Building a Privacy-Preserving Semantic Overlay for Peer-to-Peer Networks

Niels Zeilemaker [#1], Zekeriya Erkin [*2], Paolo Palmieri [#3], Johan Pouwelse [#4]

[#] *Parallel and Distributed Systems Group, Delft University of Technology*
Mekelweg 4, 2628 CD, Delft, The Netherlands
[1] *n.s.m.zeilemaker@tudelft.nl* [3] *p.palmieri@tudelft.nl* [4] *j.a.pouwelse@tudelft.nl*

[*] *Information Security and Privacy Lab, Delft University of Technology*
Mekelweg 4, 2628 CD, Delft, The Netherlands
[2] *z.erkin@tudelft.nl*

***Abstract*—Searching a Peer-to-Peer (P2P) network without using a central index has been widely investigated but proved to be very difficult. Various strategies have been proposed, however no practical solution to date also addresses privacy concerns.**

By clustering peers which have similar interests, a semantic overlay provides a method for achieving scalable search. Traditionally, in order to find similar peers, a peer is required to fully expose its preferences for items or content, therefore disclosing this private information. However, in a hostile environment, such as a P2P system, a peer can not know the true identity or intentions of fellow peers.

In this paper, we propose two protocols for building a semantic overlay in a privacy-preserving manner by modifying existing solutions to the Private Set Intersection (PSI) problem. Peers in our overlay compute their similarity to other peers in the encrypted domain, allowing them to find similar peers. Using homomorphic encryption, peers can carrying out computations on encrypted values, without needing to decrypt them first.

We propose two protocols, one based on the inner product of vectors, the other on multivariate polynomial evaluation, which are able to compute a similarity value between two peers. Both protocols are implemented on top of an existing P2P platform and are designed for actual deployment. Using a supercomputer and a dataset extracted from a real world instance of a semantic overlay, we emulate our protocols in a network consisting of a thousand peers. Finally, we show the actual computational and bandwidth usage of the protocols as recorded during those experiments.

I. Introduction

Over recent years numerous papers have been published on designing systems which provide fully decentralized search in *Peer-to-Peer* (P2P) networks. However, searching a vast network of peers sharing numerous items has proved to be difficult. This is, to a large extent, determined by the manner in which peers are organized.

Traditionally, P2P networks can be divided into four classes: structured, unstructured, hybrid, and semantic networks. Structured networks allow peers to efficiently locate items, but keeping information stored in the network up to date is costly. Unstructured networks do not suffer from this overhead, but can not provide a method for efficiently locating items. Hybrid networks can provide both, by electing superpeers to index and locate items of ordinary peers. Lastly, semantic networks allow peers to locate items by connecting them to peers with similar interests. This greatly reduces the number of peers contacted during a search (as a message is not forwarded), but at the same time reduces the number of items that can be found.

Even though the aforementioned methods are well researched, privacy considerations are not addressed. Moreover, only two out of the four approaches mentioned can be considered for practical purposes: a hybrid or a semantic network, as both structured and unstructured networks suffer from substantial overheads.

Concerning privacy, a hybrid network, such as the one used by Kazaa [1], has a significant drawback as it requires peers to fully expose the items they are sharing with the superpeers. This violates the privacy of the peers, as the superpeers are trusted with this information without knowing true identities or intentions of these peers. Furthermore, by sending a query to a superpeer, and thus trusting it, a query is then forwarded to a large number of unknown peers in the superpeer-network. All of them are implicitly trusted with the privacy sensitive material which is present in the search queries [2].

Current deployed semantic networks have similar drawbacks [3], [4], as peers exchange their preferences in plain text with randomly selected peers, allowing peers to compute the similarity between them. We define *preferences* as the items for which we want to compute similarity, e.g. a list of favorite movies, jokes, etc. Exchanging these lists also exposes the preferences of a peer to all. In this paper, however, we improve upon this by extending existing cryptographic protocols, which allow us to find common elements in two private sets without exposing the sets themselves.

This problem is known in cryptography as the Private Set Intersection (PSI) problem [5]. In this paper, we focus on the PSI problem within the P2P context to find similar peers in a distributed network. There are a number of existing PSI protocols for different privacy requirements and security assumptions. Unfortunately, while previous work provably

protects private data, the underlying cryptographic approaches have not seen implementation in a P2P environment, due to their high overhead, either in computation, bandwidth, or both.

A. Contribution

In this paper, we propose the first practical, fully implemented method for building a semantic overlay in a privacy-preserving manner. We do so by designing two protocols that allow for the computation of similarity between peers while preserving the privacy of each peer's preferences. The protocols adapt solutions from the PSI problem in cryptography to fit the needs of large P2P networks. We thoroughly test the performance of the implementations in a real network by emulating an instance with a thousand peers. Using a dataset extracted from an existing semantic overlay, we show that our solutions are scalable and efficient in terms of both computation and bandwidth. Most importantly, they fit nicely within the real world limitations of P2P networks, since they do not require any trusted third party or assume the possibility of a connection between any two peers.

B. Related Work

Structured networks, such as Chord [6], Kademlia [7], or Pastry [8], allow peers to efficiently locate information by predefining where items should be stored and how items can be found. In their approaches, keywords are converted into hashes and then used to route a query to the responsible peer. This responsible peer, maintains a list of peers which have matching items for the hashed keywords.

However, the costs associated with keeping such a network up to date are prohibitively high. For each keyword by which a peer wants its items to be found, it needs to "announce" this to the responsible peer, i.e., in the case of a file-sharing system, a peer needs to split the name of a file into separate keywords and send an announce-message to each peer responsible for that keyword. Hence, multiple announces are required for each shared file causing a substantial portion of the bandwidth available at a peer to be used. Moreover, as peers in P2P networks leave and join the system at a high rate (*churn*), peers need to re-announce their items on a regular basis.

Unstructured networks require much less maintenance. However, in contrast to structured networks, locating items is much more difficult in an unstructured network. Naive approaches, such as implemented by Gnutella in 2000 [9], caused each search query to be sent to almost all the peers in the network (*flooding*), resulting in a substantial overhead.

Later in 2001, Kazaa improved upon this concept by creating a hybrid network [1]. Their network consists of two types of peers; superpeers and ordinary peers. Superpeers index the ordinary peers connected to them and thereby dramatically reduce the number of peers contacted during each search.

In semantic networks, peers are clustered by their interest for the same content. When issuing a search query, a peer sends a message to the peers that have similar interests. In this way, the number of peers contacted during a search is greatly reduced (as a message is not forwarded), but also causing that not all items can be found (as scope is limited). Various clustering approaches have been suggested, which usually either group peers into a fixed number of clusters, or let peers independently find their most similar neighbors.

The challenge, as described in the introduction, of finding common elements in two sets without revealing the elements of the sets, falls into the more general category of comparing private information from two parties without leaking the information itself. It can therefore be formulated and solved in terms of two-party secure function evaluation.

The PSI problem has been specifically discussed in various recent works, and is generally solved by protocols based on the homomorphic properties of an encryption scheme (most notably, ElGamal or Paillier). In 2004, Freedman *et al.* presented a cryptographic protocol based on oblivious polynomial evaluation and the Paillier cryptosystem [5]. The work by Kissner and Song also focuses on polynomial evaluation, but the authors propose protocols for a number of set operations, including unions, in a multi-party environment [10]. The most efficient set of polynomial protocols to date is instead the one proposed by Kiayias and Mitrofanova in [11], and is based on an ElGamal encryption scheme. Hazay and Lindell proposed a different approach to the PSI problem [12], which they solve using oblivious pseudo-random functions.

Another useful cryptographic tool in secure two-party evaluation protocols is identity based encryption [13]. It has been used for the first time for PSI by De Cristofaro *et al.* in [14]. Their assumption is that the first party receives an authorization from a trusted entity before being able to make queries to the other party's set using identity based encryption. Later, this line of research further progressed in terms of efficiency [15], [16], [17]. In this paper, however, we do not assume any trusted third party, and therefore all the communication in the protocols we propose happens between the two set owners, namely Alice and Bob.

II. Privacy-Preserving Protocols

We propose two protocols that allow a peer in a P2P network to calculate its similarity to other peers. In this section, we assume a setting in which the first peer, Alice, has a set of items $\mathcal{I}_A$. Alice queries a second peer, Bob (which has a set of items $\mathcal{I}_B$), in order to compute the number of common items in the two sets without disclosing the sets themselves. Since we are interested in finding the similarity between two peers, we use the cardinality of the intersection between the two sets as a similarity score $sim_{A,B}$.

Both protocols are based on additively homomorphic encryption, namely the Paillier cryptosystem [18], that allows to add two plaintext messages by operating on their encryptions:

$$\mathcal{D}_{sk}(\mathcal{E}_{pk}(m_1) \times \mathcal{E}_{pk}(m_2)) = m_1 + m_2 \,. \tag{1}$$

where m_1 and m_2 are plaintexts and $\mathcal{E}_{pk}()$ and $\mathcal{D}_{sk}\,()$ the encryption and decryption functions, respectively. It follows that, given a constant c, the following also holds:

$$\mathcal{D}_{sk}(\mathcal{E}_{pk}(m)^c) = m \times c \,. \tag{2}$$

The Paillier cryptosystem is probabilistic, in the sense that a fresh random parameter is introduced as input in each encryption operation.

A. Private Set Intersection Protocol I

The first protocol is based on the inner product of vectors. In order for the algorithm to work, we make the assumption that there is a global list of all possible items, $\mathcal{I}_G = \{\mathcal{I}_{G,1}, \ldots, \mathcal{I}_{G,M}\}$, of size M and this list is available to all peers. Based on that, each peer N creates a binary vector $\Gamma_N = \langle \gamma_1, \ldots, \gamma_M \rangle$, for the items in the global list. If node N has item $\mathcal{I}_{G,i}$ in his set $\mathcal{I}_N$, then $\gamma_{N,i} = 1$ else $\gamma_{N,i} = 0$.

Protocol 1: Alice selects the secret key parameters for a Paillier encryption scheme, and publishes the respective public key and parameters. The communication proceeds as follows:

1) Alice encrypts each term in her list using her public key: $\mathcal{E}_{pk_A}(\gamma_{A,i})$ for $1 \leq i \leq M$, and sends them to Bob.
2) Bob computes the similarity score by using the homomorphic property of the encryption scheme as follows:

$$\mathcal{E}_{pk_A}(sim_{A,B}) = \mathcal{E}_{pk_A}\left(\sum_{i=1}^{M} \gamma_{A,i} \cdot \gamma_{B,i}\right) := \prod_{i=1}^{M} \mathcal{E}_{pk_A}(\gamma_{A,i})^{\gamma_{B,i}} \quad (3)$$

   Bob then sends $\mathcal{E}_{pk_A}(sim_{A,B})$ to Alice.
3) Alice obtains the similarity score $sim_{A,B}$ after decryption.

It could be argued that step 2 is computationally expensive due to modular exponentiation. However, in practice this computation can be performed with minimum effort. In fact, the values of $\gamma_{B,i}$ are either 1 or 0. Therefore, Bob only needs to multiply Alice's encrypted values for i values where $\gamma_{B,i} = 1$ (this corresponds to adding in the plain text domain) and re-randomize the final encryption. Thus, the computation becomes:

$$\prod_{i=1,\gamma_{B,i}=1}^{M} \mathcal{E}_{pk_A}(\gamma_{A,i}) \ . \quad (4)$$

This simplification results in a very efficient protocol in terms of computation: Alice performs M encryptions, Bob performs M modular multiplications in the worst case, and Alice performs 1 decryption. The bandwidth requirement, on the other hand, is linear in the number M as Alice sends M encryptions to Bob.

B. Private Set Intersection Protocol II

Contrary to the previous protocol, this approach does not assume the existence of a global list of possible items, and is therefore suited for settings in which such an assumption is unrealistic. Here we compute the cardinality of the intersection of two sets using multivariate polynomial evaluation. In the following we propose a modification of the protocol introduced by Freedman *et al.* [19] that follows this technique.

Protocol 2: Alice selects the secret key parameters for a Paillier encryption scheme, and publishes the respective public key and parameters. Communication between the parties proceeds as follows:

1) Alice builds a polynomial having roots in each of the items contained in her set $\mathcal{I}_A$. That is, she computes the $n+1$ coefficients $\alpha_0, \ldots, \alpha_n$ of the polynomial

$$f(x) = \alpha_0 + \alpha_1 x + \alpha_2 x^2 + \ldots + \alpha_n x^n \quad (5)$$

   for which $f(\mathcal{I}_{A,i}) = 0$ for any item in her set.
2) Next, Alice encrypts the coefficients and sends them to Bob $\mathcal{E}_{pk_A}(f(x))$.
3) Bob uses the homomorphic properties of the encryption scheme to evaluate the polynomial for each item in his set Γ_B, and multiplies each result by a fresh random number r_i, obtaining $\mathcal{E}_{pk_A}(r_i \cdot f(\mathcal{I}_{B,i}))$.
4) Bob adds all evaluated polynomials to a list, permutates the order, and sends it back to Alice.
5) After receiving the list of evaluated polynomials from Bob, Alice decrypts each ciphertext and counts the number of 0's she received. The ciphertexts decrypt to 0 in case of an item in the intersection $\mathcal{I}_A \cap \mathcal{I}_B$, or to a random value otherwise. Therefore, $sim_{A,B}$ is the number of 0's decrypted by Alice.[1]

The polynomial required in step 1 can be obtained by simple multiplication of the factors $(x - \mathcal{I}_{A,i})$. 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.

Computationally this protocol is more expensive than Protocol 1. Alice needs to perform $|\mathcal{I}_A|$ encryptions, Bob needs to perform $|\mathcal{I}_B| \cdot |\mathcal{I}_\mathcal{A}|$ modular multiplications, and Alice needs to perform $|\mathcal{I}_B|$ decryptions. The required bandwidth for the protocol is linear in the number of items both Alice and Bob have in their sets.

However, the bandwidth efficiency of this protocol can be improved by adopting a partitioning scheme. By reducing the degree of the polynomial, we increase the bandwidth efficiency as we can use a smaller keysize and increase the size of the universe at the same time. We do so by partitioning the universe (or domain) of possible items into a number of subsets. Each polynomial will then be relative to the items contained in one specific partition only, thus reducing on average the values of the coefficients and therefore the communication cost.

Without a partitioning scheme, a universe in the order of magnitude of millions of items or more (for instance, 2^{24}) and a relatively small set counting 100 items, would require a keysize of 1200 bits. A polynomial with a degree of 100,

[1]Let us observe that, if Bob also adds an encryption of the value of his input $\mathcal{I}_{B,i}$, obtaining therefore $\mathcal{E}_{pk_A}(r \cdot f(\mathcal{I}_{B,i}) + \mathcal{I}_{B,i})$, the result of the computation is the value of the element $\mathcal{I}_{B,i}$ if $\mathcal{I}_{B,i} \in \mathcal{I}_A \cap \mathcal{I}_B$, and a random value otherwise. This would allow us to identify the elements in the intersection set and therefore obtain a private set matching protocol, as done in [19].

would in this case have a least significant coefficient with a maximum possible value close to 2^{2400}. Encrypting such a large coefficient would require a 1200 bit key, as Pallier encryption works over modulo n^2.

Partitioning the universe inserts a step in the protocol before step 1, in which Alice needs to divide her set of items $\mathcal{I}_A$ into multiple subsets. For each subset, or partition, she needs to compute a polynomial and send it to Bob together with her partitioning choice. Note that the number of coefficients Alice sends to Bob does not change. The resulting improvement allows us to accommodate an increase in the size of the universe depending on the expected maximum number of items per partition. E.g. if we expect a maximum number of items per partition of 25, then we can use an universe of size 2^{80} and a 1024 bit key. However, if Alice has more than the maximum number of items for a given partition, she is required to make a selection.

A similar reduction in communication cost can be observed from Bob, as he can remove items from his set $\mathcal{I}_B$ which do not corresponding to the received partitions. However, in order to maximize the benefits without compromising the security of the protocol, the number of distinct partitions should be small enough compared to the number of elements in the universe, but comparable to the number of elements in $\mathcal{I}_A$.

C. Security Discussion

Both protocols protect Alice's privacy by hiding her item set from Bob. The set is first encoded (as a vector in Protocol 1, as the coefficients of a polynomial in Protocol 2), and then transmitted to Bob in encrypted form. Due to the homomorphic properties of the chosen encryption scheme, Bob can not decrypt, but can perform simple operations on the received values. In both algorithms, he computes a function that takes Alice's encrypted values and his own values as inputs, and sends the result to Alice, who decrypts using her secret key. His inputs are hidden thanks to the properties of the powers in the case of the first protocol, and to multiplication by random number in the second. Since the encryption scheme is probabilistic, the same plaintext encrypted twice results in two different outputs. This prevents parties from directly comparing the encrypted values.

We note that in Protocol 2, Alice discloses the number of items that are being compared (by counting the number of coefficients minus the number of polynomials). This allows Bob to decide a minimum number of items below which he will not compute the intersection, thus preventing Alice to run the protocol for a set composed of one item only and learning whether or not Bob possesses that specific item.

Both protocols achieve security in the semi-honest setting [19]. This setting, also known as honest but curious, assumes that the parties do not deviate from the protocol, and therefore includes the case of passive wiretappers, the most common threat to P2P networks. The protocols are however not secure against active adversaries. While an active eavesdropper deploying a man-in-the-middle attack by modifying the messages between the parties can be detected by adding signatures to the messages themselves, the case of a malicious Alice or Bob requires significant modifications to the protocols. In particular, a zero-knowledge proof protocol can be used to prevent Bob from pretending he always has a degree of similarity with Alice [20]. We assume Bob has no interest in pretending not to have a similarity with Alice, since in that case he could just not reply to Alice's query.

A zero knowledge protocol, together with a commitment scheme, is also useful in preventing malicious attacks by Alice [21]. We note however that Protocol 2 already offers some degree of protection against a malicious Alice. In fact, besides improving bandwidth efficiency, another benefit of partitioning is preventing binary search attacks. Under such a scenario, Alice computes multiple polynomials with the same roots (for example 1 polynomial having a root in the first item, 2 polynomials for the second item, 4 for the third and so on). This allows her to identify exactly which items Bob has by counting the number 0's returned by Bob. However, since Bob evaluates a polynomial only at elements in the corresponding partition and replies to only one polynomial for each partition, this attack can not be carried out.

D. Improving Discovery Speed

As introduced in the previous sections, both protocols allow Alice to compute a similarity score $sim_{A,B}$ between her and Bob. However, in order to improve the speed of discovering peers which are similar to Alice we extend these protocols with forwarding. When sending a request to Bob, Bob will not only execute his steps of the protocol, but as well forward the request of Alice to his most similar neighbors. Bob will combine all replies from his neighbors with his own and send it back to Alice, allowing her to discover the similarity not only between her and Bob, but also between her and Bob's neighbors.

The reasoning behind this is that if Alice and Bob are found to be similar, then we can expect Alice to be similar to one of Bob's most similar peers as well. Moreover, using Bob as a proxy is required, as we assume an active connection between her and Bob, but the same can not be assumed between her and Bob's neighbors in a P2P environment, due to firewalls, NAT-routers, etc.

III. Experimental Results

To evaluate the performance of the two outlined protocols, we have deployed 1000 peers on a supercomputer and monitor the time required to find similar peers. In the following paragraphs, we elaborate on both the dataset and experimental setup before discussing the results.

A. Dataset

Our dataset is constructed based on collected information from a real-world P2P file sharing network, called Tribler [4]. This ensures that our experimental results are close as possible to an actual implementation. Peers in the Tribler P2P network construct a semantic overlay in a non-privacy preserving manner. Using a *BuddyCast* message, peers send a list of

TABLE I
DATASETS CREATED FOR PERFORMANCE EVALUATION

	Full dataset	Subset
Peers	74 797	1 000
Items	296 358	7 635
Preferences	1 332 508	25 429
Average #preferences per peer	18	25

their preferences to others in order to compute a similarity among them. We deployed instrumented clients into the Tribler network, which collected these lists. Using the generated files, we then constructed a dataset containing the preferences of the peers in the Tribler network. In total, we collected the preferences of almost 75 000 peers. Those peers preferred 1 332 508 items, with an average of 17.81 items being preferred per peer.

For the experiment, we made a subset and selected 1000 peers at random. Furthermore, we made sure that each peer had at least 10 items, and each item was in the preference lists of at least two peers. This requirement causes every item in the preference list of a peer to overlap with at least one other peer. Full details of both datasets are shown in Table I.

Using the data from a deployed semantic overlay allows us to compare the performance of both protocols without having to resort to a synthetic dataset.

B. Experimental setup

To evaluate the time required to find similar peers we implemented a semantic overlay, in which peers connect to another peer every 5 seconds in a semi-random fashion. After connecting to a new peer, one of the protocols will be used to compute a similarity score. Similarity scores are used to create a list containing the 10 peers, which are most similar to it. A peer then uses this list to maintain an open connection to those 10 neighbors. Related work has shown that maintaining a connection to 10 neighbors yields a good trade-off between the cost of discovering those and the expected hit ratio when performing keyword search [3], [4].

1) Monitoring: Each peer is provided with a list of its preferences (as defined by the subset), and a list of its most similar neighbors. Using this similar neighbor list, we determine if a peer has found its most similar peers or not. This list is generated using the complete subset, i.e. by computing the similarities between all peers. During the experiments, we monitor the peers by comparing the neighbors of each peer to the static list of its most similar neighbors.

Moreover, in our emulations we want to see the differences in terms of performance between an "empty" overlay, which has no connections, and an overlay with existing connections also known as a bootstrapped overlay. We analyze effects of peers in the network being bootstrapped on the speed of discovering similar peers of the non-bootstrapped peers.

2) Infrastructure: During the experiments we use 20 computing-nodes, which all run 50 instances (an instance is an emulation of a peer trying to find similar peers). In this way, we create an actual semantic P2P network on the supercomputer consisting of 1000 peers.

TABLE II
PARAMETERS USED IN EXPERIMENTS

Size of p and q	512 bits
Size of an encrypted value	2048 bits
Number of partitions	256 (1 byte)
Connection interval	5 seconds
Similar neighbors to find	10
Length of preference lists	100

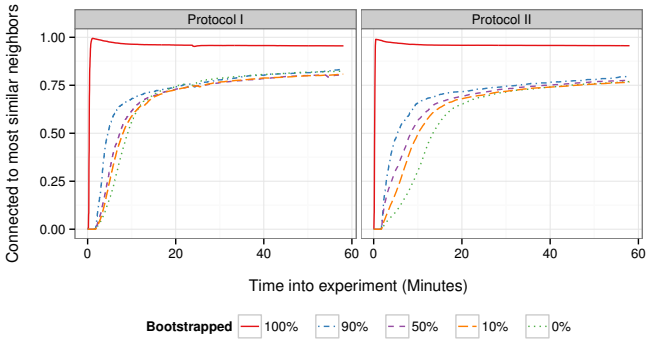

Fig. 1. Speed of discovering the 10 most similar neighbors in a network of 1000 peers. When the network is not completely bootstrapped, the performance of the non-boostrapped peers is shown.

3) Parameters: Table II shows the used parameters during the experiments. Both PSI-C protocols are implemented in Python using GMPY[2] which exposes GMP[3] to Python.

C. Results

1) Discovery speed: Figure 1 shows the speed of finding the most similar neighbors for each protocol in an overlay bootstrapped at different percentages. Comparing both protocols, we can see that the performance difference is minimal. This is to be expected, as both protocols should compute the same cardinality. After 10 minutes, both protocols find more than 50% of all similar neighbors regardless of the level of bootstrapping of the overlay. However, after 10 minutes the discovery speed of the protocols degrades. We believe that this is caused by clustering in the overlay. Although we promote clustering, as we are building a semantic overlay, it is also working against us in discovering peers with a clearly different set of preferences than our current similar neighbors (peers are stuck in a local optimum).

2) CPU consumption: Figure 2 shows the mean CPU time peers spend during the experiment. We use CPU time as the wall time it took to perform the encryption/decryption steps during the 60 minute run as shown above.

From the figure, we can see that Protocol 2 requires much more CPU time compared to Protocol 1, especially while decrypting the received responses from Bob. This is expected, as in Protocol 2, Bob replies with multiple encrypted evaluated polynomials while in Protocol 1 he only returns one encrypted sum. Moreover, we can see that receiving a request from Alice requires Bob to spend more CPU time in Protocol 2 then in Protocol 1. This is due to Bob needing to evaluate the multiple polynomials for each item for which he has matching a

[2]http://code.google.com/p/gmpy/
[3]The GNU Multiple Precision Arithmetic Library http://gmplib.org/

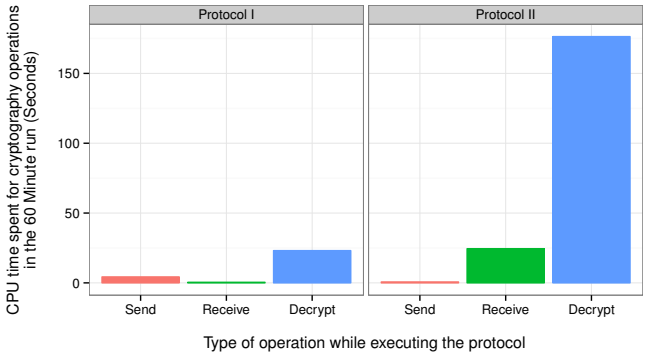

Fig. 2. CPU time spent at the encryption/decryption phases of each protocol.

Fig. 3. Bandwidth usage of both protocols.

partition. In Protocol 1, Bob only has to perform computations whenever $\gamma B, i = 1$ as explained in Section II-A.

In both protocols, it is possible to reuse a similarity request for multiple peers, since the list of preferences of a peer usually does not change frequently (in our emulations, not at all). However, it has to be noted that in a situation, in which the preferences of a peer may vary more frequently, creating a request using Protocol 1 is more expensive.

3) Bandwidth usage: With respect to the bandwidth usage of both protocols, Protocol 2 requires less bandwidth. The most important factor influencing the bandwidth costs is the dynamic size of the messages in Protocol 2. Protocol 1 always sends a preference vector of a fixed length; in contrast Protocol 2 sends as many encrypted coefficients as Alice has. As in our subset, peers have 18 items on average, the expected bandwidth usage of Protocol 2 is 18% of that of Protocol 1 as it sends vectors with a length of 100 items (for the sending and forwarding steps). However, the reply of Protocol 2 requires more bandwidth as this protocol does not sum all values computed at Bob, but has to return each evaluated polynomial.

IV. Conclusion

In this paper, we proposed, implemented, and evaluated two protocols that solve the problem of building a privacy preserving semantic overlay. Our solution extends existing protocols for the Private Set Intersection problem, in order to allow two peers to calculate their respective similarity. In order to show that both protocols are indeed ready for widespread deployment, we emulated our privacy-preserving semantic overlay using data from an existing semantic overlay. Moreover, we showed that neither protocol required a trusted third party or assumed the possibility of a connection between any two peers. Therefore both fit nicely within the real world limitations of a Peer-to-Peer environment.

Each of the proposed protocols introduces a trade-off between bandwidth and computational overhead. Protocol 1 requires more bandwidth, but less CPU time. Protocol 2 is the opposite, requiring less bandwidth and more CPU time.

References

[1] J. Liang, R. Kumar, and K. Ross, "Understanding kazaa," Misc, 2004,.

[2] D. C. Howe and H. Nissenbaum, "Trackmenot: Resisting surveillance in web search," *Lessons from the Identity Trail: Anonymity, Privacy, and Identity in a Networked Society*, pp. 417–436, 2009.

[3] S. Voulgaris and M. Steen, "Epidemic-style management of semantic overlays for content-based searching," in *Euro-Par 2005 Parallel Processing*. Springer Berlin Heidelberg, 2005, vol. 3648, pp. 1143–1152.

[4] N. Zeilemaker, M. Capota, A. Bakker, and J. Pouwelse, "Tribler: P2p media search and sharing," in *19th ACM international conference on Multimedia*. ACM, 2011, pp. 739–742.

[5] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology - EUROCRYPT 2004*, ser. LNCS, vol. 3027. Springer, 2004, pp. 1–19.

[6] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," ser. SIGCOMM '01. New York, NY, USA: ACM, 2001, pp. 149–160.

[7] P. Maymounkov and D. Mazires, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*. Springer Berlin Heidelberg, 2002, vol. 2429, pp. 53–65.

[8] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Middleware 2001*. Springer Berlin Heidelberg, 2001, vol. 2218, pp. 329–350.

[9] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, 2001, pp. 99–100.

[10] L. Kissner and D. Song, "Privacy-preserving set operations," in *in Advances in Cryptology - CRYPTO 2005, LNCS*. Springer, 2005, pp. 241–257.

[11] A. Kiayias and A. Mitrofanova, "Testing disjointness of private datasets," in *Financial Cryptography*, ser. LNCS, vol. 3570. Springer, 2005, pp. 109–124.

[12] C. Hazay and Y. Lindell, "Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries," in *5th conference on Theory of cryptography*, ser. TCC'08, pp. 155–175.

[13] D. Boneh and M. K. Franklin, "Identity-based encryption from the weil pairing," in *CRYPTO '01*. London, UK, UK: Springer-Verlag, 2001, pp. 213–229.

[14] E. D. Cristofaro, S. Jarecki, J. Kim, and G. Tsudik, "Privacy-preserving policy-based information transfer," in *PETS '09*, 2009, pp. 164–184.

[15] E. Cristofaro, P. Gasti, and G. Tsudik, "Fast and private computation of cardinality of set intersection and union," in *Cryptology and Network Security*, ser. LNCS. Springer, 2012, vol. 7712, pp. 218–231.

[16] E. Cristofaro, J. Kim, and G. Tsudik, "Linear-complexity private set intersection protocols secure in malicious model," in *ASIACRYPT '10*, ser. LNCS. Springer, 2010, vol. 6477, pp. 213–231.

[17] E. D. Cristofaro and G. Tsudik, "Practical private set intersection protocols with linear complexity," in *Financial Cryptography*, 2010, pp. 143–159.

[18] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology — EUROCRYPT '99*, ser. LNCS, vol. 1592. Springer, May 2-6, 1999, pp. 223–238.

[19] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *EUROCRYPT '04*, ser. LNCS, vol. 3027. Springer, 2004, pp. 1–19.

[20] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*, ser. STOC '88, 1988, pp. 103–112.

[21] G. Brassard, D. Chaum, and C. Crépeau, "Minimum disclosure proofs of knowledge," *J. Comput. Syst. Sci.*, vol. 37, no. 2, pp. 156–189, Oct. 1988.