Are You Really My Friend? Efficient and Secure Friend-matching in Mobile Social Networks

Mohammad Etemad
Crypto Group, Koç University
İstanbul, Turkey
metemad@ku.edu.tr

Filipe Beato
ESAT/COSIC - KU Leuven
Leuven, Belgium
filipe.beato@esat.kuleuven.be

Alptekin Küpçü
Crypto Group, Koç University
İstanbul, Turkey
akupcu@ku.edu.tr

Bart Preneel
ESAT/COSIC - KU Leuven
Leuven, Belgium
bart.preneel@esat.kuleuven.be

***Abstract*—Social networks provide users with solutions to manage and acquire social connections in the modern society. Since users of a social network can make a new friendship even when they are not meeting physically, there should be a mechanism enabling them to securely verify identity of each other. One such mechanism is to check if there are enough number of common friends, relying on the friendships established already. Current protocols for mobile social networks require parties to act honestly during the protocol, and are limited to the two-party settings. We propose two solutions for friend matching based on authenticated data structures and polynomial operations that preserve privacy of non-common friends and provide authenticity of the result. Both constructions are efficient and general to be employed in multi-party settings.**

1. Introduction

Social networks such as Facebook and Linkedin play an important role in current modern society. The successful growth of these networks have collected millions of users, connecting people having common interests and suggesting new potential friends based on these similarities. For example, Facebook may suggest people graduated from the same school to each other. In the digital world, users can use the existing friendship relationships as a source of trust to extend their connections. For instance, you can find your classmates through those who are already in your friend list.

In addition, there was extensive progress and popularity in smart mobile devices aligned with the growth in their resources, boosting mobile social networks (MSNs). While offering social networking services over the mobile devices, MSNs present distinct differences compared to the fixed (landline) social networking. First, mobile devices are physically carried around by their owners and create a good opportunity to provide location-based services. Second, the computational and power constraints in mobile devices limit the use of expensive cryptographic computations. This affects data input and output capabilities of these devices. Third, the MSN has a much larger number of potential users, and a diverse set of services can be offered as it does not require access to a computer [8].

In both the real and digital worlds, people can extend their connections through existing ones. For this, they should have and prove existence of enough number of common friends. Trust on the existing common friends helps them trust each other and make a new friendship.

To become friends through a social network, either two users meet physically, exchange their social network IDs, and lookup each other on the given network; or one user sends a request to another user that has many common friends (or interests) with him. The existence of sufficient common friends ensures both users that the other user is really the one who is claimed to be.

MSNs enable users search for potential friends or people with common interests on-the-fly. When a user enters a place, her mobile device performs a search and discovers those with enough common friends or interests. For example, when the user is waiting for her flight in an airport, her mobile device discovers someone with enough common friends in her vicinity [8], or someone graduated from the same school as she did. This can be generalized to multi-party cases, i.e., your mobile device surprises you notifying that five of your classmates are in the same conference you are attending. We focus only on matching common friends.

Secure common friend matching. The common friend matching problem requires the users to be registered at the same social network, through which they populate and extend their friend lists. Since the friendship relationships are established through the network, a naive solution is to ask the server compute and send the parties the set of common friends. Though this works for fixed social networks, it does not suit the MSN as it requires access to the network all the time, while the users may be in a place with no network coverage or Internet access. Another naive solution in such cases is that the users reveal their friend lists to each other. It is clear that it does not preserve users' privacy.

Matching common friends is actually a private set intersection (PSI) problem, where each user has its own private friend list. The users want to jointly find the intersection of their friend lists, while learning minimum information about the non-common friends. Ideally, they should learn nothing beyond the intersection. Existing PSI schemes mostly rely on public key based primitives that prohibit their usage in MSNs. Moreover, in the absence of a trusted party, each user can modify her private set to either learn more about the other party or give less information to the other party. Further, these scheme are mostly two-party schemes, i.e., if

more than two users involved, the heavy PSI scheme should be run among all users either sequentially or pairwise.

Our contributions These observations motivated us to proposes a *light-weight* common friend matching scheme that preserves *privacy* of friend lists and is *generalizable* to more than two users. *Authenticity* is another problem we address, i.e., each user can check authenticity of the friend list(s) given by other user(s) and the result. Our scheme does not need online access to the social network or Internet. It only assumes occasional access to the network to manage the friend lists. Our contributions are summarized as:

- We give the **first security definition** of a common friend matching scheme, and prove our construction's security formally against the definition.
- Our scheme operates only on the given friend lists. Hence, it can be performed mechanism even without requiring Internet access or the MSN.
- The user IDs are high entropy IDs chosen from a large domain, and their hashes are used to populate the lists. Hence, each user only learns the common friends (if any), helping preserve **privacy** of the users.
- Our scheme provides **authenticity** of the result by accompanying it with a cryptographic proof.
- Our solution is **generalizable** to more than two users. Therefore, instead of performing the matching sequentially or between all possible pairs of the involved users, all users send their friend lists to one user who runs the matching algorithm and distributes the result and proof to the other users for verification.

2. Background

2.1. Related Work

Proximity measure is a way for two (or more) people, who do not trust each other, to perform a joint computation [8]. The main purpose is to realize whether the involved users can trust each other through measuring how close or similar they are. Different approaches have proposed to measure proximity between users, e.g., the number of common friends, or the number of common interests and attributes. The focus of this work is only on common friends.

Private set intersection (PSI) is a cryptographic protocol helping two mutually untrusted parties, each with its own private set, to jointly calculate the intersection of their sets, while minimum amount of information is revealed during the protocol execution. In other words, at the end of protocol execution, each party has the intersection in hand, without (ideally) learning any extra information about the set of the other party. Several schemes using different primitives have been proposed for PSI [3], [4], [7], [12], [15], [16], [18].

An inherent problem with the PSI protocols in the absence of a trusted third party (TTP) is the malicious behavior of parties. A party may not provide her set entirely as input to the protocol (preventing the other party form learning the existence of some elements), or may include extra elements in her set (to learn more about the other party). The authorized PSI (APSI) protocols [3], [6] solve the second problem (not the first one) making the parties use only the input elements that are authorized by an authority.

De Cristofaro *et al.* [5] introduced the *private contact discovery* as an efficient cryptographic primitive using which two users of a social network, on input their contact lists, calculate their common contacts, and learn nothing beyond that. However, the underlying primitive used in their construction and the discovery protocol are relatively complex and does not suit the MSN capabilities.

Nagy *et al.* [21] used PSI techniques to find common friends in social networks, through combining PSI with bearer capabilities to provide authenticity and privacy, in a semi-honest setting.

Commutative encryption is informally a pair of encryption functions f and g satisfying $f(g(m)) = g(f(m))$, for a message m [1]. When used for two-party computation, each party only applies its function, and at the end, both of them get the same result. VENETA [26] used it for friend-of-friend detection.

Dot product. Similarities between the users' interests and properties are computed to measure their proximity. The interests and attributes of each user is represented as her *social coordinate*s, and dot product of two social coordinates is used to measure the proximity between them [8]. The dot product is applied in different scenarios [27], [8]. Liao *et al.* [19] introduced S-MATCH, as a privacy-preserving profile matching framework using property-preserving encryption.

Trust on existing friends. If a user A trusts another user B, she can also trust the other users confirmed or suggested by B. This helps generate a web of trust with diverse potential applications. Nagy *et al.* [21] used this relationship for finding out potential friends: Two users of a social network can trust each other if they have sufficient number of common (already trusted) friends.

Freedman and Nicolosi [11] proposed a privacy-preserving protocol for verifying social proximity, and used it in an email system for automatically whitelisting incoming emails based on the sender. Each user *attest*s a set of other users (her friends in a social network), meaning that she will accept emails coming from those users. Alice will accept emails coming from Carol (not in her own list) if there is a user Bob who is attested by Alice, and he attests Carol.

Yu *et al.* [27] used the friendship relations as human-established trust relationship to defend against *sybil attacks*, where a malicious user creates multiple fake identities for herself to be able to 'out vote' the honest users.

To sum up, the privacy-preserving PSI schemes use complex cryptographic primitives that do not suit the processing powers of mobile devices. The other approaches fail to address users' privacy. All these schemes, including the authorized PSI protocols, authenticate the friend IDs of each user one-by-one. This allows a user manipulate her list before getting involved in a protocol execution.

Table 1 presents a comparison among these works.

2.2. Preliminaries

Notation. $|X|$ shows the number of elements in a set X, and $x \leftarrow$ X denotes the fact that x is sampled uniformly

TABLE 1: Common friend matching protocols. ($\mathcal{F}_u$ is the friend list of user u. Costs of our construction are for two-player case.)

Protocol	Players	Cost of		
		Communication	Initiator (A)	Responder (B)
De Cristofaro *et al.* [6]	2	$\mathcal{O}(\lvert\mathcal{F}_A + \mathcal{F}_B\rvert)$	$\mathcal{O}(\lvert\mathcal{F}_A\rvert)$	$\mathcal{O}(\lvert\mathcal{F}_B\rvert)$
De Cristofaro *et al.* [5]	2	$\mathcal{O}(\lvert\mathcal{F}_A\rvert^2)$	$\mathcal{O}(\lvert\mathcal{F}_A\rvert)$	$\mathcal{O}(\lvert\mathcal{F}_B\rvert)$
Nagy *et al.* [21]	2	$\mathcal{O}(\lvert\mathcal{F}_A + \mathcal{F}_B\rvert)$	$\mathcal{O}(\lvert\mathcal{F}_A\rvert)$	$\mathcal{O}(\lvert\mathcal{F}_B\rvert)$
VENETA [26]	2	$\mathcal{O}(\lvert\mathcal{F}_A + \mathcal{F}_B\rvert)$	$\mathcal{O}(\lvert\mathcal{F}_A + \mathcal{F}_B\rvert)$	$\mathcal{O}(\lvert\mathcal{F}_A + \mathcal{F}_B\rvert)$
Our construction	Multiple	$\mathcal{O}(\lvert\mathcal{F}_A + Common\rvert)$	$\mathcal{O}(\lvert\mathcal{F}_A\rvert)$	$O(\lvert Common\rvert)$

from the set X. || represents concatenation, and PPT denotes probabilistic polynomial time. λ is the security parameter.

A function $\nu(\lambda) : Z^+ \rightarrow [0,1]$ is *negligible* if $\forall$ positive polynomials $p, \exists$ a constant c such that $\forall\ \lambda{>}c,\ \nu(\lambda){<}1/p(\lambda)$. *Overwhelming* probability is greater than $1-\nu(\lambda)$ for some negligible function $\nu(\lambda)$.

In addition, we consider $\texttt{Enc}_{pk}(\cdot)$ to be an additive homomorphic public key encryption, such that the followings are valid: $\texttt{Enc}_{pk}(a+b) = \texttt{Enc}_{pk}(a) \cdot \texttt{Enc}_{pk}(b)$ and $\texttt{Enc}_{pk}(a \cdot b) = \texttt{Enc}_{pk}(a)^b$ for $a, b \in R$.

Hash functions generate fixed-length strings, given arbitrary-length strings. Let $h{:}\mathcal{K}{\times}\mathcal{M}{\rightarrow}\mathcal{C}$ be a family of hash functions, and $h_K()$ with $K{\in}\mathcal{K}$ identify a member. A hash function family is *collision resistant* if $\forall$ PPT adversaries $\mathcal{A}, \exists$ a negligible function $\nu(\lambda)$ s.t. $Pr[K \leftarrow \mathcal{K}; (M, M') {\leftarrow}\mathcal{A}(h, K) : (M' \neq M){\wedge}(h_K(M) = h_K(M'))] \leq \nu(\lambda)$. A hash function family is *preimage resistant* if $\forall$ PPT adversaries $\mathcal{A}, \exists$ a negligible function $\nu(k)$ s.t. $Pr[K{\leftarrow}\mathcal{K}; M{\leftarrow}\mathcal{M}; C{\leftarrow}h_K(M); M'{\leftarrow}\mathcal{A}(h, C) : C{=}h_K(M')]{\leq}\nu(\lambda)$. Collision resistance implies preimage resistance [17].

Authenticated Data Structure (ADS) is a scheme for data authentication used by an untrusted server to answer the client queries and provide cryptographic proofs showing authenticity of his answers [25], [23]. The client can verify the proof against some local metadata. There are different types of ADSs: accumulators [2], authenticated skip lists [14], authenticated hash tables [24], Merkle hash trees [20], 2-3 trees [22], and hierarchical ADSs (HADS) [9].

Definition 2.1. An **ADS scheme** consists of the following polynomial-time algorithms [23], [9]:

- $k \leftarrow \texttt{Gen}(1^\lambda)$: Run by users to generate a key k given the security parameter λ. They share k with the network.
- $(ans, \pi) \leftarrow \texttt{Certify}(k, cmd)$: Run by the network to perform a user command cmd. The proof π enables the user verify authenticity of the answer ans.
- $\{\textbf{accept}, \textbf{reject}\} \leftarrow \texttt{Verify}(k, ans, \pi)$: A client runs this algorithm to verify authenticity of the answer. It takes as input the key k, the answer ans, and the proof π. It finally outputs an `accept` or a `reject` signal.

Definition 2.2. A **signature scheme**, used for message integrity, includes the following PPT algorithms [13]:

- $(sk, vk) \leftarrow \textbf{Gen}(1^\lambda)$: Run by the network to generates a pair of signing and verification keys (sk, vk) using the security parameter λ. He then shares vk with all users.
- $\sigma \leftarrow \textbf{Sign}(sk, m)$: Run by the network to compute a signature σ on a value m using the signing key sk.
- $\{\textbf{accept}, \textbf{reject}\} \leftarrow \textbf{Verify}(vk, m, \sigma)$: Users use this algorithm to check whether σ is a valid signature on a value m, using the verification key vk. It outputs an acceptance or a rejection signal, accordingly.

3. Formal Definitions

Model. We consider *users* to be members of the same *social network*, and connected by a friendship relationship in order to access and exchange information. For simplicity, we assume relationships among users to be symmetric. The network allows users to get registered, search and add friends, and perform social activities. The network stores all information about users and their activities, and presents them upon request. Most user operations are done through the network, but they can also work jointly to perform a task, i.e., compute list of common friends. We use the terms *user* and *client* interchangeably. Also, *network* refers to the MSN. Figure 1 shows our common friend matching model.

Adversarial model. We consider the network to be trusted, providing users with the correct friend lists and updates when two users become friends. However, users do *not* trust each other. Hence, before becoming friends, they should know each other either directly or through the common friends, which requires be checked and proven. This work targets the second case: helping two or more users to see if they have enough common friends and can be friends. Once they became friends, they trust each other.

3.1. Overview of Our Scheme

Showing enough number of common friends ensures parties that they are connecting to the right user, when they do not physically meet. This should be done in a privacy-preserving manner, i.e., each user learns only the common friends. Further, each user wants to make sure the other user is participating with the correct set of her friend IDs.

In our scheme, the MSN assigns a high-entropy random ID to each user upon registration. Besides, to ensure the friend lists are not manipulated, we store hash of friend IDs of each user in a tree-based ADS, and ask the MSN to certify the digest of the ADS. Storing hash of IDs prevents revealing the real friend IDs, and helps preserve privacy. The tree ties all friend IDs together, and the MSN's certification

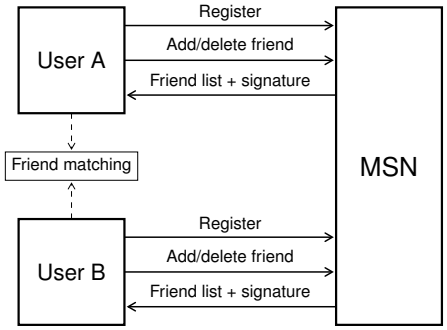

Figure 1: Our common friend matching model.

ensures that the tree is not tampered with. As a result, each user can rest assured that the other user provides a correct friend list. Note that certifying the IDs in a friend list one-by-one, as in [3], [4], does not provide authenticity of a friend list since one may hide some of her friends.

3.2. The Common Friend Matching Scheme

A common friend matching scheme allows users of a social network to efficiently match their common friends while ensuring authenticity of the result and preserving privacy of the users. Let $h(.)$ be a collision resistant hash function.

Definition 3.1. A **common friend matching scheme** includes the following PPT algorithms between stateful users and a stateful social network:

$(sk_p, vk_p) \leftarrow$ **Setup**(1^λ) : Run by the network to generate the signing and verification keys (sk_p, vk_p), given as input the security parameter λ. He shares the verification key vk_p with the users.

$(id_u, L_u, \sigma_u) \leftarrow$ **Register**(u) : The network selects a random ID id_u for the user u, creates an empty friend list L_u, signs it as σ_u, and sends both to the user.

$(L'_{u_1}, \sigma'_{u_1}, L'_{u_2}, \sigma'_{u_2}) \leftarrow$ **Add**$(sk_p, vk_p, id_{u_1}, id_{u_2}, L_{u_1}, L_{u_2})$: The network runs this algorithm on receipt of a friendship request from a user u_1 and confirmation of a user u_2. It adds id_{u_1} into L_{u_2} (friend list of id_{u_2}) to compute L'_{u_2}. Similarly, it updates L_{u_1} (id_{u_1}'s friend list) by inserting id_{u_2}. He then signs the friend lists as σ'_{u_1} and σ'_{u_2}, and sends each to the respective user.

$(L'_{u_1}, \sigma'_{u_1}, L'_{u_2}, \sigma'_{u_2}) \leftarrow$ **Break**$(sk_p, vk_p, id_{u_1}, id_{u_2}, L_{u_1}, L_{u_2})$: The network runs this algorithm on receipt of a breaking request from a user u_1. It removes id_{u_1} from L_{u_2}, and id_{u_2} from L_{u_1}. He then signs the resulting L'_{u_1} and L'_{u_2} as σ'_{u_1} and σ'_{u_2}, respectively, and sends each back to the corresponding user.

$(\pi, L_{cmn}) \leftarrow$ **Match**$(vk_p, L_{u_1}, \sigma_{u_1}, L_{u_2}, \sigma_{u_2}, \ldots)$: Run by a user to compute the set of common friends L_{cmn} and a proof π. It takes as input the network's verification key and the friend lists and certifications of all users whose common friends are being calculated.

(**accept**/**reject**) $\leftarrow$ **Verify**(vk_p, L_{cmn}, π) : Each user runs this algorithm to verify the proof π and the set of common friends L_{cmn}. It returns `accept` if the proof is verified, and `reject` otherwise.

When the user asks the network to make a friendship with another user, it waits for conformation of the other user to complete the task. Breaking friendship is done upon request. The network is not involved in a `Match` protocol that is run among the involved users. The friend lists are *encoded*.

Authenticity. If the user running the matching algorithm deviates from honest behavior by giving wrong answers and proofs[1], the other users should detect it with overwhelming probability, and vice versa. The authenticity game $\texttt{AuthGame}_{\mathcal{A}}(\lambda)$ between the adversary $\mathcal{A}$ who acts as the malicious user running the matching algorithm and a challenger $\mathcal{C}$ who plays the role of the network and other involved honest user(s), is defined as:

- **Initialization**. The challenger starts the `Setup` algorithm to initialize the environment and generate the pair of signing and verification keys (sk, vk). He stores the keys, and shares the verification key with $\mathcal{A}$.
- **Friendship**. $\mathcal{A}$ first asks $\mathcal{C}$ to register him. $\mathcal{C}$ runs the `Register` algorithm and sends $\mathcal{A}$ the result. $\mathcal{A}$ specifies a user id_u and asks the challenger to run either the `Add` or the `Break` algorithm. The challenger executes the requested algorithm and sends him the results. $\mathcal{A}$ also asks for the friend list of a user $id_{u'}$ from the challenger, executes the `Match` algorithm, and returns back the proof. $\mathcal{C}$ verifies the proof and informs $\mathcal{A}$ about the result. This is repeated polynomially-many times. $\mathcal{C}$ keeps a local copy that is transparent to $\mathcal{A}$.
- **Challenge**. $\mathcal{A}$ asks $\mathcal{C}$ the friend list of a user id_u and returns a list L_{cmn} and a proof π. $\mathcal{C}$ outputs `accept` if the proof is verified, and `reject` otherwise.

$\mathcal{A}$ wins the game if the challenger accepts, while the obtained common friend list differs from the real one. The output of game is defined to be 1 in this case.

Definition 3.2 (Authenticity). Our common friend matching scheme is **authentic** if no PPT adversary $\mathcal{A}$ can win the above game with probability better than negligible in the security parameter, i.e., $Pr[\texttt{AuthGame}_{\mathcal{A}}(\lambda) = 1] \leq \nu(\lambda)$, $\forall$ PPT $\mathcal{A}$, and some negligible function $\nu(\lambda)$.

Privacy. By privacy, we mean that our scheme reveals no information about non-common friend IDs to the users running the matching algorithm. The adversary is trying to discover the identity of some non-common users from the received encoded friend lists. Our scheme is private if no adversary can win the following experiment with a significant probability. The privacy game $\texttt{PrivGame}_{\mathcal{A}}(\lambda)$ between the adversary $\mathcal{A}$ who acts as the malicious user and a challenger $\mathcal{C}$ who plays the role of the network and other (honest) users, is defined as ($\mathcal{A}$ has oracle access to the ID encoding algorithm, $\mathcal{E}(.)$):

- **Initialization**. The challenger starts the `Setup` protocol to initialize the environment and generate the key pair (sk_p, vk_p). He stores the keys, and shares only the verification key vk_p with $\mathcal{A}$.
- **Query**. $\mathcal{A}$ asks the challenger the friend list of a user id_{u_i}. On receipt the answer from $\mathcal{C}$, $\mathcal{A}$ runs the `Match` algorithm and sends back the result. This process can be repeated polynomially-many times.
- **Challenge**. $\mathcal{A}$ outputs a user IDs id_u and a value h_u.

$\mathcal{A}$ wins the game if h_u matches the encoded value of id_u and he has not already queried $\mathcal{E}(.)$ for id_u. The output of game is defined to be 1 in this case.

Definition 3.3 (Privacy). Our common friend matching scheme is **private** if no PPT adversary $\mathcal{A}$ can win the above privacy game with probability better than negligible in the security parameter, i.e., $\forall$ PPT $adversaries$ $\mathcal{A}, \exists$ a $negligible$ $function$ $\nu(\lambda)$ $s.t.: Pr[\texttt{PrivGame}_{\mathcal{A}}^{\mathcal{E}(.)}(\lambda)\texttt{=1}] \leq \nu(\lambda)$.

1. The case a malicious user provides a manipulated list is similar and simpler: $\mathcal{A}$ only provides the friend list, and $\mathcal{C}$ runs the matching protocol.

4. ADS-Based Construction

We now give a lightweight construction suitable for mobile devices. The polynomial-based construction is given in the appendix. An important assumption is that the network does not need to be online for this operation to be performed. It suffices to have the MSN certification on the users' friend lists (discussed more in Section 4.3). Moreover, we assume the users do not trust each other, while the MSN is trusted, which resembles the real usages.

Once Alice becomes a user of an MSN, the network assigns her a high-entropy random ID id_{Alice} chosen uniformly from a large domain (e.g., $\{0,1\}^{128}$), generates an empty Merkle hash tree resulting in a root R_{Alice}, signs the root of tree concatenated with the user ID as σ_{Alice}= $\texttt{Sign}_{sk_p}(id_{Alice}||R_{Alice})$ to bind the tree to her ID, and sends the ID, the tree, and the signature σ_{Alice} to Alice.

Later, upon forming the friendship connection between Alice and Carol, the MSN adds (hash of) Carol's ID into ADS_{Alice}, and (hash of) Alice's ID into ADS_{Carol}, signs the new roots in the same way, and sends the new ADSs and signatures to the respective users. Note that a user's tree is built using hashes of her friends' IDs. When a user gives her friend list to another user, the receiving user cannot uncover IDs of non-common friends supposed that the hash function is secure. Our construction is given in Figure 2.

First, we discuss the two-party case. Whenever, Alice and Bob come to a close distance, e.g., both are waiting for a flight at the airport or traveling in the same train, their mobile devices check to see if they have any friend in common. One of the devices, say Alice's device (it does not matter which one), starts a common friend matching protocol and sends her ADS and the respective signature to the other device. Bob's device first verifies that the tree and the signature are both genuine and belong to Alice, and then runs the matching protocol.

We modify the ADS matching algorithm from [10], as presented in Algorithm 4.1. Each ADS is traversed only once, and the algorithm specifies the matching IDs and generates the proof. Each comparison includes some values from each ADS, located in the proof in a given order. If it is started by Alice's ADS, the values selected from Alice's ADS always come before those of Bob's ADS, separated by ':'. The consecutive values belonging to the same ADS are separated by ',', and ';' separates different comparisons.

Assume the encoded friend list of Alice is $\mathcal{F}_A = \{a_1, a_2, ..., a_n\}$, and that of Bob is $\mathcal{F}_B = \{b_1, b_2, ..., b_m\}$. The algorithm starts by the leftmost leaf nodes in both ADSs, $Node_A$ and $Node_B$, and goes on until the end of one of the ADSs is met (lines 1 and 2 of the Algorithm 4.1). It first checks if they store the same value (line 3). If this is the case, it reflects this into the proof (line 4), finds the next nodes on both ADSs (lines 5 and 16) and goes to the next round (line 18). Otherwise, it selects the max value between them (line 7), say b_1 on ADS_{Bob}, (since there will be no matching for values less than this value) and searches for b_1 on ADS_{Alice} (line 8). If it reaches end of ADS_{Alice} (line 9), puts the intermediate information of ADS_{Bob} up to the end

Algorithm 4.1: `Match`: common friend matching alg.

Input: Current nodes of ADS_{Alice} and ADS_{Bob}: $Node_A$ and $Node_B$, initialized by the leftmost nodes of the corresponding ADSs at the beginning.
Output: The proof: π

1 **if** $Node_A$ *is null OR* $Node_B$ *is null* **then**
  `// One ADS reached end.`
2 **return** proof of the remaining part of the other ADS;
3 **if** $Node_A.val = Node_B.val$ **then**
  `// Reflect the matching into the proof.`
4 $\pi = Node_A.val$ + ':' + $Node_B.val$;
5 $Next_A = ADS_{Alice}$.`Next()`; `// Go ahead.`
6 **else**
7 Find the node with the bigger value, say $Node_B$;
  `//Find matching on` ADS_{Alice}.
8 $(Next1_A, Next2_A) = ADS_{Alice}$.`Next`$(Node_B.val)$;
9 **if** $Next1_A$ *is null* **then**
    `//No matching found, end of` $\texttt{ADS}_{\texttt{Alice}}$.
10 $\pi = \pi$ + ':' + $Node_B.val$, *proof of the remaining part of* ADS_{Bob};
11 **else**
12 **if** $Next2_A$ *is null* **then**
    `// Put matching into proof.`
13 $\pi = \pi + Next1_A.val$ + ':' + $Node_B.val$;
    $Next_A = ADS_{Alice}$.`Next()`; `// Go ahead.`
14 **else**
15 $\pi = Next1_A.val$ + ',' + $Next2_A.val$ + ':' + $Node_B.val$; `// No matching.`
16 $Next_B = ADS_{Bob}$.`Next()`; `// Go ahead.`
17 $\pi = \pi$ + ';' + `Match`$(Next_A, Next_B)$; `// Go next round.`
18 **return** π;

into the proof (line 10). If a matching is found on ADS_{Alice} (line 12), i.e., $\exists\ a_i \in ADS_{Alice}\ s.t.\ b_1 = a_i$, both values are inserted into the proof showing a matching: $\pi = a_i : b_1$ (line 13) and it goes to the next round (line 18) after moving to the next nodes on the ADSs (lines 14 and 17). Otherwise, the boundary records (the two *consecutive* values on ADS_{Alice} that b_1 would have been located between them, i.e., $\exists\ a_i, a_{i+1} \in ADS_{Alice}\ s.t.\ a_i < b_1 < a_{i+1}$), together with b_1, are inserted into the proof: $\pi = a_i, a_{i+1} : b_1$ (line 16). This shows b_1 has no matching on ADS_{Alice}. This process goes on until the end of one ADS is reached, in which case, all required information up to the end of the other ADS are also inserted into the proof.

During this process, the algorithm jumps to the next *processing* node in many situations. It is a node whose value is either immediately after a_i and b_j (on their respective ADSs) in case of previous matching (i.e., $a_i = b_j$), or the closest value to one of them on the other ADS, otherwise. If the current and processing nodes are not successive, the intermediate information required for verifying the ADS by the other user, will be added to the proof. The algorithm `Next`, used to find the processing nodes inside the Algorithm 4.1, serves this goal.

For example, we execute our matching algorithm on ADS_{Alice} and ADS_{Bob} given in Figures 3 and 4, respectively. For a simple presentation, we use the values instead their hashes. The algorithm goes to the leftmost nodes of

Let A=(Gen,Certify,Verify) be a secure ADS scheme and S=(Gen,Sign,Verify) be a secure signature scheme. Build a common friend matching scheme CFM=(Setup,Register,Add,Break,Match,Verify) as:

- $\texttt{Setup}(1^\lambda)$:
  - The MSN runs $k \leftarrow \texttt{A.Gen}(1^\lambda)$ and $(sk_s, vk_s) \leftarrow \texttt{S.Gen}(1^\lambda)$,
  - sets $sk_p = sk_s$ and $vk_p = \{vk_s, k\}$, and shares vk_p with all his users.
- $\texttt{Register}(u)$:
  - The MSN selects a random ID id_u, creates an empty ADS as $L_u = \texttt{A.Certify}(vk_p.k, \text{`}\texttt{Create}\text{'})$,
  - concatenate the root of tree R_u and id_u, and signs the results as $\sigma_u = \texttt{S.Sign}_{sk_p}(id_u||R_u)$,
  - and sends the user ID id_u, the tree, and the signature σ_u to the user.
- $\textbf{Add}(sk_p, vk_p, id_{u_1}, id_{u_2}, L_{u_1}, L_{u_2})$:
  - The user id_{u_1} sends a friendship request for id_{u_2} to the MSN.
  - The MSN relays the request to id_{u_2} and waits for her confirmation.
  - Upon receipt, the MSN runs $L_{u_2}.\texttt{Certify}(vk_p.k, \text{`}\texttt{Add}||id_{u_1}\text{'})$ and $L_{u_1}.\texttt{Certify}(vk_p.k, \text{`}\texttt{Add}||id_{u_2}\text{'})$,
  - signs their updated roots as $\sigma'_{u_1} = \texttt{S.Sign}_{sk_p}(id_{u_1}||R'_{u_1})$ and $\sigma'_{u_2} = \texttt{S.Sign}_{sk_p}(id_{u_2}||R'_{u_2})$, and
  - sends $(\sigma'_{u_1}, R'_{u_1})$ and $(\sigma'_{u_2}, R'_{u_2})$, to u_1 and u_2, respectively.
- $\textbf{Break}(sk_p, vk_p, id_{u_1}, id_{u_2}, L_{u_1}, L_{u_2})$:
  - The user id_{u_1} sends a break request for id_{u_2} to the MSN.
  - The MSN runs $L_{u_2}.\texttt{Certify}(vk_p.k, \text{`}\texttt{Break}||id_{u_1}\text{'})$ and $L_{u_1}.\texttt{Certify}(vk_p.k, \text{`}\texttt{Break}||id_{u_2}\text{'})$,
  - signs their updated roots as $\sigma'_{u_1} = \texttt{S.Sign}_{sk_p}(id_{u_1}||R'_{u_1})$ and $\sigma'_{u_2} = \texttt{S.Sign}_{sk_p}(id_{u_2}||R'_{u_2})$, and
  - sends $(\sigma'_{u_1}, R'_{u_1})$ and $(\sigma'_{u_2}, R'_{u_2})$, to u_1 and u_2, respectively.
- $\textbf{Match}(vk_p, L_{u_1}, \sigma_{u_1}, L_{u_2}, \sigma_{u_2}, \ldots)$:
  - All involved users send their friend lists L_{u_i} and certifications σ_{u_i} to a user u_j.
  - u_j first runs $\texttt{A.Verify}(vk_p.k, L_{u_i}, null)$ and $\texttt{S.Verify}(vk_p.vk_s, id_{u_i}||R_{u_i}, \sigma_{u_i})$, then
  - runs the Algorithm 4.1 to generate the list of common friends L_{cmn} and the proof π, and
  - sends them to all other involved users.
- $\textbf{Verify}(vk_p, L_{cmn}, \pi)$:
  - Each user verifies the list of common friend, as described later in this section, and
  - runs $\texttt{A.Verify}(vk_p.k, L_{u_i}, null)$ and $\texttt{S.Verify}(vk_p.vk_s, id_{u_i}||R_{u_i}, \sigma_{u_i})$ to verify other users' lists.
  - She outputs accept if both verifications were successful, and reject otherwise.

Figure 2: Construction of our common friend matching scheme.

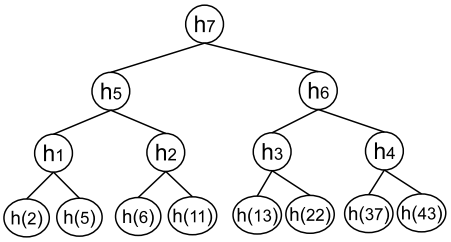

Figure 3: ADS_{Alice} storing hash of IDs of her friends.

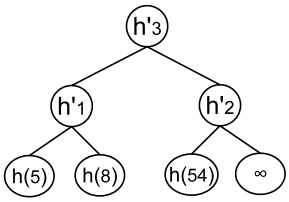

Figure 4: ADS_{Bob} storing hash of IDs of his friends.

both ADSs, and observes that $2 < 5$. Hence, it looks for and finds another node storing 5 in ADS_{Alice}. After this step, the proof looks like $\pi = 2, 5 : 5$. Since a matching is found, it runs ADS_{Alice}.Next(5) and ADS_{Bob}.Next(5) to jump to the following nodes on their respective ADSs. Now, the current values are 6 and 8, respectively. Since $6 < 8$, the algorithm runs ADS_{Alice}.Next(8) that returns 6 and 11 meaning that 8 does not have a matching on ADS_{Alice}. This fact should be reflected into the proof: $\pi = 2, 5 : 5\ ;\ 6, 11 : 8$. Now it runs ADS_{Bob}.Next(8) to go to the following node on ADS_{Bob}, the one who stores 54. Since $11 < 54$, and 54 is the last value on ADS_{Bob}, the algorithm adds proof of the remaining parts of ADS_{Alice} and ends, outputting the proof $\pi = 2, 5 : 5\ ;\ 6, 11 : 8\ ;\ h_3, 37, 43 : 54, \infty$.

As a very small example, it shows the algorithm skips over the unnecessary nodes and results in an efficient proof.

Since, Bob is running this protocol, he observes that there is only one matching: $h(5)$. He sends the proof to Alice who verifies the proof and finds out the matching value(s) as well. The verification process is illustrated graphically in Figure 5, which shows how Alice reconstructs the required parts of the ADSs. Then, she checks whether the root of her own (reconstructed) ADS matches the locally stored value, and the root of the other ADS is verified against the received signature. If both checks passed, she accepts the result, and rejects otherwise. During this process she realizes that there is one common value: $h(5)$.

4.1. Generalization of the Algorithm

Existing PSI algorithms require each party to extract the result (i.e., the common set values) directly that limits their usage for multi-party settings. If there are *n* mobile users around, and all want to find their common friends, they need $O(n)$ (sequential) rounds of executing the existing PSI algorithms to find the common friends among all of them. Our algorithm, on the hand, is easily generalizable for multi-party case. All parties send their ADSs and signatures to one party who runs the algorithm on all the ADSs and

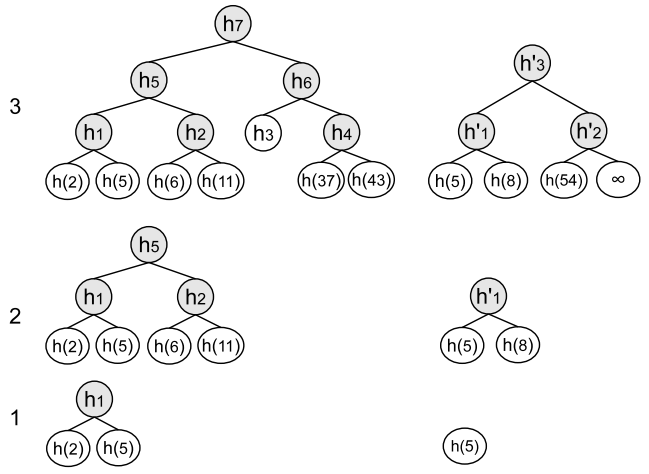

Figure 5: Verification. The colored nodes are computed.

distributes the proof to other parties. On receipt, they all perform verification and extract the common values, if there are some and the verification was successful.

The matching algorithm is similar to the two-party case, but only compares more than two values to either see that all values are equal or find the maximum value among them. It starts with the leftmost leaf nodes in all ADSs. If they all store the same value, puts all in the proof. Otherwise selects the maximum value among them, say b_1 on ADS_{Bob}, (since there is no matching for values less than this value) and searches for b_1 on all other ADSs. If a matching is found, all values are inserted into the proof showing the matching. Otherwise, the boundary records of all other ADSs, together with b_1, are inserted into the proof. This shows b_1 has no matching on other ADSs. Then, it jumps to the next *processing* node on all ADSs and repeats the process.

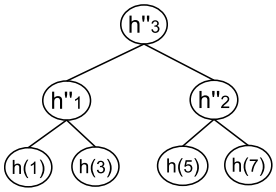

Figure 6: Carol's ADS.

With the ADS_{Carol} in Figure 6, our matching algorithm goes to the nodes storing 2, 5, and 1 in ADS_{Alice}, ADS_{Bob}, and ADS_{Carol}, respectively. Since 5 is the maximum, it calls ADS_{Alice}.Next(5) and ADS_{Carol}.Next(5) and generates the proof as $\pi = 2, 5 : 5 : h''_1, 5$. Then, it calls ADS_{Alice}.Next(), ADS_{Bob}.Next(), and ADS_{Carol}.Next(), and observes that 7 is the last value in ADS_{Carol} and 8 is the maximum value. It reflects these in the proof and exits: $\pi = 2, 5 : 5 : h''_1, 5 \ ; \ h_2, h_6 : 8, h'_2 : 7$. Then, the party running the algorithm distributes the proof to the other parties for verification. The proof shows there is only one common value $h(5)$ among these three ADSs.

4.2. Security Proof

Theorem 4.1. Our common friend matching scheme is authentic according to Definition 3.2 if the underlying Merkle hash tree and signature scheme are secure.

Proof 4.1. We reduce security of our scheme to that of the underlying building blocks. If a PPT adversary $\mathcal{A}$ wins security game of our scheme with non-negligible probability, we use it to construct another PPT algorithm $\mathcal{B}$ who breaks security of either the Merkle hash tree or the signature scheme, with non-negligible probability. $\mathcal{B}$ acts as the adversary in the security games with the Merkle hash tree challenger $\mathcal{C}_M$ and the signature scheme challenger $\mathcal{C}_S$. In parallel, $\mathcal{B}$ plays the role of challenger in our game with $\mathcal{A}$. Since our matching process is asymmetric, i.e., one user runs the matching algorithm and sends the result to the other user(s) for verification, we consider two scenarios.

First scenario: the user who runs the matching algorithm is malicious, i.e., she may provide a fake friend list or perform the matching incorrectly.

Setup. $\mathcal{B}$ receives a key k from $\mathcal{C}_M$ and a verification key pk from $\mathcal{C}_S$, and shares them with $\mathcal{A}$. $\mathcal{A}$ sends him a request to execute an algorithm:

- For a `Register` algorithm, the challenger assigns him a random ID $id_{\mathcal{A}}$, asks $\mathcal{C}_M$ build an empty Merkle hash tree, asks $\mathcal{C}_S$ to sign root of the tree, and sends the ID, the tree, and the signature to the adversary.
- For an `Add` (`Break`) algorithm, he gives a user ID id_u and asks $\mathcal{B}$ to add (remove) it to (from) his friend list. $\mathcal{B}$ asks $\mathcal{C}_M$ add (remove) id_u into (from) the friend list of the adversary and $id_{\mathcal{A}}$ into (from) the friend list of id_u, asks $\mathcal{C}_S$ to sign roots of the respective modified Merkle hash trees, and sends the signature together with the updated ADS to the adversary. $\mathcal{B}$ also keeps local copies of all ADSs that is transparent to $\mathcal{A}$.
- For a `Match` algorithm, he requests the friend list of a user id_u, computes the list of common friends, and returns back the result accompanied with the proof. $\mathcal{B}$ verifies the result and notifies $\mathcal{A}$ about the result. These two steps can be done polynomially-many times.

Challenge. The adversary specifies a user id_u, asks $\mathcal{B}$ the friend list of id_u, computes the set of common friends between himself and id_u, and gives the result and the proof to $\mathcal{B}$ who runs the `Verify` algorithm. If $\mathcal{B}$ accepts $\mathcal{A}$'s answer with probability p (which means that the root of $\text{ADS}_{\mathcal{A}}$ matches the given signature), while the obtained set of common friends disagrees with the local knowledge, either of the following cases happens:

- There are two different ADSs with the same root value. (This includes the case where the matching algorithm is done wrong.) $\mathcal{B}$ can use these two ADSs to break security of the Merkle hash tree with probability p.
- The root of $\mathcal{A}$'s ADS reconstructed during verification does not match the local knowledge, but the signature is accepted. This means that $\mathcal{A}$ has forged a signature on a modified (root) value. $\mathcal{B}$ can use this fact to break security of the signature scheme with probability p.

Since both Merkle hash tree and signature scheme are secure by assumption, p must be negligible, which means that $\mathcal{A}$ has negligible chance to pass $\mathcal{B}$'s verification and win the game. Hence, our scheme is secure in this regard if the underlying building blocks are secure.

Second scenario: the user who sends her friend list is malicious, i.e., she may provide a fake friend list. This

is very similar to the first scenario except that the adversary only populates his list (with the help of the challenger) and asks the challenger to run the matching algorithm. Now, the challenger verifies the input given by the adversary before running the matching protocol. As above, there are two possible cases for the adversary's misbehavior, each enabling the challenger to break security of either of the underlying building blocks. However, since they are both secure, the adversary has negligible chance of breaking them. As a result, he cannot break security of our scheme with probability better than negligible in the security parameter, confirming again our scheme is authentic.

Theorem 4.2. Our common friend matching scheme is private according to Definition 3.3 supposed that the underlying hash function is preimage resistant.

Proof 4.2. We reduce privacy of our scheme to the security of the underlying hash function. (The adversary has access to the hash function used in the Merkle hash tree.) If a PPT adversary $\mathcal{A}$ wins privacy game of our scheme with non-negligible probability, we use it to construct another PPT algorithm $\mathcal{B}$ who breaks preimage resistance of the hash function, with non-negligible probability. $\mathcal{B}$ acts as the challenger in the privacy game with $\mathcal{A}$, and simultaneously, plays the role of server in the security games with the hash function challenger $\mathcal{C}_H$.

Setup. $\mathcal{C}_H$ gives description of a hash function $h(.)$ to $\mathcal{B}$ who shares it with $\mathcal{A}$.

Query. $\mathcal{A}$ requests the friend list of a user id_{u_i}. $\mathcal{B}$ asks $\mathcal{C}_H$ a list (of a given size) of hash values, builds a Merkle hash tree over these values, and sends it to $\mathcal{A}$. $\mathcal{A}$ gives back a list of common friends and the proof. This step can be done polynomially-many times.

Challenge. $\mathcal{A}$ outputs a user ID id_u not in his own friend list, and a hash value h_u given already by $\mathcal{C}_H$. $\mathcal{B}$ relays them to $\mathcal{C}_H$. $\mathcal{A}$ wins if $h_u = h(id_u)$.

If $\mathcal{A}$ wins with probability p, $\mathcal{B}$ also wins with the same probability. Since $h(.)$ is preimage resistant by assumption, p should be negligible, i.e., $\mathcal{A}$ has negligible chance of finding such a user ID. Hence, our scheme is private if the underlying hash function is preimage resistant.

4.3. Discussion

We discuss some issues about our scheme in real usage.

- A malicious user can regularly (legitimately, by asking the server) add new friends into her friend list and try to learn more about the other users. This is a general problem in the existing protocols and schemes. However, the MSN can easily detect and prevent such activities. Indeed, the MSN can put a reasonable upper bound on the number of friend addition/removal activities per day, or even on the maximum number of friends a user can acquire through the network.
- Our scheme relies on the MSN certificates to perform offline matchings. This makes simple *replay attacks* possible. Again, this is a general problem and is not special to our scheme. For example, Nagy *et al.* [21] also refreshes users' capabilities periodically. A malicious user can participate with an outdated certified friend list in a matching algorithm run. This can be prevented using timestamps. The MSN adds a timestamp to the signatures showing the time the list is signed. This approach has a *validity window* for the signatures, during which a malicious user can still perform replay attacks. However, combined with the previous observation and the fact that the user checks for a lower bound on the number of common friends to accept a friendship request, this attack will not have significant implications.
- A number of *colluding* malicious users can learn more about the friend list of a user u_i. Whenever a malicious user receives the (encoded) friend list of u_i for matching, she can share it with other users to extract more friend IDs. This is also a general problem in most of the existing schemes to make comparison against different users possible. However, we do not consider this as a serious attack as the other users would have been given the list upon request. The colluding users can uncover at most the union of their common friends with u_i. The important fact is that they cannot go beyond that even though the are colluding. (This is the result of privacy-preservingness of our scheme.)

5. Performance Analysis

We implemented a prototype of our scheme and evaluated its performance on a Samsung Galaxy S1 mobile phone having a Cortex-A8 chip and 512 MB RAM, running Android 2.1. We employ SHA1 hash function for generating 160-bit random IDs for users. Then, a Merkle hash tree is built over the set of friend IDs assigned to each user. Finally, the root value of each tree is signed using an RSA scheme with 2048 modulus and 160-bit exponent. (These are the operations that supposed to be done by the network.)

For each experiment, we generate two Merkle hash trees representing two users, apply our common friend matching algorithm on them, and measure the times. We perform all experiments on only one device, and exclude the communication time. The results are given in Table 2. The values are averages of 10 runs.

We repeat our experiments with two users with different number of total and common friends. The user IDs are generated randomly (outputs of the hash function), but we select some of them as common and put them on both lists.

This table shows that it takes up to 1.1 s for friend matching and a similar time for verification. Therefore, two devices can run our protocol and find their common friends in at most 2.2 s, in the above settings. Moreover, it shows that the number of common friends has more effect on the matching and verification times than the total number of friends (in our settings). Note that we have done our experiments on a regular device. Thus, we expect much better performance on recent mobile phone devices.

For the case with input size 100 and intersection size 10, the scheme of Nagy *et al.* [21] takes $\approx 652\ ms$ while

TABLE 2: Friend matching and verification times. ($\mathcal{F}_u$ represents the list of friends of a user u.)

$\lvert\mathcal{F}_A\rvert$	$\lvert\mathcal{F}_B\rvert$	$\lvert Common\rvert$	Time (ms)	
			Matching	Verification
100	200	10	227	182
		20	431	353
		50	662	607
100	500	10	269	226
		20	493	428
		50	747	719
200	1000	10	305	259
		20	483	411
		50	794	742
		100	1143	1056

ours takes 227 ms, showing $\approx$ 3X better performance. The same happens in similar scenarios, e.g., for input sizes 200.

Acknowledgments We acknowledge the support of TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project number 114E487, European Union COST Actions IC1306 and IC1206, and Royal Society Newton Advanced Fellowship NA140464.

References

[1] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *ACM SIGMOD international conference on Management of data*, pages 86–97. ACM, 2003.

[2] J. Benaloh and M. De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *EUROCRYPT'93*, pages 274–285. Springer, 1994.

[3] J. Camenisch and G. M. Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security*, pages 108–127. Springer, 2009.

[4] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *Advances in Cryptology-ASIACRYPT 2010*, pages 213–231. Springer, 2010.

[5] E. De Cristofaro, M. Manulis, and B. Poettering. Private discovery of common social contacts. In *Applied Cryptography and Network Security*, pages 147–165. Springer, 2011.

[6] E. De Cristofaro and G. Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security*, pages 143–159. Springer, 2010.

[7] C. Dong, L. Chen, J. Camenisch, and G. Russello. Fair private set intersection with a semi-trusted arbiter. In *Data and Applications Security and Privacy XXVII*, pages 128–144. Springer, 2013.

[8] W. Dong, V. Dave, L. Qiu, and Y. Zhang. Secure friend discovery in mobile social networks. In *INFOCOM, 2011 Proceedings IEEE*, pages 1647–1655. IEEE, 2011.

[9] M. Etemad and A. Küpçü. Database outsourcing with hierarchical authenticated data structures. In *Information Security and Cryptology–ICISC 2013*, pages 381–399. Springer, 2014.

[10] M. Etemad and A. Küpçü. Database outsourcing with hierarchical authenticated data structures. Cryptology ePrint Archive, Report 2015/351, 2015. `http://eprint.iacr.org/2015/351.pdf`.

[11] M. J. Freedman and A. Nicolosi. Efficient private techniques for verifying social proximity. In *IPTPS*, volume 5, page 1, 2007.

[12] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptology-Eurocrypt'04*, pages 1–19. Springer, 2004.

[13] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM*, 17(2):281–308, 1988.

[14] M. T. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. *Johns Hopkins Information Security Institute*, 2000.

[15] C. Hazay and K. Nissim. Efficient set operations in the presence of malicious adversaries. *Journal of cryptology*, 25(3):383–433, 2012.

[16] S. Jarecki and X. Liu. Fast secure computation of set intersection. In *Security and Crypto. for Networks*, pages 418–435. Springer, 2010.

[17] J. Katz and Y. Lindell. *Introduction to modern cryptography*. CRC Press, 2014.

[18] L. Kissner and D. Song. Privacy-preserving set operations. In *Adv. in Cryptology–CRYPTO 2005*, pages 241–257. Springer, 2005.

[19] X. Liao, S. Uluagac, and R. A. Beyah. S-match: Verifiable privacy-preserving profile matching for mobile social services. In *Dependable Systems and Networks (DSN)*, pages 287–298. IEEE, 2014.

[20] R. Merkle. A certified digital signature. In *CRYPTO'89*, pages 218–238. Springer, 1990.

[21] M. Nagy, E. De Cristofaro, A. Dmitrienko, N. Asokan, and A.-R. Sadeghi. Do i know you?: efficient and privacy-preserving common friend-finder protocols and applications. In *29th Annual Computer Security Applications Conference*, pages 159–168. ACM, 2013.

[22] M. Naor and K. Nissim. Certificate revocation and certificate update. *Selected Areas in Comm, IEEE Journal on*, 18(4):561–570, 2000.

[23] C. Papamanthou and R. Tamassia. Time and space efficient algorithms for two-party authenticated data structures. *Information and Communications Security*, pages 1–15, 2007.

[24] C. Papamanthou, R. Tamassia, and N. Triandopoulos. Authenticated hash tables. In *CCS'08*, pages 437–448. ACM, 2008.

[25] R. Tamassia. Authenticated data structures. *Algorithms-ESA 2003*, pages 2–5, 2003.

[26] M. Von Arb, M. Bader, M. Kuhn, and R. Wattenhofer. Veneta: Serverless friend-of-friend detection in mobile social networking. In *IEEE Conf. on Wireless and Mobile Comp.,*, pages 184–189, 2008.

[27] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman. Sybilguard: defending against sybil attacks via social networks. In *ACM SIGCOMM Computer Comm. Review*, volume 36, pages 267–278. ACM, 2006.

Appendix

1. Polynomial-Based Construction

We now give a construction based on polynomial representation suitable for the multi-party settings. Although the ADS-based protocol from Section 4 has higher practical impact even for constrained devices, this protocol is generalizable and widely applicable. We follow similar model from Section 4, where each user is assigned a unique high-entropy ID from a large domain upon registration. These IDs are then represented as the roots of the polynomial and exchanged upon friendship connection agreement among two users. For instance, Alice and Bob learn their IDs whenever they become friends. Similarly, we consider users to be malicious and the MSN to be trusted, as it already contains information about the connections. However, this computation can be performed offline and outsourced to external parties without leaking privacy of the users.

Protocol. Each MSN user, upon registration and friendship acquaintances composes their friendship set $\mathcal{S}$ as a polynomial, where each member of $\mathcal{S} = \{s_j\}_{1 \leq j \leq k}$ is represented as a root of $f \in R[x]$, such that $f(x) = \prod_{1 \leq j \leq k}(x - s_j)$. For simplicity, we consider the intersection of friend

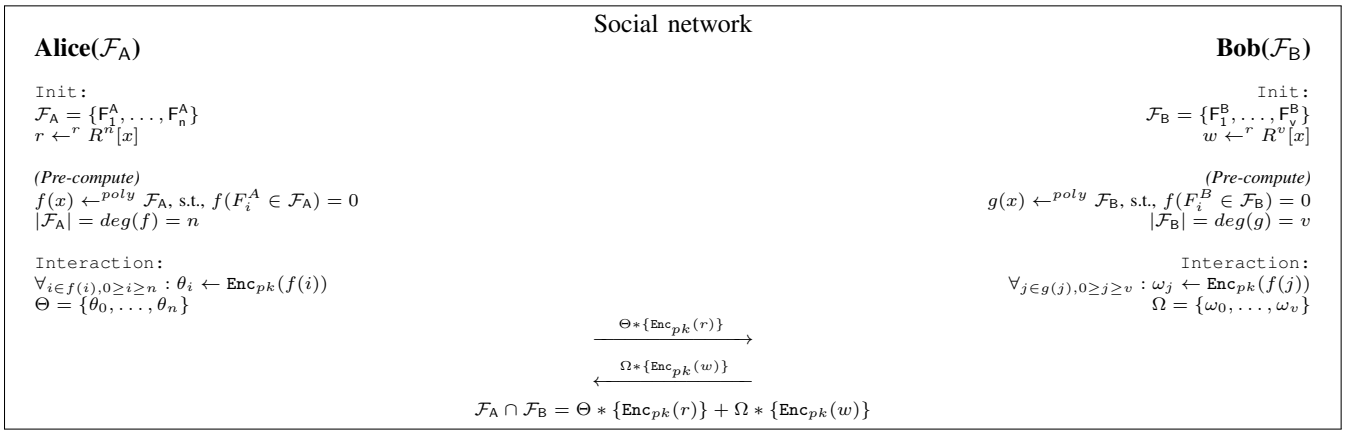

Figure 7: Two-party common friends matching. A and B learn only the intersection.

lists among two parties, Alice and Bob, with respective sets of friends $\mathcal{F}_A = \{a_1, a_2, ..., a_n\}$, and $\mathcal{F}_B = \{b_1, b_2, ..., b_m\}$, as depicted in Figure 7. The extension to multi-party settings is straightforward so that each party broadcasts the masked and encrypted set with all the other parties.

Hence, the computation of intersection of two sets $\mathcal{F}_A \cap \mathcal{F}_B$ represented as two polynomials f and g is the same as deriving common roots, which is given by the addition property, i.e., $f+g$. In order to produce this without learning extra information, we need to randomize the polynomials.

Let f, g be the polynomial representation of $\mathcal{F}_A$ and $\mathcal{F}_B$ respectively, and $deg(f) \geq deg(g)$. The intersection is computed using two masking random polynomials r and w as $f * r + g * w$, such that $r, w \leftarrow^r R^{deg(f)}[x]$ with coefficients chosen at random, and $r(x) = \sum_{i=0}^{deg(f)} r_i x^i$ and $w(x) = \sum_{i=0}^{deg(f)} w_i x^i$. This has been proposed by Kissner and Song [18], but they require both sets to have the exact same size in their protocol. In order to generalize it, we require r and w to have degree equal to the maximum degree specified by the MSN[2], and be calculated individually by each user. Otherwise, users with different numbers of friends will polynomials of different degree. The random polynomials r and w convert f and g to a normalized degree, such that $deg(f * r) = deg(g * w)$. Upon registration, Alice generates a new $r \leftarrow^r R^n[x]$, such that $deg(r) = \alpha$, where α is the maximum number of friends of a user in MSN.

Let $R[x]$ be a commutative ring with unity and an R-module G where $r \cdot g = g^r$ for $r \in G \vee g \in G$. In addition, let $\mathtt{Enc}_{pk} : R \rightarrow G$ be an additive homomorphic public key encryption. The following properties hold for all $a, b \in R$:

$$\mathtt{Enc}_{pk}(a + b) = \mathtt{Enc}_{pk}(a) \cdot \mathtt{Enc}_{pk}(b)$$
$$\mathtt{Enc}_{pk}(a \cdot b) = \mathtt{Enc}_{pk}(a)^b$$

Given the polynomials f and g with encrypted representation $\mathtt{Enc}_{pk}(f(x)) = \mathtt{Enc}_{pk}(f(0)), \ldots, \mathtt{Enc}_{pk}(f(deg(f)))$, the following properties also hold:

$$\mathtt{Enc}_{pk}((f + g)(x)) = \mathtt{Enc}_{pk}(f(x)) \cdot \mathtt{Enc}_{pk}(g(x))$$
$$\mathtt{Enc}_{pk}((f \cdot g)(x)) = \mathtt{Enc}_{pk}(f(x))^{g(x)}$$

In the end of the protocol, each user learns the resulting $\mathcal{F}_\mathsf{A} \cap \mathcal{F}_\mathsf{B}$ (or $\mathcal{F}_{\mathsf{u}_0} \cap \mathcal{F}_{\mathsf{u}_1} \cap \ldots \cap \mathcal{F}_{\mathsf{u}_\mathsf{n}}$ for multi-party settings with n users), without learning extra user IDs with overwhelming probability. In Figure 7 users perform the intersection using homomorphic additive encryption under the pk of the MSN, in order to protect the the communication, using the MSN as a trusted computation party. For instance, Alice and Bob which have both Carol and Dave on their friends list will just learn that two users, i.e., the intersection, at the end of the protocol and not others. The MSN acts as a TTP, and by assumption he already has knowledge on the connections. The protocol could be generalized to decentralized settings by sharing sk, pk and assuming that there are at least two honest users [18]. Although in some scenarios the MSN may already know the intersection of the sets, on cases that the sets are private to the MSN the masking part provides information theoretical security for $f * r$ such that r is a uniform random polynomial. Note that the random polynomials r and w are required to be long lived as if Alice (or Bob) uses a new $r' \neq r$, then $gcd(f * r, f * r') = f$.

Security This protocol is private and secure in the honest-but-curious model. Each player learns the intersection without learning any extra user ID not in the intersection, assuming that the additively homomorphic encryption is semantically secure. Therefore, on cases the MSN does not know $\mathcal{F}$ of each party, it will only learn the masked value and the end intersection.

The proof of correctness is an extension of the proof in Lemma 2 [18], where $deg(f) \neq deg(g)$. The extension to this extra possibility is straightforward. Normalizing each polynomial to degree α introduces false positives $\zeta \in \mathcal{F}_\mathsf{A} \cap \mathcal{F}_\mathsf{B}$, i.e., extra roots of the resulting polynomial. However, the probability that $\zeta \in \mathcal{F}_\mathsf{A} \cap \mathcal{F}_\mathsf{B}$ is true is negligible in the number of MSN users. The proof that the masked element is secure follows from Song construction [18].

2. We assume a fixed normalized degree size in the MSN.