In 23rd USENIX Security Symposium (USENIX Security 14) (pp. 797-812).

[10] Pinkas, B., Schneider, T., & Zohner, M. (2018). Scalable private set intersection based on OT extension. ACM Transactions on Privacy and Security (TOPS), 21(2), 1-35.

[11] Ramezanian, S., Meskanen, T., & Niemi, V. (2021, July). Multi-party Private Set Operations with an External Decider. In IFIP Annual Conference on Data and Applications Security and Privacy (pp. 117-135). Springer, Cham.

[12] Rogaway, P., & Shrimpton, T. (2004, February). Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In International workshop on fast software encryption (pp. 371-388). Springer, Berlin, Heidelberg.

[13] 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.

[14] Yao, A. C. (1982, November). Protocols for secure computations. In 23rd annual symposium on foundations of computer science (sfcs 1982) (pp. 160-164). IEEE.

[15] Zhao, C., Zhao, S., Zhao, M., Chen, Z., Gao, C. Z., Li, H., & Tan, Y. A. (2019). Secure multi-party computation: theory, practice and applications. Information Sciences, 476, 357-372.