Do I know you? – Efficient and Privacy-Preserving Common Friend-Finder Protocols and Applications

Marcin Nagy[1], Emiliano De Cristofaro[2], Alexandra Dmitrienko[3], N. Asokan[4], and Ahmad-Reza Sadeghi[5]

[1] Aalto University, Finland, marcin.nagy@aalto.fi
[2] PARC (a Xerox Company), U.S.A., me@emilianodc.com
[3] Fraunhofer SIT/CASED, Germany, alexandra.dmitrienko@sit.fraunhofer.de
[4] University of Helsinki and Aalto University, Finland, asokan@acm.org
[5] TU Darmstadt/CASED, Germany, ahmad.sadeghi@trust.cased.de

Abstract

The increasing penetration of Online Social Networks (OSNs) prompts the need for effectively accessing and utilizing social networking information. In numerous applications, users need to make trust and/or access control decisions involving other (possibly stranger) users, and one important factor is often the existence of common social relationships. This motivates the need for secure and privacy-preserving techniques allowing users to assess whether or not they have mutual friends.

This paper introduces the *Common Friends* service, a framework for finding common friends which protects privacy of non-mutual friends and guarantees authenticity of friendships. First, we present a generic construction that reduces to secure computation of set intersection, while ensuring authenticity of announced friends via bearer capabilities. Then, we propose an efficient instantiation, based on Bloom filters, that only incurs a constant number of public-key operations and appreciably low communication overhead. Our software is designed so that developers can easily integrate *Common Friends* into their applications, e.g., to enforce access control based on users' social proximity in a privacy-preserving manner. Finally, we showcase our techniques in the context of an existing application for sharing (tethered) Internet access, whereby users decide to share access depending on the existence of common friends. A comprehensive experimental evaluation attests to the practicality of proposed techniques.

1 Introduction

Online Social Networks (OSNs) play a key role in today's computing ecosystem, as social interactions/connections are increasingly used to enhance trust in, and usability of, a growing number of applications. Popular OSNs, such as Facebook, have become de-facto providers of online identities and are often used to enforce verification of personas and information. Numerous applications leverage technologies like OAuth [24] and OpenID [47] to authenticate users while relying on third-party services offered by OSN providers. Others connect to social network profiles and rely on data harvested from them, e.g., to verify self-reported information [45] or detect Sybil nodes [13].

In many realistic scenarios, users need to make access control decisions involving other (possibly stranger) users, e.g., for sharing rides [2] and cabs [1], to construct distributed computing platforms [42] and online dating services [3], or to base routing decisions for anonymous communications [35, 41]. One important trust-enhancing factor, potentially guiding such decisions, is the existence of previously established social relationships. For instance, an intuitive access control policy may be to only carpool with one's friends or friends-of-friends, or to base routing decisions on social proximity. However, the process of discovering common friends may harm the privacy of the two parties and that of their friends. At least one party needs to disclose the identity of his friends and, depending on the application scenario, this could reveal the identity of the user, and possibly even information about his lifestyle and social attitudes.

Motivated by the above issues, this paper presents the design and the implementation of a framework supporting secure discovery of common friends, which we denote as *Common Friends*. It allows two devices to assess whether their owners are friends or have mutual friends in a given social network, without reciprocally revealing any information about non-common friends.

We first introduce a generic construction that reduces the problem to secure computation of set intersection [23] and, at the same time, ensures authenticity of claimed friends using bearer capabilities [48]. We then propose a very efficient instantiation, based on Bloom filters [10], that only incurs a constant number of public-key operations (independent from the size of friend lists). Our proposed framework provides a clear and usable interface for application developers, enabling them to support access control decisions based on users' social proximity, independently of underlying cryptographic techniques. Finally, we integrate the *Common Friends* service into an existing application for sharing Internet connection [5], whereby users decide whether or not to share based on the existence of common friends. A comprehensive experimental evaluation attests to the practicality of proposed techniques.

1.1 Securely Finding Common Friends

As our main building block, we turn to *Private Set Intersection (PSI)* [23, 37, 17, 33, 29, 18], a cryptographic primitive allowing two parties, each inputting its own private set, to interact so that they only obtain, at most, the set intersection. If one considers the lists of users' friends as (unordered) sets, then PSI could be used to let users only learn the friends they share, by obtaining the set intersection. Alternatively, if only the number of shared friends is needed, one could use the *Private Set Intersection Cardinality (PSI-CA)* variant [23, 4, 27, 14], which only outputs the magnitude of the set intersection. Unfortunately, however, with PSI/PSI-CA, users could include identities of arbitrary friends in their input list (i.e., claim *non-existent* friendships). The *Authorized PSI (APSI)* variant [17, 15, 11], which extends PSI by ensuring that inputs are authorized by an appropriate authority, would not work either as it assumes that only one party's set is *certified* and the certification is performed by a single authority.

The work in [16] addresses the problem of claiming non-existent friendships by requiring users to provide a proof of prior relationship, via cryptographic credentials. Common friends are (privately) discovered following a relatively expensive technique resembling Secret Handshakes [6, 40],[1] where validity of certificates is verified obliviously to guarantee privacy while enforcing authenticity. Whereas, our approach is to use bearer capabilities [48] (aka sparse capabilities or bearer tokens): each user distributes a time-limited, randomly generated capability to his friends via a secure (i.e., authentic and confidential) channel. Possession of the capability represents a proof of an existing friendship, thus, users can input it into a cryptographic protocol, such as PSI, which reciprocally discloses only their common (authentic) friends. Using this approach, input sets to the PSI protocol are actually high-entropy objects, generated from a large space that is impractical to enumerate. Consequently, we do not need the full security of standard PSI techniques [23, 37, 17, 33] designed to work with potentially "predictable" items, such as names or identifiers. As we discuss later in Sec. 2.4, the unpredictability of capabilities allows us to instantiate PSI using a novel construction based on Bloom filters [10], with appreciably lower communication overhead and reduced number of modular exponentiations (constant vs. linear in the number of friends).

[1]Secret Handshakes allow two parties with certificates issued by the same organization to privately authenticate each other.

1.2 Contributions

In this paper, we present the design and the implementation of *Common Friends*, a framework that enables two devices to assess whether their owners are friends or share common friends in a given social network. *Common Friends* combines PSI with bearer capabilities [48] to ensure (1) privacy, i.e., users only learn information about their common friends, and (2) authenticity, i.e., one cannot falsely claim non-existent friendships. The *Common Friends* service is appealing in a number of realistic scenarios, where users can make trust and access control decisions, in a privacy-preserving manner, based on the existence (and possibly magnitude) of social relationships, e.g., for distributed computation, social networking, online dating, or ride-sharing.

In summary, this paper makes several contributions:

- The insight that when input sets include high-entropy items one can design more *efficient PSI schemes* than traditional PSI designed for low-entropy elements and the concrete design of such a PSI scheme based on Bloom Filters (Sec. 2.4);
- A detailed description of the design and implementation of a framework encapsulating the secure use of PSI protocols (independently from the actual implementation/variant) and bearer capabilities in the *Common Friends* scenario (Sec. 3). Our implementations provide a clear interface for developers to easily integrate *Common Friends* into their applications and use social proximity to guide trust and access control decisions. As a proof-of-concept, we successfully integrate it with a tethering application for sharing connectivity (Sec. 4).
- A performance evaluation that attests to the practicality of our solutions (Sec. 5).

2 The Common Friends Service

In this section, we describe the *Common Friends* service. We first introduce the desired security properties and then present our generic system design that reduces to the problem of private set intersection, followed by an efficient instantiation based on Bloom filters. Finally, we discuss the security of our proposals.

2.1 Security Goals and Attacker Model

We now define the secure common friend discovery functionality, along with relevant corresponding security goals.

Attacker Model. Before presenting security definitions, we introduce the attacker model. We consider *honest-but-curious* (aka semi-honest) adversaries, i.e., participants are assumed to follow protocol specifications but nonetheless attempt to infer more information, during or after protocol execution. In particular, we assume that legitimate participants will not disclose, or share, secret information.

Common Friends. The *Common Friends* service relies on a two-party protocol involving "initiator" I and "responder" R, on input the list of their friends $f(ID_I)$ and $f(ID_R)$, respectively. (ID_I and ID_R denote, respectively, the identity of I and R in a given social network). Specifically, we rely on three protocol variants securely realizing three functionality variants, presented in Table 1, and satisfying privacy and authenticity definitions discussed below.

Protocol Variant	R's output	I's output
Basic	$f(ID_I) \cap f(ID_R)$	$\perp$
Cardinality-only	$\|f(ID_I) \cap f(ID_R)\|$	$\perp$
Mutual Output	$f(ID_I) \cap f(ID_R)$	$f(ID_I) \cap f(ID_R)$

Table 1: Secure Common Friend Discovery Variants.

Initiator's Privacy. I's privacy is guaranteed if, on each possible pair of inputs $(f(ID_I), f(ID_R))$, R's view can be efficiently simulated on input: $f(ID_R)$ and either $f(ID_R) \cap f(ID_I)$ in the basic variant, or $|f(ID_R) \cap f(ID_I)|$ in the cardinality-only variant.

More precisely, let $\mathsf{View}_R(f(ID_I), f(ID_R))$ be a random variable representing the view of the responder R during a protocol interaction with inputs $f(ID_I), f(ID_R)$. Then, there exists a Probabilistic Polynomial Time (PPT) algorithm R*such that, in the basic variant:

$$\{\mathsf{R}^*(f(ID_R), f(ID_R) \cap f(ID_I))\}_{(f(ID_R),f(ID_I))} \overset{c}{\equiv} \{\mathsf{View}_R(f(ID_R), f(ID_I))\}_{(f(ID_R),f(ID_I))}$$

or, in the cardinality-only variant:

$$\{\mathsf{R}^*(|f(ID_R), f(ID_R) \cap f(ID_I)|)\}_{(f(ID_R),f(ID_I))} \overset{c}{\equiv} \{\mathsf{View}_R(f(ID_R), f(ID_I))\}_{(f(ID_R),f(ID_I))}$$

Responder's Privacy (Basic and Cardinality-Only Variants). If the functionality yields no output to the initiator, then responder's privacy is guaranteed if no information is disclosed about its input, not even the number or the identity of the common friends.

Description	Notation
Entities	
Server	S
Initiator User	I
Responder User	R
Generic User (can be either I or R)	U
Keys	
DH public key of U, I, R, resp.	PK_U, PK_I, PK_R
DH private key of U, I, R	SK_U, SK_I, SK_R
DH session key between I and R	K_{IR}
Data	
Certificate of server S	$Cert_S$
U's identifier in the social network	ID_U
Set of U's friends in the social network	$f(ID_U)$
Capability uploaded by user U	c_U
U's friends and their capabilities downloaded from S	R_U=$\{(ID_j, c_j) \mid ID_j \in f(ID_U)\}$
I's input set to PSI	$\overline{R_I}$=$\{(c_j\|\|PK_I\|\|PK_R) \mid (ID_j, c_j) \in R_I)\}$
R's input sets to PSI	$\overline{R_R}$=$\{(c_k\|\|PK_I\|\|PK_R) \mid (ID_k, c_k) \in R_R)\}$

Table 2: Notation.

That is, for every PPT adversary I* playing initiator's role, every initiator input set $f(ID_I)$, and any responder inputs $(f(ID_R)^{(0)}, f(ID_R)^{(1)})$ (of equal size), the views of I* if responder inputs $f(ID_R)^{(0)}$ and if responder inputs $f(ID_R)^{(1)}$ are computationally indistinguishable.

Responder's Privacy (Mutual Output Variant). Clearly, when the functionality yields, as output, the identity of common friends to both parties, responder's privacy is defined like initiator's privacy, i.e., I's view should be efficiently simulated with only its inputs and outputs. Specifically, let $\mathsf{View}_I(f(ID_I), f(ID_R))$ be a random variable representing the view of the initiator I during a protocol interaction with inputs $f(ID_I), f(ID_R)$. Then, there exists a PPT algorithm I*such that:

$$\{\mathsf{I}^*(f(ID_I), f(ID_I) \cap f(ID_R))\}_{(f(ID_I),f(ID_R))} \overset{c}{\equiv} \{\mathsf{View}_I(f(ID_I), f(ID_R))\}_{(f(ID_I),f(ID_R))}$$

Authenticity (Informal Definition). A user should not be able to falsely claim to have a common friend with the other party if there is no such common friend. Obviously, it follows that if the latter controls access to a resource on the basis of the existence of common friends, then, the former cannot succeed in getting access to this resource by claiming non-existent friendships and/or inflating the number of common friends.

2.2 System Description

Table 2 summarizes the notation used throughout this paper. The *Common Friends* service consists of two sub-protocols:

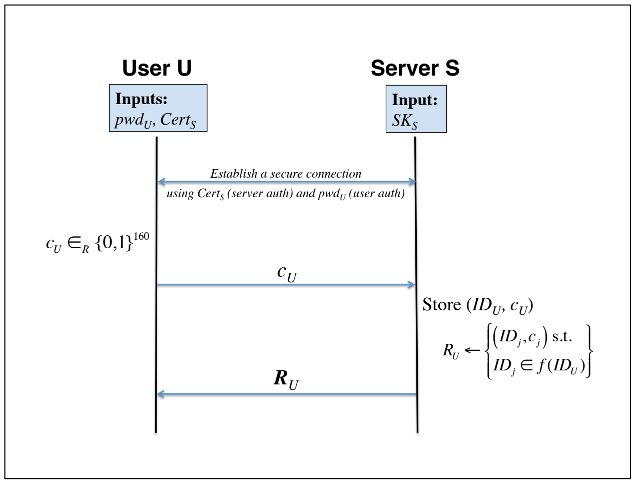

User U
Server S
Inputs: pwd_U, $Cert_S$
Input: SK_S
Establish a secure connection using $Cert_S$ (server auth) and pwd_U (user auth)
$c_U \in_R \{0,1\}^{160}$
c_U
Store (ID_U, c_U)
$R_U \leftarrow \{(ID_j, c_j) \text{ s.t. } ID_j \in f(ID_U)\}$
R_U

Figure 1: *Common Friends* Capability distribution.

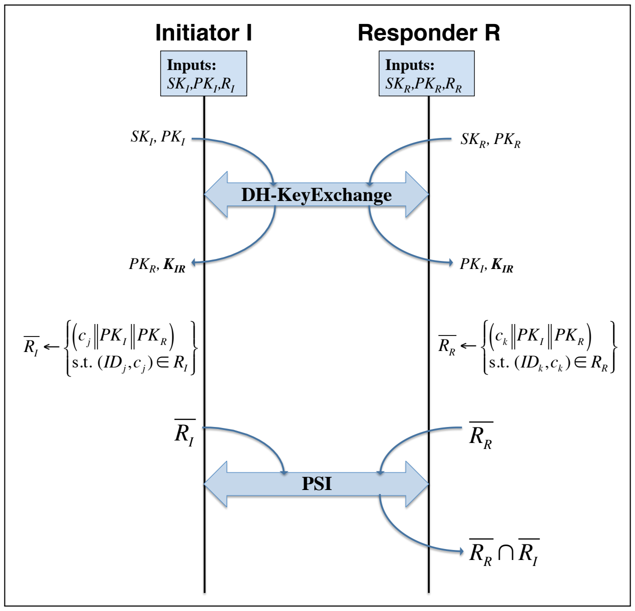

Initiator I
Responder R
Inputs: SK_I, PK_I, R_I
Inputs: SK_R, PK_R, R_R
SK_I, PK_I
SK_R, PK_R
DH-KeyExchange
PK_R, K_{IR}
PK_I, K_{IR}
$\overline{R_I} \leftarrow \{(c_j \| PK_I \| PK_R) \text{ s.t. } (ID_j, c_j) \in R_I\}$
$\overline{R_R} \leftarrow \{(c_k \| PK_I \| PK_R) \text{ s.t. } (ID_k, c_k) \in R_R\}$
$\overline{R_I}$
$\overline{R_R}$
PSI
$\overline{R_R} \cap \overline{R_I}$

Figure 2: The *friend finding* protocol in the *Common Friends* service (basic variant). First, I and R run a DH key exchange. Next, friend capabilities are bound to public keys and input sets to the PSI protocol are populated. Finally, on completion of PSI, R learns the common friends (and nothing else).

- A ***capability distribution*** protocol (Fig. 1) which is executed periodically by every user in the system, and
- A ***friend finding*** protocol (Fig. 2) which is executed between two users whenever they want to find common friends. (To ease presentation, we first present the basic protocol variant, and discuss further variants in Sec. 2.3, 2.4).

Capability distribution. We assume the presence of a server S, in the form of a social network application, which is used to distribute capabilities, as depicted in Fig. 1. First, user U and server S establish a secure channel. We use $Cert_S$ for server authentication and let the social network authenticate the user based on his password pwd_U. U periodically generates a random capability c_U from a large space (e.g., 160-bits) and uploads it to S via the established channel. S stores c_U, along with the social network user identifier ID_U, and returns the list $R_U = \{(ID_j, c_j)|ID_j \in f(ID_U)\}$, i.e., the identifiers and corresponding capabilities of each friend of U's. This protocol is run periodically in order to keep R_U up-to-date.

Observe that R_U contains capabilities that uniquely identify U's friends. They are distributed over a confidential and authentic channel, thus ensuring that U cannot claim non-existent friendships.

The capability distribution system is implemented on top of *PeerShare*, a generic scheme for securely distributing data among social groups, which we developed earlier [43].

Friend Finding. The *friend finding* protocol involves two users, I and R, members of the given social network. Let I be the user that initiates the protocol by contacting user R to find their common friends. The protocol, illustrated in Fig. 2, starts with I and R exchanging their (Diffie-Hellman) public keys, i.e., PK_I, and PK_R, respectively. The resulting shared Diffie-Hellman (DH) key K_{IR} will be used for two purposes: (a) to protect the messages exchanged as part of the Private Set Intersection (PSI) protocol protocol executed next and (b) to limit access if the PSI protocol determines that I and R have common friends.

To avoid man-in-the-middle attacks, the DH channel needs be cryptographically bound to the protocol instance. To this end, rather than inputing the set R_I (respectively R_R), I (R) builds the set $\overline{R_I}$ ($\overline{R_R}$), by appending DH public keys PK_I, PK_R to each capability in R_I (R_R). This transformation has negligible impact on performance, as PSI protocols hash each element in the list before further processing. The resulting sets:

$$\overline{R_I} = \{(c_j||PK_I||PK_R) \mid (ID_j, c_j) \in R_I)\}, \text{ and}$$
$$\overline{R_R} = \{(c_k||PK_I||PK_R) \mid (ID_k, c_k) \in R_R)\}$$

are used as inputs to the PSI protocol executed next.

Note that the friend finding protocol can trivially be extended to determine whether two users are *direct* friends of each other, provided that each user U adds c_U to the list of capabilities given in input to the PSI protocol.

2.3 PSI vs PSI-CA Instantiations

We now present the PSI instantiations we use to privately intersect users' capabilities, as stated in the *friend finding* protocol.

Available PSI Protocols. A few different instantiations of PSI have been proposed, with different security models, assumptions, and complexities. PSI can be constructed using generic Garbled Circuits [51, 29], Oblivious Polynomial Evaluation [23, 37], or Oblivious Pseudo-Random Functions (OPRFs) [26, 32, 17, 33].

According to the performance evaluations in [18], the most efficient protocol is the OPRF-based construction by De Cristofaro and Tsudik [17]. It is secure, in the presence of honest-but-curious adversaries, under the OneMore-RSA assumption in the Random Oracle Model (ROM) [8]. Assuming that m is the size of set held by one party (the Responder), and n that of the other party (the Initiator), the protocol in [17] incurs $O(m+n)$ computational and communication complexities. In particular, the former is dominated by $O(m+n)$ modular exponentiations (specifically, RSA signatures), while the latter corresponds to transferring $2n$ group elements and m outputs of a cryptographic hash function.

PSI-CA Variants. A possible alternative could be to use a more restrictive variant that only yields the number of common friends, and not their identities. To this end, we turn to Private Set Intersection Cardinality (PSI-CA) protocols [23, 4, 27, 14]: PSI-CA allows two parties, each holding a private set, to interact in a cryptographic protocol such that one party learns the magnitude of the set intersection (and nothing else), while the other obtains nothing. Clearly, PSI-CA could be used instead of PSI to let users learn only how many friends they have in common. This corresponds to the ***cardinality-only*** protocol variant presented in Sec. 2.1 On the one hand, this approach provides strictly more stringent privacy guarantees. On the other hand, however, certain application scenarios may require users to know the specifics of which friends are common, e.g., to make better informed access control/trust decisions. In this case, PSI would be the preferred option.

To the best of our knowledge, the most efficient PSI-CA protocol is presented in [14], with honest-but-curious security in ROM, under the OneMore-DH assumption [8]. Complexities are similar to the PSI protocol in [17], i.e., linear in the size of sets. Specifically, computation complexity is dominated by $O(m+n)$ modular exponentiations (in prime order groups with random exponents taken from a subgroup), while communication complexity corresponds to transferring $2m$ group elements and n outputs of a hash function (assuming that m is the size of set held by the initiator, and n that of the responder).

2.4 Improving Efficiency with Bloom Filter based PSI (BFPSI)

Recall from Sec. 2.2 that capabilities are generated at random from a large space, thus, they are high-entropy objects and impractical to enumerate. Consequently, we do not necessarily need to use traditional PSI protocols (designed to work with low-entropy, possibly enumerable, items): since input sets only include high-entropy items, we can rely on more efficient techniques, which realize the same set-intersection functionality, with same provable security properties.

Intuition. A straightforward approach for private set intersection is to let both parties hash each item in their set (using a cryptographic hash function) and send the results to each other. Since the hash is one-way, parties cannot invert the hash function and can only learn the set intersection by finding matches between the received hashes and those computed over their own set items. However, if set items are low-entropy objects, a malicious party could test, offline, for the presence of a given item in counterpart's set, regardless of whether or not it belongs to the intersection. As a consequence, PSI protocols need more sophisticated techniques, relying on public-key cryptography, to prevent parties from succeeding in such attacks.

On the other hand, if set items are high-entropy objects, e.g., generated at random from a large space as in the case of bearer capabilities, then the testing attack would not work since it is impractical to enumerate sets. Thus, we notice that the use of traditional PSI is actually an "overkill" and the naive hash-based approach described above suffice to realize the private set intersection functionality. Besides removing the need for a number of public-key crypto operations (at least) linear in the size of sets, this approach enables the use of optimization/compression techniques, like Bloom filters [10], which we present below. We anticipate that the resulting Bloom filter based protocol will disclose the identity of common friends to both parties. Thus, it corresponds to the ***mutual output*** protocol variant, discussed in Sec. 2.1.

Bloom Filters [10]. A Bloom Filter (BF) is a data structure used to efficiently represent and test sets. Let us consider a set $X = \{x_1, \ldots, x_\alpha\}$ of α elements, and an array of β bits initialized to 0. The notation BF(j) denotes the position j in the BF. The Bloom Filter uses γ independent cryptographic hash functions $h_1, \ldots, h_\gamma$ with range $1, \ldots, \beta$, salted with random (periodically refreshed) nonces so that it cannot be tracked over time. For each element $x \in X$, $BF(h_i(x))$ is set to 1 for $1 \le i \le \gamma$. To check whether an element y is a member of X, we simply test if BF$(h_i(y))$ equals 1 for all $1 \le i \le \gamma$.

Note that Bloom filters introduce false positives, i.e., an element might seem present although it was never inserted. The probability p of false positive can be approximated as:

$$p = (1 - (1 - 1/\beta)^{\gamma \cdot \alpha})^\gamma$$

It follows that the optimal value of γ that minimizes p is:

$$\gamma = \frac{\beta}{\alpha} \ln 2$$

Description	Notation
Data	
Bloom Filter sent by I	BF_I
Random value chosen by I	$irand$
Random value chosen by R	$rrand$
Challenge set containing HMAC values using $ckey$ of elements in intersection	$cset$
Response set containing HMAC values using $rkey$ of elements in intersection	$rset$
$\overline{R_I} \cap \overline{R_R}$ with possible false positives	X'
Actual $\overline{R_I} \cap \overline{R_R}$	X
Algorithms	
DH Key-Exchange (I)	$K_{IR} \leftarrow$ DH-Key(SK_I, PK_R)
DH Key-Exchange (R)	$K_{IR} \leftarrow$ DH-Key(SK_R, PK_I)
Message Authentication Code	HMAC$(key, message)$
Key Derivation Function	KDF$(\cdot, \cdot)$
Keys	
HMAC keys used by I and R, resp.	$ckey, rkey$

Table 3: New notation introduced for Bloom Filter based PSI.

Hence, the optimal size of the filter, for a desired false positive probability p, using the optimal value of γ, can be estimated as:

$$\beta = \left\lceil \frac{-\log_2 p}{\ln 2} \right\rceil \times \alpha \qquad (1)$$

where $\alpha = max(m, n)$ in our *Common Friends* setting, assuming m is the number of Initiator's friends and n – that of Responder's.

Using Bloom Filter based PSI (BFPSI). Fig. 3 illustrates how to use a Bloom Filter based PSI (BFPSI) to realize the *friend finding* protocol. New notation is summarized in Table 3.

As in the generic protocol description, interaction starts with user I engaging user R, followed by a DH key exchange. I and R use input sets, $\overline{R_I}$ and $\overline{R_R}$, respectively, constructed as before. I inserts every element of $\overline{R_I}$ into a Bloom filter BF_I which is then sent to R. R can now discover the set X' of friends potentially shared with I by testing every element of $\overline{R_R}$ for membership in BF_I.

Although the length of the Bloom filter primarily depends on number of friends I and R have, it is also determined by the false positive probability value. Observe that p affects not only communication but also computation overhead since the lower the value of p is the higher the number of hash operations required to insert one element into the Bloom filter. Therefore, a practical implementation cannot afford to choose a value of p that is negligible by the usual standards for cryptographic algorithms. In our implementation, we choose $p = 10^{-4}$. However, we now need to account for the possibility that the protocol returns more common friends than there actually exist, due to the small yet non-negligible probability of false positives. Since the input sets are impractical to enumerate, users cannot maliciously exploit the false positive rate to claim unwarranted friendships or violate counterpart's privacy. Nonetheless,

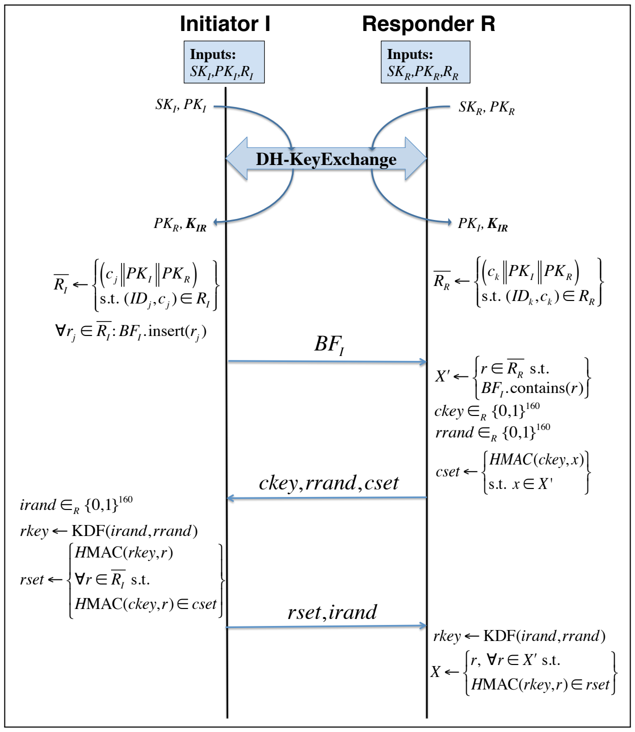

Initiator I
Responder R
Inputs: SK_I,PK_I,R_I
Inputs: SK_R,PK_R,R_R
SK_I, PK_I
SK_R, PK_R
DH-KeyExchange
PK_R, $\boldsymbol{K_{IR}}$
PK_I, $\boldsymbol{K_{IR}}$
$\overline{R_I} \leftarrow \left\{ \begin{matrix} (c_j \Vert PK_I \Vert PK_R) \\ \text{s.t. } (ID_j, c_j) \in R_I \end{matrix} \right\}$
$\overline{R_R} \leftarrow \left\{ \begin{matrix} (c_k \Vert PK_I \Vert PK_R) \\ \text{s.t. } (ID_k, c_k) \in R_R \end{matrix} \right\}$
$\forall r_j \in \overline{R_I} : BF_I.\text{insert}(r_j)$
BF_I
$X' \leftarrow \left\{ \begin{matrix} r \in \overline{R_R} \text{ s.t.} \\ BF_I.\text{contains}(r) \end{matrix} \right\}$
$ckey \in_R \{0,1\}^{160}$
$rrand \in_R \{0,1\}^{160}$
$cset \leftarrow \left\{ \begin{matrix} HMAC(ckey, x) \\ \text{s.t. } x \in X' \end{matrix} \right\}$
$ckey, rrand, cset$
$irand \in_R \{0,1\}^{160}$
$rkey \leftarrow \text{KDF}(irand, rrand)$
$rset \leftarrow \left\{ \begin{matrix} \text{HMAC}(rkey, r) \\ \forall r \in \overline{R_I} \text{ s.t.} \\ \text{HMAC}(ckey, r) \in cset \end{matrix} \right\}$
$rset, irand$
$rkey \leftarrow \text{KDF}(irand, rrand)$
$X \leftarrow \left\{ \begin{matrix} r, \forall r \in X' \text{ s.t.} \\ \text{HMAC}(rkey, r) \in rset \end{matrix} \right\}$

Figure 3: Friend Finding using Bloom filter based PSI (BFPSI).

we need the means for I and R to verify that the output of the protocol does indeed consist of their mutual friends (and remove the false positives).

To this end, we introduce a simple challenge-response protocol, illustrated in Fig. 3, where R asks I to prove knowledge of the capabilities that constitute the set X as follows:

- R first constructs a candidate intersection set X' by testing every element of $\overline{R_R}$ for presence in BF_I.
- R then constructs a challenge set $cset$ consisting of HMACs (key-hashed message authentication codes) computed on every item in X'. A freshly generated random key $ckey$ is used as the key for the HMACs in $cset$. Note that we need HMACs, rather than MACs, to ensure the one-wayness of the function.
- R sends $cset$ and $ckey$, along with a random coin $rrand$.
- I can construct HMACs for each element of its own $\overline{R_I}$ using the received $ckey$ as the key and check whether the resulting HMAC is present in $cset$.
- For each of these elements, I computes HMACs with a key $rkey$, obtained via a key derivation function using its own random coin $irand$ and $rrand$. The resulting response set $rset$ is sent to R, along with $irand$.
- R can recompute $rkey$, construct a HMAC on every element of X' using $rkey$ and check if the resulting HMAC is found in $rset$. If it is, then that element is added to X.

Remark: Relying on Bloom filters to realize private intersection of high-entropy items yields constructions incurring a constant number of public-key cryptography operations and a reduced communication overhead – a remarkable performance gain which we further analyze in Sec. 5.

2.5 Security Considerations

We now analyze the security of our proposed techniques, following security requirements outlined in Sec. 2.1.

Authenticity. Our proposed techniques guarantee authenticity of claimed friendships, via bearer capabilities. These, by definition, confer the same authorizations on anyone who holds them, One potential concern is that users could maliciously re-distribute them to other users. However, we assume that: (1) capabilities are stored securely, and (2) parties who receive capabilities legitimately (honest but curious) do not share them with others who are not authorized to receive them. We argue that such assumptions are reasonable in the context of the *Common Friends* service, which is designed to be implemented on mobile devices. These are usually equipped with software and hardware platform security features that can ensure application-specific secure storage [38].

Nonetheless, it is trivial to extend our constructions to support "friendship certificates", i.e., signatures issued on friends' public keys. Friends can securely exchange public keys via the server S, in the same way they exchange bearer capabilities. At the end of the *friend finding* protocol interaction, once R has determined the candidate intersection set X', it can ask I to confirm possession of a valid friendship certificate from each entity in X'.

Privacy. The proposed techniques reduce the problem of privately discovering common friends to secure computation of set intersection. Thus, privacy of our proposals stem from the security of the underlying protocol that *Common Friends* instantiates, e.g., the PSI construction in [17], the PSI-CA variant in [14], or the BFPSI variant we introduce. The security of the latter relies on the fact that items are taken at a random from a large space, thus, while we do not claim it achieves security comparable to traditional PSI protocols, we can demonstrate that the BFPSI construction reveals nothing besides the intended output.

Initiator's Privacy (Proof Sketch). We prove that responder R learns nothing about initiator I's items outside intended output, regardless of the protocol variant. In the basic and cardinality-only variant, this follows immediately from the security of the underlying PSI [17] and PSI-CA protocols [14], respectively. Whereas, in the mutual-output variant (which relies on BFPSI), I's privacy follows from the one-way property of the hash functions used to construct the Bloom filter and the unpredictability of input sets (bearer capabilities). Recall that, in ROM, the hash of an unpredictable function is a PRF, thus, if R could learn more than the intersection, it would be violating the PRF properties. That is, let us assume that:

$$\{\mathsf{R}^*(f(ID_R), f(ID_R) \cap\ f(ID_I))\}_{(f(ID_R),f(ID_I))} \overset{c}{\not\equiv} \{\mathsf{View}_R(f(ID_R), f(ID_I))\}_{(f(ID_R),f(ID_I))}$$

Then, there must exist one item $c^* \in X'$ s.t. $c^* \notin f(ID_R) \cap\ f(ID_I)$, i.e., $BF_I.contains(c^*) = 0$. Since c^* is drawn from a large space (computationally infeasible to enumerate), it must hold that BF is invertible, thus, the hash function used for constructing the Bloom filter is not a secure PRF.

Responder's Privacy (Proof Sketch). Recall that, in the basic and cardinality-only variants, I has no output from the protocol, and, R's privacy immediately stems from the security of the underlying PSI [17] and PSI-CA protocols [14], respectively, thus, I's view should be efficiently simulated with only its inputs and outputs.

R's privacy in the the mutual-output variant, i.e., protocol in Fig. 3, is also straightforward. Recall that R sends I $cset$ with the HMAC of all items in the intersection (which is intended output of the protocol). Recall that Bloom filters may introduce false positives, however, if I could learn something about the false positive found by R, then the HMAC used to construct $cset$ must not be a secure HMAC. However, this is impossible since, in ROM, HMAC is known to be pseudo-random [7].

3 Framework Design

We now present the design of our *Common Friends* framework and discuss how developers can integrate it into their own applications. We argue that it is crucial to abstract away the details of underlying cryptographic techniques, so that application developers, who might not be cryptography experts, can easily rely on secure and privacy-preserving techniques to discover common friends (and possibly use them to guide trust and/or access control decisions). Our goal is to do so in such a way that application developers:

(1) can use an intuitive and well-defined API;
(2) only need to specify the kind of functionality they need (e.g., finding *how many* or *which* common friends);
(3) do not need to refactor their code if a new PSI or PSI-CA technique (perhaps more efficient or relying on different assumptions) becomes available, but only update the *Common Friends* library.

Framework Description. Fig. 4 illustrates how applications can use the *Common Friends* service. Tables 4 and 5 summarize the details of employed methods and containers. To use the *Common Friends* service, application

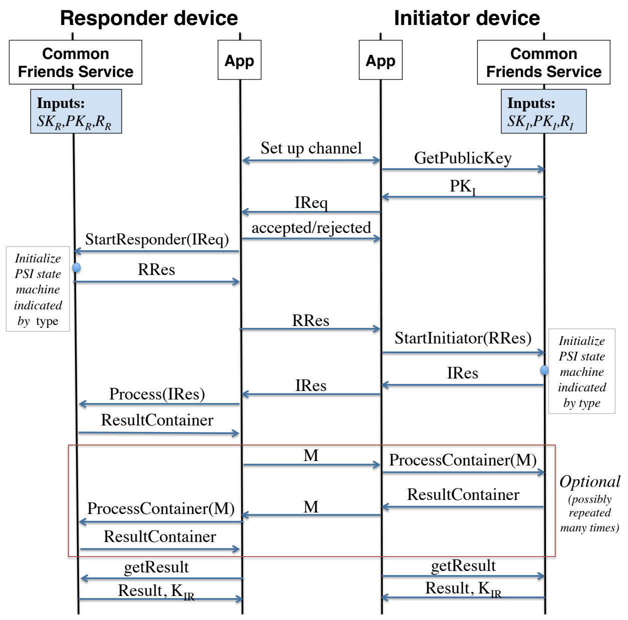


Figure 4: Common Friend Service framework. Optional message exchanges involving *ProcessContainer* invocations are used by PSI protocols that require more than three message flows.

Name	Input	Output	Invoker	Description
$StartResponder$	$IReq$	$RRes$	R	Triggers PSI
$StartInitiator$	$RRes$	$IRes$	I	Triggers PSI; extracts K_{IR}
$Process$	$IRes$	RC	R	Processes IRes
$ProcessContainer$	M	RC	R,I	PSI variant specific method
$getResult$	-	PR	R,I	Gets final PSI result and shared key K_{IR}

Table 4: *Common Friends* service interface.

Notation	Description	Constituent Data
type	Type of *Common Friends* service needed	PSI type, hop length, number of friends
IReq	IRequest	supported algorithms, PK_I
RRes	RResponse	accepted $type$, PK_R, PSI protocol specific payload (RDC)
IRes	IResponse	PSI protocol specific payload (IDC)
RC	Result Container	PSI state machine status, optionally M to send
M	Message	PSI variant specific content
PR	Protocol Result	PSI final result, secret key K_{IR}

Table 5: Parameters in the *Common Friends* service interface.

instances on a responder device R and a initiator device I first set up a communication channel between them. Before starting a PSI instance, I sends a request *IReq* to R consisting of (a) I's Diffie-Hellman Public Key PK_I and (b) the type of protocol I wants to run. Currently, we support two different types: a protocol that only outputs the cardinality of the intersection, i.e., PSI-CA, and one protocol that outputs the actual intersection set i.e., BFPSI (for improved efficiency compared to traditional PSI).

R's application instance can choose to accept or reject the proposed protocol type and send a notification to I in either case. On accept, it starts a protocol run by invoking the *StartResponder*, with *IReq* as an argument. This method performs the first step of the PSI protocol which returns a response in the form of an *RRes* message. R sends *RRes* to I, which starts its *Common Friends* service engine. This returns an *IRes* message that is transported back to R. R invokes the *Process* method with *IRes* as the parameter which returns a *ResultContainer* object which contains a status field that can take one of two values: *done* or *wait*, and an optional message *M*.

The three protocol messages (*IReq, RRes, IRes*) are mandatory for all PSI schemes. Some PSI protocols (e.g., PSI-CA in [14]) contain only three flows. They can be acommodated using the three messages. Others (e.g., BFPSI) may need more message exchanges. To accommodate this variation, *Common Friends* framework allows the possibility of an optional phase that can be repeated as many times as needed by the PSI protocol being used.

The application instances determine whether to carry out these optional exchanges by examining *ResultContainer* returned by the PSI protocol engine and performing the following operations:

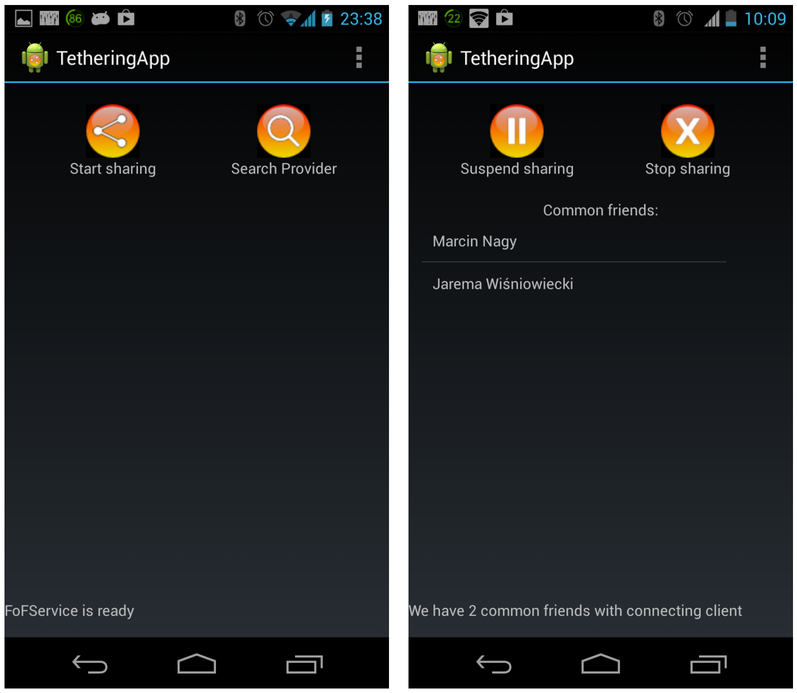
- If it contains a message M then transfer M to peer.
- If its status component is *wait*, wait for peer to respond. Otherwise (status is *done*), call *getResult* to extract PSI result.

While the optional phase is being executed, the application instances simply act as conduits for their respective PSI protocol engines to communicate with each other. Depending on the type of PSI, the result of PSI may be empty for the initiator. As mentioned before, K_{IR} can be used for subsequent access control.

Plugging in Bloom filter based PSI. To plug the BFPSI protocol (described in Sec. 2.4) into the *Common Friends* service, we need to provide BFPSI-specific implementations of each of the methods identified in Table 4. Constructing the Bloom filter BF_I and testing whether elements of $\overline{R_R}$ are present in BF_I are implemented within the *StartInitiator* and *StartResponder* methods, respectively. The creation of the challenge set (to eliminate false positives) is implemented in the *Process* method on R and the corresponding creation of the response set is implemented in the *ProcessContainer* method on I. The *ProcessContainer* method on R processes the response set and populates the intersection.

4 Implementation

We now present the implementation of *Common Friends* on Android, and its integration with an existing tethering application from our prior work [5].



FoFService is ready

Common friends:

Marcin Nagy

Jarema Wiśniowiecki

We have 2 common friends with connecting client

(a) (b)

Figure 5: Screenshot of the Tethering Application: View before *Common Friends* protocol run (a) and view presenting results of *Common Friends* protocol (b).

Framework. We implemented *Common Friends* (Sec. 3) as a simple Android service that exposes its interface to third party applications via Android Interface Definition Language (AIDL) declarations. Communication between the service and application uses Android specific AIDL interface. (However, the core service is implemented in standard Java, thus, could be executed on any device equipped with a Java Virtual Machine). The application instances on I and R are responsible for setting up a communication channel to exchange the protocol messages received from the *Common Friends* Service. Protocol messages are containers implemented as *Parcelable* and *Serializable* Android classes, and are opaque to the calling applications. Application instance on R chooses the protocol variant to use. Currently our implementation supports PSI-CA and BFPSI, implemented as plugins in *Common Friends* framework.

Developers can embed the *Common Friends* functionality into their applications by simply adding the *Common Friends* Service AIDL interface declaration to their application source tree, together with the container classes. The framework can also be extended with additional PSI protocol engines: abstract class *AlgorithmEngine* provides basic primitives (methods: *StartResponder*, *StartInitiator*, *Process*, and optionally *ProcessContainer*) for future extensions with new PSI protocols.

PSI-CA. We implemented the PSI-CA protocol proposed in [14], using the standard Android cryptography provider (Bouncy Castle). We used Elliptic Curve Diffie-Hellman (ECDH), based on the NIST P-192 curve [44], to implement both the Diffie-Hellman key agreement (needed for integrating PSI-CA into the *Common Friends* service) and the modular arithmetic operations within the PSI-CA protocol [14].

Bloom Filter based PSI (BFPSI). To implement the BFPSI protoocol (see Sec. 2.4) we selected a fixed false positive probability of $p = 10^{-4}$, and used Bloom filter with length calculated according to Equation 1. Diffie-Hellman key exchange was as in the case of PSI-CA. We used HMAC-SHA-1 to instantiate HMAC and SHA-1 for KDF$(\cdot,\cdot)$. Bloom filter operations were implemented using code available from https://github.com/MagnusS/Java-BloomFilter with SHA-1 as the underlying hash function.

Tethering Application. To demonstrate the applicability of our techniques to real-world scenarios where access control decisions are securely made based on the existence of common friends, we also extended an application for tethering (proposed in our prior work [5]) by integrating it with our *Common Friends* service.

The application allows a device to either act as a WiFi tethering access point, or as a WiFi tethering client. We ex-

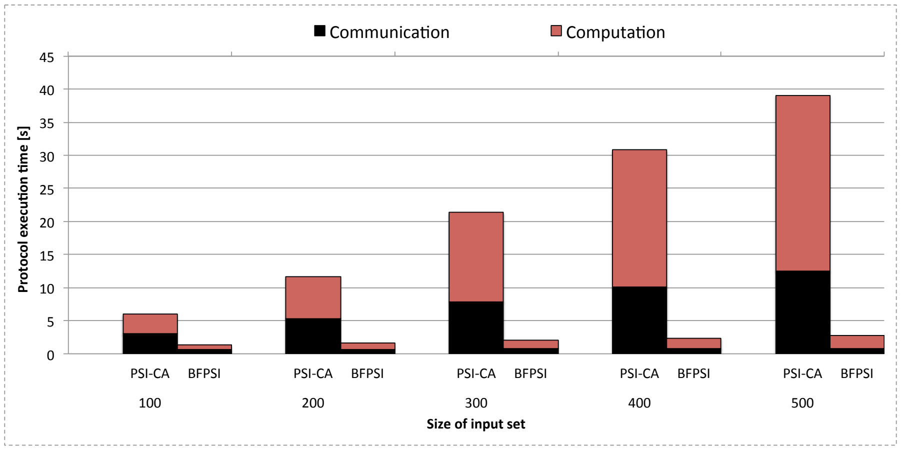


Figure 6: Comparison of BFPSI and PSI-CA protocol performance.

tend the application by allowing a user to choose whether or not to authorize another user to connect to his access point based on whether or not the two are friends on a given social network or have some common friends. The device acting as access point is turned into a "hotspot" using the Android WiFi Manager API, and plays the role of R. It also opens a Bluetooth socket to listen for incoming tethering requests. Our tethering service is advertised by a specific Universal Unique Identifier (UUID), which is used in the service discovery.

The device acting as a tethering client plays the role of I, and initiates Bluetooth service discovery procedure looking for a suitable WiFi tethering access point. On successful discovery, both applications establish a Bluetooth connection in RFCOMM mode and run BFPSI or PSI-CA to learn which or how many friends are common. Based on gathered information, R decides whether or not to send the WiFi SSID and password to I over the secure channel (using the previously established Diffie-Hellman shared key K_{IR}). Fig. 5 presents screenshots of the tethering application.

Code Availability: Source code of our implementations can be made available for research use upon request.

5 Performance Analysis

This section present an empirical evaluation of the performance of *Common Friends* service when using PSI-CA [14] vs. BFPSI (Sec 2.4). Specifically, we analyze the computational, communication and energy consumption costs incurred by them.

Computation and Communication Overhead. To measure running times and bandwidth overhead, we performed experiments (over 30 trials) on a Samsung Galaxy Nexus smartphone running Android 4.2 API 17 and a Samsung Galaxy Tablet GT-P3100 running Android 4.1.2 API 16, connected over Bluetooth.

We made the assumption that both parties have the same number of friends and varied this number in the range $\{100, 200, 300, 400,$
$500\}$. The intersection of the sets was always at 10% of the set size.

Processing time. Total average execution time increases linearly for both protocols as expected, but at different rates (Fig. 6). In particular, Table 6 shows that, with 5-fold increases in set sizes, computation time for PSI-CA increases by several seconds, whereas, with BFPSI it increases by less than half a second.

Communication bandwidth. As shown in Table 7, the total number of bytes exchanged also increases linearly for both protocols. However, the amount of data exchange is significantly larger for PSI-CA, by a factor of almost 6 compared to BFPSI.

Power Analysis. It is well-known that energy consumption for sending/receiving a message increases with the message size [46, 49]. As a result, the use of BFPSI protocol can have a lower impact on battery life, which is crucial for mobile users. To study this aspect, we performed a power analysis of the *Common Friends* service with input sets of 200 items, using two Samsung Nexus S devices running the CyanogenMod 9.1.0-crespo Android release and a laptop running a power analysis tool for Android devices called Little Eye.[2] Currently, the tool is optimized for precise power analysis measurements only on certain device models, but it can be used for rough estimates on others as well. (In general, power analysis on mobile devices at the granularity of applications is known to be a challenging problem [49], however, our estimates suffice to provide an

[2]http://www.littleeye.co/

Input size	BFPSI				PSI-CA			
	Comm. [s]		Comp. [s]		Comm. [s]		Comp. [s]	
	avg	std	avg	std	avg	std	avg	std
100	0.649	0.061	0.652	0.061	3.053	0.089	2.999	0.24
200	0.646	0.049	1.047	0.062	5.307	0.373	6.401	0.358
300	0.72	0.086	1.33	0.088	7.904	0.212	13.438	0.195
400	0.811	0.066	1.597	0.056	10.099	0.16	20.709	0.799
500	0.816	0.085	1.968	0.099	12.543	0.176	26.535	0.69

Table 6: Average values and standard deviations of computation and communication time (in seconds) for one BFPSI and PSI-CA protocol transaction for various input set sizes.

Input size	BFPSI	PSI-CA
100	2,548	34,833
200	3,424	67,933
300	4,292	100,399
400	5,168	133,222
500	6,036	166,029

Table 7: Total number of bytes exchanged in a protocol run for increasingly large sets.

intuition of power requirements for continuous executions of the *Common Friends* service.)

Fig. 7 and Fig. 8 show power diagrams for BFPSI and PSI-CA protocols respectively, when executed 5 times (x-axis shows elapsed time and the peaks correspond to the five executions). We also calculated overall energy consumed by *Common Friends* during each test. Measurements include CPU power and communication, but exclude power consumed by the device screen. According to our measurements, BFPSI execution required 0.18 mAh, while PSI-CA utilized 0.55 mAh, thus indicating that BFPSI protocol consumes approximately 3 times less energy than PSI-CA.

To confirm that observed differences are not induced by the power consumption characteristics of the device model we used, we repeated the tests on a different device model (Samsung Galaxy S3). The resulting measurements were 0.12 mAh and 0.38 mAh for BFPSI and PSI-CA, respectively. Thus, we conclude that ratio of power consumption between BFPSI and PSI-CA remains the same across different device models.

Discussion. Instantiating *Common Friends* with BFPSI clearly offers improved performance compared to using PSI-CA. BFPSI requires fewer computations (constant vs linear number of public-key operations), lower bandwidth and power consumption. As a result, the use of BFPSI in *Common Friends* service is likely to offer a better user experience and support more frequent runs. On the other hand, if one only wants to disclose the number of common friends, then one needs to tolerate the additional overhead incurred by the use of PSI-CA.

Finally, observe that traditional PSI and PSI-CA protocols incur similar complexities (e.g., they both require a number of public-key operations linear in set sizes). Therefore, we can expect that, when applied to finding common friends, BFPSI will exhibit performance gains over traditional PSI protocols very close to those observed over PSI-CA. This confirms our intuition that, while PSI protocols are designed to deal with low-entropy input sets, we do not need their full security in the context of finding common friends, thus enabling appreciably improved efficiency.

6 Related Work

Motivated by the increasing influence of social networks, a few techniques have focused on secure operations on users' social network profiles, such as, matching of common attributes, interests, and (similar to our work) friends. Li et al. [39] formally analyze the problem of privacy-preserving personal profile matching and propose a set of protocols that leverage PSI and/or PSI-CA to securely match attribute sets of different users. Dong et al. [19] represent a user's profile as a vector and measure social proximity via private vector dot product [31], while Zhang et al. [53] extends it to improve its granularity with finer grained attributes.

Zhang et al. [52] also propose a privacy-preserving verifiable profile matching scheme which is based on symmetric cryptosystem and thus improves efficiency. It relies on a pre-determined ordered set of attributes and uses it as a common secret shared by users. However, the scheme is not applicable to unordered sets of attributes such as random capabilities (as in our case).

In VENETA [50], Von Arb et al. use PSI for privacy-preserving matching of common entries in the users' address books to support decentralized SMS-messaging via Bluetooth. VENETA does not address the problem of malicious users claiming non-existent friendships, but only suggests to limit the size of input sets to 300. Huang et al. [28] present an Android app that instantiates PSI with garbled circuits and lets users privately find common entries in their address books. Besides being vulnerable to the same potential attack as in VENETA, the work in [28] reports timing values of 150 seconds to match 128 contacts, thus raising concerns about its practicality, even though Carter et al. [12] recently present a faster prototype implementation based on specialized secure function evaluation protocols.

De Cristofaro et al. [16] present a framework for private discovery of common social contacts. In their scheme, users need to provide a proof of prior relationship to claim a given friendship (specifically, a cryptographic certificate). Common friends are privately discovered following a technique resembling Secret Handshakes [6, 40], where validity of certificates is verified obliviously to guarantee privacy while enforcing authenticity. However, this scheme incurs significantly higher computation overhead

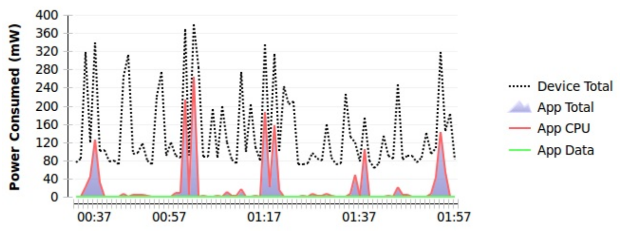


Figure 7: Power Consumption of BFPSI protocol.

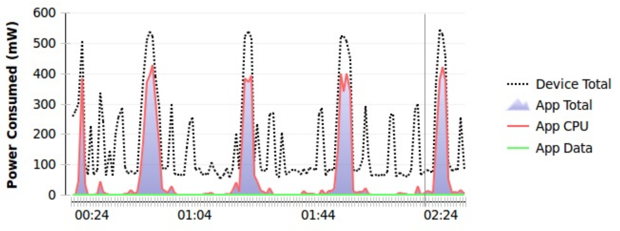


Figure 8: Power Consumption of PSI-CA protocol.

compared to our solutions relying on bearer capabilities and BFPSI. Specifically, [16] incurs a number of expensive modular exponentiations linear in the number of friends (and a quadratic number of modular multiplications) and a communication overhead similar to traditional PSI techniques.

Our previous work [5] presents a framework for resource sharing (e.g., Internet connectivity) in ad-hoc mobile networks where users enforce access control based on whether users are friends in a given social networks or at least have some friends in common. In [5], we mentioned the possibility of using a social network application to exchange capabilities between social network users as proofs of the friendship relation, and using these capabilities with available PSI schemes to determine common friends. In contrast, besides actually constructing and implementing a framework for secure discovery of common friends, this work shows that traditional PSI techniques, designed to work with low-entropy set items, are actually an "overkill." More efficient solutions, such as the one based on Bloom filters presented in Sec. 2.4, can be used to significantly reduce communication complexity and remove the need for a linear number of public-key operations. Also, we present the design of the *Common Friends* framework, which is intended to enable developers to integrate it in their application and use it, e.g., to support trust and access control decision based on social proximity. We verify practicality of proposed techniques with an experimental evaluation which shows the significant performance gains of using BFPSI over traditional PSI protocols designed for low-entropy items. We also integrate our *Common Friends* service into the tethering application sketched in [5], which supports sharing of tethering connections, and present a full-blown implementation.

Bloom filters have been used in the context of secure protocols in a number of other scenarios. For instance, privacy-preserving information matching based on *encrypted* Bloom filters has been proposed by Bellovin and Cheswick [9] for privacy-preserving database search. Kerschbaum [36] applies them for the protection of supply chain integrity and mitigate risks of industrial espionage. Also, Eppstein and Goodrich [20] propose Privacy-enhanced Invertible Bloom Filters for secure comparison of compressed DNA sequences. Clearly, none of these techniques apply Bloom filters to securely discover common friends and/or for efficient, privacy-preserving intersection of high-entropy items.

Finally, a few techniques [30, 22, 25, 34, 21] have improved performance of PSI by introducing assumptions such as the presence of trusted hardware tokens. These tokens might need to be trusted by both parties [30, 22, 25], by only one party [34], or even untrusted [21]. While efficient, these protocols require handing over the hardware token, and hence are inapplicable in scenarios like finding common friends between stranger devices.

7 Conclusion

This paper presented the *Common Friends* service, a framework supporting secure discovery of mutual friends, which protects privacy of non-common friends and guarantees authenticity of friendships. We first presented a generic construction that reduces the problem of finding friends to private set intersection, while ensuring authenticity of claimed friends via bearer capabilities. Next, we introduced a more efficient instantiation, based on Bloom filters, that only incurs a constant number of public-key cryptography operations. We also integrated *Common Friends* with an existing application for sharing Internet connection, whereby users decide whether or not to share based on the existence of common friends. A comprehensive experimental evaluation attested to the practicality of proposed techniques.

The protocols described in this paper allow user to detect whether another user is two hops away in a social graph. As part of future work, we plan to generalize them to detect friends who are more than two hops away. We also intend to extend the infrastructure proposed in this paper to detect other common attributes between two users, such as shared interests and group membership, and explore the use of social proximity to support additional access control decisions (e.g., for cab/ride sharing, routing, impromptu online dating, or multimedia content dissemination). Finally, whether or not we can design an efficient Bloom filter based PSI-CA variant for high-entropy items remains an open question.

Acknowledgements: We thank Elena Reshetova for contributing to the implementation of the original tethering app [5], Thomas Schneider for collaborating on the idea of using capabilities with PSI protocols, and Mark Manulis and Bertram Poettering for their fruitful collaboration. Marcin Nagy's work was supported by the SCAMPI project (grant number 258414) of the EU Seventh Framework Programme. The work done at University of Helsinki and Aalto University was partially supported, respectively, by a donation from Nokia Research Center and a Google Research Award.

References

[1] Bandwagon. http://bandwagon.io.

[2] Sidecar—My ride is your ride. http://side.cr/.

[3] Zoosk. http://zoosk.com.

[4] R. Agrawal, A. Evfimievski, and R. Srikant. Information sharing across private databases. In *ACM International Conference on Management of Data (SIGMOD)*, pages 86–97, 2003.

[5] N. Asokan, A. Dmitrienko, M. Nagy, E. Reshetova, A.-R. Sadeghi, T. Schneider, and S. Stelle. Crowdshare: Secure mobile resource sharing. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 432–440, 2013.

[6] D. Balfanz, G. Durfee, N. Shankar, D. K. Smetters, J. Staddon, and H.-C. Wong. Secret Handshakes from Pairing-Based Key Agreements. In *IEEE Symposium on Security and Privacy (S&P)*, pages 180–196, 2003.

[7] M. Bellare. New proofs for NMAC and HMAC: Security without collision-resistance. In *CRYPTO*, pages 602–619, 2006.

[8] M. Bellare, C. Namprempre, D. Pointcheval, and M. Semanko. The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme. *Journal of Cryptology*, 16(3):185–215, 2003.

[9] S. M. Bellovin and W. R. Cheswick. Privacy-Enhanced Searches Using Encrypted Bloom Filters. Technical Report CUCS-034-07, Columbia University and AT&T, 2004. https://mice.cs.columbia.edu/getTechreport.php?techreportID=483.

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

[11] J. Camenisch and G. Zaverucha. Private intersection of certified sets. In *Financial Cryptography and Data Security (FC)*, pages 108–127, 2009.

[12] H. Carter, C. Amrutkar, I. Dacosta, and P. Traynor. For your phone only: custom protocols for efficient secure function evaluation on mobile devices. *Security and Communication Networks*, 2013.

[13] G. Danezis and P. Mittal. Sybilinfer: Detecting Sybil Nodes using Social Networks. In *Network and Distributed System Security Symposium (NDSS)*, 2009.

[14] E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and Private Computation of Cardinality of Set Intersection and Union. In *International Conference on Cryptology and Network Security (CANS)*, pages 218–231, 2012.

[15] E. De Cristofaro, J. Kim, and G. Tsudik. Linear-Complexity Private Set Intersection Protocols Secure in Malicious Model. In *ASIACRYPT*, pages 213–231, 2010.

[16] E. De Cristofaro, M. Manulis, and B. Poettering. Private Discovery of Common Social Contacts. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 147–165, 2011.

[17] E. De Cristofaro and G. Tsudik. Practical Private Set Intersection Protocols with Linear Complexity. In *Financial Cryptography and Data Security (FC)*, pages 143–159, 2010.

[18] E. De Cristofaro and G. Tsudik. Experimenting with Fast Private Set Intersection. In *Trust and Trustworthy Computing (TRUST)*, pages 55–73, 2012.

[19] W. Dong, V. Dave, L. Qiu, and Y. Zhang. Secure friend discovery in mobile social networks. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1647–1655, 2011.

[20] D. Eppstein, M. T. Goodrich, and P. Baldi. Privacy-Enhanced Methods for Comparing Compressed DNA Sequences. http://arxiv.org/abs/1107.3593, 2011.

[21] M. Fischlin, B. Pinkas, A.-R. Sadeghi, T. Schneider, and I. Visconti. Secure Set Intersection with Untrusted Hardware Tokens. In *Cryptographers' Track at the RSA Conference (CT-RSA)*, 2011.

[22] M. Fort, F. C. Freiling, L. D. Penso, Z. Benenson, and D. Kesdogan. TrustedPals: Secure Multiparty Computation Implemented with Smart Cards. In *European Symposium on Research in Computer Security (ESORICS)*, pages 34–48, 2006.

[23] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient Private Matching and Set Intersection. In *EUROCRYPT*, pages 1–19, 2004.

[24] D. Hardt. The OAuth 2.0 authorization framework. RFC 6749, RFC Editor, 2012.

[25] C. Hazay and Y. Lindell. Constructions of truly practical secure protocols using standard smartcards. In *ACM Conference on Computer and Communications Security (CCS)*, pages 491–500, 2008.

[26] C. Hazay and Y. Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography Conference (TCC)*, pages 155–175, 2008.

[27] S. Hohenberger and S. Weis. Honest-Verifier Private Disjointness Testing Without Random Oracles. In *Privacy Enhancing Technologies Symposium (PETS)*, pages 277–294, 2006.

[28] Y. Huang, E. Chapman, and D. Evans. Privacy-preserving applications on smartphones. In *USENIX Workshop on Hot Topics in Security (HotSec)*, 2011.

[29] Y. Huang, D. Evans, and J. Katz. Private Set Intersection: Are Garbled Circuits Better than Custom Protocols? In *Network and Distributed Security Symposium (NDSS)*, 2012.

[30] A. Iliev and S. Smith. More Efficient Secure Function Evaluation Using Tiny Trusted Third Parties. Technical Report TR2005-551, Dartmouth College, 2005.

[31] I. Ioannidis, A. Grama, and M. Atallah. A Secure Protocol for Computing Dot-Products in Clustered and Distributed Environments. In *International Conference on Parallel Processing (ICPP)*, pages 379 –384, 2002.

[32] S. Jarecki and X. Liu. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *Theory of Cryptography Conference (TCC)*, pages 577–594, 2009.

[33] S. Jarecki and X. Liu. Fast Secure Computation of Set Intersection. In *International Conference on Security and Cryptography for Networks (SCN)*, pages 418–435, 2010.

[34] K. Järvinen, V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Embedded SFE: Offloading Server and Network Using Hardware Tokens. In *Financial Cryptography and Data Security (FC)*, pages 207–221, 2010.

[35] A. Johnson, P. Syverson, R. Dingledine, and N. Mathewson. Trust-based anonymous communication: Adversary models and routing algorithms. In *ACM Conference on Computer and Communications Security (CCS)*, pages 175–186, 2011.

[36] F. Kerschbaum. Public-key encrypted bloom filters with applications to supply chain integrity. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 60–75, 2011.

[37] L. Kissner and D. X. Song. Privacy-Preserving Set Operations. In *CRYPTO*, pages 241–257, 2005.

[38] K. Kostiainen, E. Reshetova, J.-E. Ekberg, and N. Asokan. Old, new, borrowed, blue – a perspective on the evolution of mobile platform security architectures. In *ACM Conference on Data and Application Security and Privacy (CODASPY)*, pages 13–24, 2011.

[39] M. Li, N. Cao, S. Yu, and W. Lou. FindU: Privacy-preserving personal profile matching in mobile social networks. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 2435–2443, 2011.

[40] M. Manulis, B. Pinkas, and B. Poettering. Privacy-Preserving Group Discovery with Linear Complexity. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 420–437, 2010.

[41] P. Mittal, M. Wright, and N. Borisov. Pisces: Anonymous Communication Using Social Networks. In *Network and Distributed System Security Symposium (NDSS)*, 2013.

[42] A. Mohaisen, H. Tran, A. Chandra, and Y. Kim. Trustworthy distributed computing on social networks. In *ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, pages 155–160, 2013.

[43] M. Nagy, N. Asokan, and J. Ott. `PeerShare`: A System Secure Distribution of Sensitive Data Among Social Contacts. In *Nordic Conference on Secure IT Systems (NordSec)*, 2013. Extended version available at http://arxiv.org/abs/1307.4046.

[44] NIST. http://www.nsa.gov/ia/_files/nist-routines.pdf.

[45] G. Norcie, E. De Cristofaro, and V. Bellotti. Bootstrapping Trust in Online Dating: Social Verification of Online Dating Profiles. In *Financial Cryptography and Data Security Workshop on Usable Security (USEC)*, 2013.

[46] E. Rantala, A. Karppanen, S. Granlund, and P. Sarolahti. Modeling energy efficiency in wireless Internet communication. In *ACM Workshop on Networking, Systems, and Applications for Mobile Handhelds (MobiHeld)*, pages 67–68, 2009.

[47] D. Recordon and D. Reed. OpenID 2.0: a platform for user-centric identity management. In *ACM Workshop on Digital Identity Management*, pages 11–16, 2006.

[48] A. S. Tanenbaum et al. Using Sparse Capabilities in a Distributed Operating System. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 558–563, 1986.

[49] N. Thiagarajan, G. Aggarwal, A. Nicoara, D. Boneh, and J. P. Singh. Who killed my battery?: Analyzing mobile browser energy consumption. In *International Conference on World Wide Web (WWW)*, pages 41–50, 2012.

[50] M. Von Arb, M. Bader, M. Kuhn, and R. Wattenhofer. VENETA: Serverless friend-of-friend detection in mobile social networking. In *IEEE Conference on Wireless and Mobile Computing (WiMob)*, pages 184–189, 2008.

[51] A. C. Yao. How to Generate and Exchange Secrets. In *Foundations of Computer Science (FOCS)*, pages 162–167, 1986.

[52] L. Zhang and X.-Y. Li. Message in a Sealed Bottle: Privacy Preserving Friending in Social Networks. http://arxiv.org/abs/1207.7199, 2012.

[53] R. Zhang, Y. Zhang, J. Sun, and G. Yan. Fine-grained private matching for proximity-based mobile social networking. In *IEEE Conference on Computer Communications (INFOCOM)*, pages 1969–1977, 2012.