Technische Universität Darmstadt

Center for Advanced Security Research Darmstadt

Technical Report TUD-CS-2013-0084

CrowdShare: Secure Mobile Resource Sharing

N. Asokan, Alexandra Dmitrienko, Marcin Nagy,
Elena Reshetova, Ahmad-Reza Sadeghi,
Thomas Schneider, Stanislaus Stelle

System Security Lab
Technische Universität Darmstadt, Germany

CASED
Technische Universität Darmstadt
D-64293 Darmstadt, Germany

TUD-CS-2013-0084
First Revision: April 30, 2013
Last Update: May 17, 2013

CrowdShare: Secure Mobile Resource Sharing

N. Asokan[1], Alexandra Dmitrienko[2], Marcin Nagy[5], Elena Reshetova[4], Ahmad-Reza Sadeghi[2,3], Thomas Schneider[3], Stanislaus Stelle[3]

[1] University of Helsinki, Finland
{asokan}@acm.org
[2] Fraunhofer-Institut SIT Darmstadt, Germany
{alexandra.dmitrienko, ahmad-reza.sadeghi}@sit.fraunhofer.de
[3] Technische Universität Darmstadt, Germany
{thomas.schneider, stanislaus.stelle}@cased.de
[4] Intel, OTC, Finland
{elena.reshetova}@gmail.com
[5] Aalto University, Finland
{marcin.nagy}@gmail.com

Abstract. Mobile smart devices and services have become an integral part of our daily life. In this context there are many compelling scenarios for mobile device users to share resources. A popular example is tethering. However, resource sharing also raises privacy and security concerns. In this paper, we present `CrowdShare`, a complete framework and its (Android) implementation for secure and private resource sharing among nearby devices. `CrowdShare` provides pseudonymity for users, accountability of resource usage, and the possibility of specifying access control in terms of social network relationships. The latter, denoted `FoF Finder` service, allows two devices, to determine in a privacy-preserving manner whether their owners are friends of each other or have common friends. `FoF Finder` service is an independent software module that encapsulates secure usage of privacy-preserving set intersection protocols. It is designed to be used easily by developers to secure other applications. Further, `CrowdShare` preserves secure connectivity between nearby devices even in the absence of the mobile infrastructure. We have implemented `CrowdShare` including `FoF Finder` service on Android devices and report good performance results and preliminary user evaluations.

1 Introduction

The popularity of inexpensive communication services like Skype [4], Gtalk [1], and WhatsApp [5] is increasing rapidly. They allow people to communicate with almost the same ease as with phone calls and Short Message Service (SMS) messages, but at a significantly lower cost to the users. However, the pre-requisite to all such services is Internet access, which can be quite difficult to obtain in certain situations. First, Internet access can be expensive while traveling abroad. As a result, *tethering*, the process of sharing Internet connectivity from one device by turning it into a wireless access point that other devices can connect

to, has gained popularity. Some devices provide tethering as part of their base functionality, while other third party applications like JoikuSpot [17] and OpenGarden [2] can enable tethering. Second, in some situations Internet connectivity may be impossible like in the aftermath of a disaster or while visiting rural areas with little network coverage or when organizing demonstrations against totalitarian regimes. In such situations, ad-hoc mesh networks among mobile devices can provide similar communication or data exchange services. For example, the Serval [3] project aims to preserve connectivity between mobile devices by providing MeshSMS and Call services even in the absence of the mobile support infrastructure; Nokia Instant Community [19] allows mobile devices to form an ad-hoc network to exchange messages or share content.

Naturally any such service that allows the resources of some users (providers) to be used by other users (consumers) has to identify potential security and privacy threats and provide solutions to address them. In particular, providers need to have convenient means to specify suitable *access control*. Access control may be specified in terms of membership in a service (as is done by the community-based WiFi sharing service Fon [10]). Another natural basis to specify access control is to share Internet connectivity with "friends of friends", e.g., with visitors of an organization or guests at a party. Consumers need some level of privacy which has to be balanced against the providers' need for accountability so that providers would have evidence of resource usage by consumers.

Our goal and contribution. In this paper we present `CrowdShare`: a service design and its (Android) implementation that allows users to *share connectivity*. `CrowdShare` distinguishes itself from other tethering and mesh networking applications through incorporating a security architecture with privacy-preserving access control based on social relationships, pseudonymity for users, and accountability of usage.

Access control based on social relationships is built on the `CrowdShare` *privacy-preserving friends-of-friends finder* (`FoF Finder`) service. It allows two devices to verify if their owners are direct friends or share common friends in a social network.`FoF Finder` service combines *private set intersection* (PSI) protocols with social network interfaces to ensure authenticity (a user cannot falsely claim to be a friend of someone) and privacy (users only find information about common friends).

Although `CrowdShare` focuses on connectivity sharing, the architecture is generic and can be applied to resource sharing in general. `FoF Finder` service is of independent interest. It encapsulates the *secure* use of PSI for access control in a software module and provides a clear interface so that app developers can easily integrate `FoF Finder` into their applications. It also allows new PSI protocols to be plugged into the `FoF Finder` service easily.

In summary our contributions are 1) *design and integrated implementation* of a complete generic framework for secure resource sharing among nearby devices by incorporating a *security architecture* into existing technologies for mesh networking, tethering, and social network interfaces. and 2) *reusable components*,

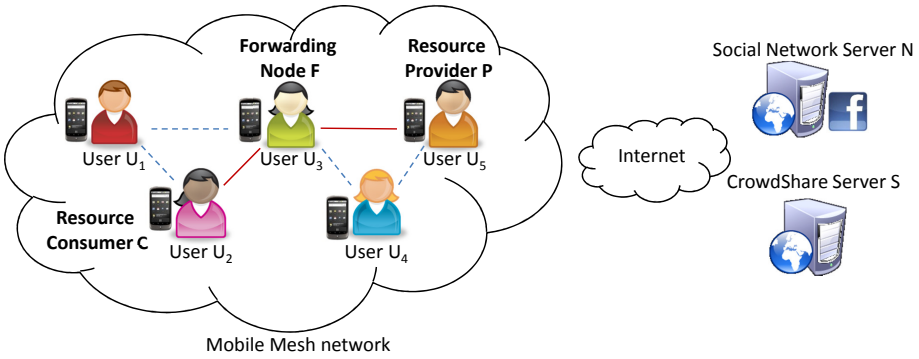

Fig. 1: `CrowdShare` system model

especially for the privacy-preserving `FoF Finder` service, that can be integrated securely and easily by app developers.

Outline. We describe our system model and requirement analysis in §2, and present a mobile platform architecture in §3. §4 describes the system protocols and services, while §5 includes an informal security analysis. In §6 we present our implementation choices. §7 includes performance evaluation, and §8 provides details of the usability study. §9 describes related work. §10 concludes the paper.

2 System Model and Requirement Analysis

System Model. The system model of the `CrowdShare` system is depicted in Fig. 1. It consists of a trusted `CrowdShare` server S, a social network server N, and a set of users $\mathcal{U}$. S admits the users to join the `CrowdShare` service, while N provides information about friend relationships among users. Each user $\mathsf{U}_i \in \mathcal{U}$ possesses a mobile platform which runs the `CrowdShare` application and enables communication of different users via the mesh network. A user U_i can play one of the following roles: (i) resource provider P, (ii) resource consumer C, or (iii) forwarding node F. P has access to (a set of) resources $\mathcal{R}$ and shares access to them with other users (e.g., Internet bandwidth, media files, or location information). P can restrict the access to his resources either to any U_i, or to a subset of users $\mathcal{F} \subset \mathcal{U}$, who are in a social relation with P in the social network (e.g., friends or friends of friends). C does not have direct access to $\mathcal{R}$ or $\mathcal{R}$ might be available but expensive, hence it consumes resources provided by P via the mesh network. Forwarding nodes F forward messages in the mesh network such that P and C can be connected over multiple hops.

Threat Model and Security Requirements. `CrowdShare` and its infrastructure could be subject to several attacks. Our threat model does not cover any attacks against the operating system of the mobile device or any outside component, e.g., a remote server. Instead we concentrate on the attacks that users

perform against the service itself. We focus on protecting against semi-honest adversaries that modify the `CrowdShare` service in order to learn sensitive information or get unauthorized access to services. We identify the following threats for `CrowdShare` which motivate the need for the respective security requirements.

1. **Man-in-the-middle Attacks ⇒ Channel Protection.** Devices in the ad-hoc mesh network should not be able to act as man-in-the middle that eavesdrops on or modifies messages that are routed through them. This motivates for channel protection.
2. **Framing Attacks ⇒ Accountability.** C could use P's resources for illegal purposes. For instance, in the case of Internet sharing C could download a pirated song, leading the copyright owner of the song to accuse P of unauthorized use. In case of such violations, P needs the ability to give evidence that the resource was requested by a particular C. This motivates accountability.
3. **User Identification ⇒ Pseudonymity.** It should not be possible for a user to learn personally identifiable information such as the phone number or the email address of another user. This motivates pseudonymity.
4. **Unauthorized Usage ⇒ Access Control.** C should not be able to use P's resources without its consent. This motivates access control, i.e., P can attach a policy to the shared resource that needs to be fulfilled by C.

To satisfy the requirement of access control, we develop a friend-of-friends service (`FoF Finder` service) which utilizes information from social networks to identify friend relations of users and allows users to share resources only with friends (and friends of friends). Potential threats against the `FoF Finder` service lead to corresponding security requirements:

1. **Disclosure of Friends ⇒ Privacy.** The list of friends in a social network reveals sensitive information about the subject. Therefore, `FoF Finder` service should not require users to disclose their friend lists to each other.
2. **Fake-Friends ⇒ Authenticity.** If the parties can lie about their list of friends, they can easily extend them with entries that are likely to be a friend of the other party. To avoid this, the list of friends should be authentic.
3. **Tracing ⇒ Decentralization.** No central entity in the system should be able to determine which consumer used which provider's services. Therefore, `FoF Finder` service should be decentralized.

3 `CrowdShare` Platform Architecture

The platform architecture for P and C is depicted in Fig. 2. Both, P and C, can also take a role of relaying node F, when necessary.

The user interface UI enables the user to configure the `CrowdShare` application and to trigger the user registration process with the server S, which is handled by the RegService component. Registration is performed only once, after the `CrowdShare` is installed. After the successful registration, MeshNet component is configured enabling the platform to join, leave, and operate in the `CrowdShare` mesh network.

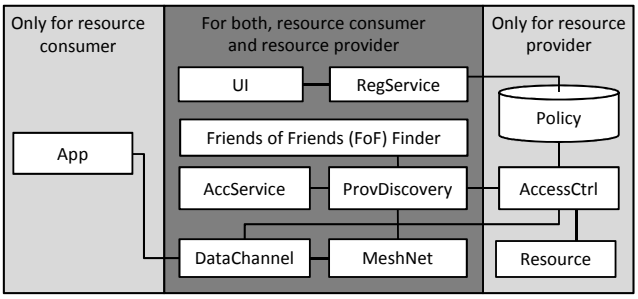

Fig. 2: CrowdShare Platform Architecture

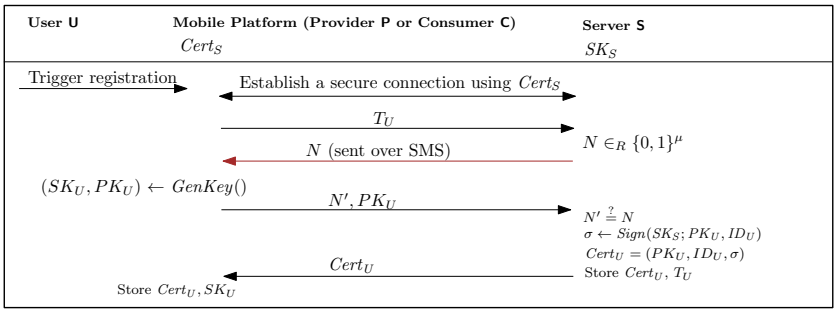

Fig. 3: Registration protocol

ProvDiscovery component is responsible for the discovery of the suitable resource provider P. The FoFService component is used to discover social relationships between P and C. AccService handles the accountability service which prohibits framing attacks against P. Finally, DataChannel serves for the delivery of the resource R from P to C.

The components specific to resource providers are: Resource, Policy, and AccessCtrl. Resource is a resource R which P can share. The user can define an access control policy for Resource by configuring Policy via the UI component. The specified policy is enforced by an AccessCtrl component. Resource is consumed by the application App running on C (e.g., a web-browser) via the channel handled by DataChannel component.

4 CrowdShare Protocols and Services

4.1 CrowdShare Protocols

Registration. The purpose of registration is to figure out a real user identity and to issue pseudonymous certificates which will be used by the CrowdShare community members for subsequent communication.

The registration protocol is depicted in Fig. 3. First, the user U invokes the RegService component of C or P and establishes a secure channel to the server S using the certificate $Cert_S$ of S that is provided together with the `CrowdShare` application.

The `CrowdShare` application sends the user's phone number T_U to S, who generates a one time password (OTP) N and sends it over the short message service (SMS) back to the platform (P or C). In turn, the `CrowdShare` application generates an asymmetric key pair (SK_U,PK_U), and sends PK_U to S together with $N' = N$. Next, S verifies if the received N' matches N sent over SMS, generates a user certificate $Cert_U$ for this user, stores $Cert_U$ together with T_U and returns $Cert_U$ to U. The received $Cert_U$ is stored together with SK_U for future use.

The user's identity is verified, because S has the assurance that the submitted phone number belongs to the user, as he was able to receive the OTP N. S keeps the mapping between certificates and phone numbers secret and reveals it only to authorized entities, e.g., in case of a subpoena.

Provider Discovery. The goal of the provider discovery protocol is to discover a suitable resource provider P which can share its resources with the resource consumer C. The corresponding protocol is shown in Fig. 4. It is initiated when the resource request cannot be served locally (e.g., the request of the web-browser for the network connectivity cannot be served due to unavailable network connection).

First, $\mathsf{C} \in \mathcal{U}$ connects to a potential resource provider $\mathsf{P} \in \mathcal{U}$ (using mesh networking services) and establishes a secure (i.e., authentic and confidential) channel to it based on $Cert_C$ and $Cert_P$ (i.e., certificates obtained during registration). Next, C sends the resource request over the established channel along with the description of the resource R. If R is available, P responds with *policy* which specifies conditions for resource sharing. Particularly, *policy* may allow resource sharing with friends (or friends of friends) only, or require execution of the accountability protocol in order to protect P from framing attacks. If required by *policy*, P and C additionally use *Friend-of-Friend Finder* service (cf. §4.2) to identify friend relationships and execute the *accountability protocol*. If all conditions are met, P is added to a set of suitable provider candidates $\mathcal{P}$.

The protocol repeats n times to populate $\mathcal{P}$ with n candidates, where n is a configurable system parameter. The best candidate $\mathsf{P}^* \in \mathcal{P}$ is selected for resource sharing, while others are kept as back-ups. The availability of every $\mathsf{P} \in \mathcal{P}$ is monitored through listening to heart beat messages transmitted on a regular base. If any of them disappear, a new round of the provider discovery protocol is triggered to find a new candidate.

Accountability. Accountability is achieved by having C sign a resource quota request RQR that contains PK_C, the type of the resource R, and the resource leasing time τ. The resulting signature σ_{RQR} is sent to P^*, verified, and stored as an evidence.

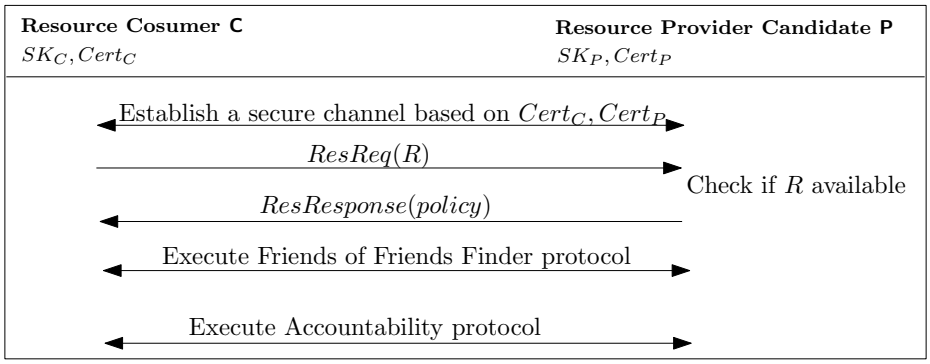

Fig. 4: Provider discovery protocol

Data Channel. A data channel is used for the delivery of the resource R from P^* to C. To provide confidentiality and authenticity to the data channel, we use standard techniques for setting up virtual private network (VPN) connections. Depending on the type of the shared resource, the VPN connection is either between C and P^* or C and S (e.g., between C and S for Internet connectivity sharing).

4.2 Friends-of-Friends Finder (`FoF Finder`) Service

We consider two approaches for designing a `FoF Finder` service based on friend relationships in existing social networks.

Server-aided approach. During registration, each user authorizes S to access his friend list from the social network server N and to map the social network identifiers of the user's friends (and friends of friends) to their `CrowdShare` membership certificates. This mapping is sent to the registering device. During provider discovery, P checks if the certificate of C belongs to one of his friends or friends of friends by comparing the certificate identifier of C with identifiers in the friends database.

The server-aided solution requires each user to learn about entities in his social graph at hop lengths > 1, e.g., the number of friends of friends he has or their certificates which serve as pseudonyms. Depending on how users have set the visibility of their friend relations in the social network, this may be information that was otherwise not available to users.

`FoF Finder` service based on Private Set Intersection. We propose an alternative solution for Friend-of-Friend Finder which minimizes the information about the social graph leaked to a user. It makes use of the Private Set Intersection (PSI) protocols [16, 9] to determine if P and C are direct friends or have friends in common. There are two variants of PSI: The vanilla PSI protocol

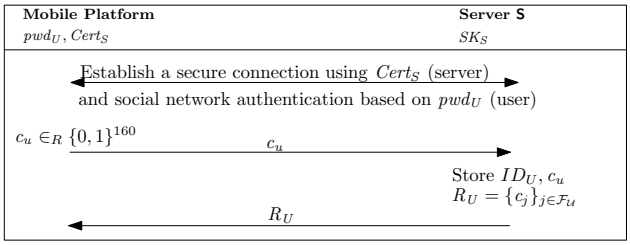

Fig. 5: Capability upload/download

[16] outputs the intersection of the two sets and does not reveal any additional information about the remaining elements of either input set. The PSI Cardinality (PSI-CA) protocol [9] outputs only the cardinality of the intersection and reveals no other information about any element of either set. Both protocols are secure against semi-honest (honest, but curious) adversaries, allow inputs of arbitrary size and have similar performance. To further minimize the amount of leaked information (e.g., by revealing only one bit that denotes whether the intersection is non-empty) would require more flexible but also more complex variants of PSI protocols, cf. [15, 14, 7].

A straightforward approach would be to have each party use the list of social network user identifiers of its friends as its input to PSI. However, because user identifiers in social networks are not secret, such an approach is vulnerable to the "fake friends" threat we referred to in §2, as a malicious user could put arbitrary identifiers in his friend list. Further, PSI protocols themselves do not provide authenticity, hence they are vulnerable to man-in-the-middle attacks.

To address the former problem, we use so-called sparse capabilities [21] (sometimes also known as "bearer tokens") instead of social network user identifiers. Each user U distributes a time-limited capability to its friends via a secure (i.e., authentic and confidential) channel so that possession of the capability from U represents a proof of the friendship relationship with U.

We use the server S for capability distribution as depicted in Fig. 5. First, `CrowdShare` and 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 during channel establishment. Next, the `CrowdShare` generates a random capability c_u and uploads it to S via the established channel. The server stores c_u together with the user identifier ID_U and returns a list R_U of capabilities of user's friends. This protocol runs periodically in order to keep R_U updated.

The downloaded capability list R_U uniquely identifies the user's friends and can be used as an input to the PSI algorithm. On the other hand, they are distributed over the confidential and authentic channels, and, hence, resistant to "fake friends" attacks.

To address the authenticity issue of PSI protocols mentioned before, we bind a PSI inputs to the certificates of the parties running it. Particularly, the `FoF Finder` protocol runs over the secure (i.e., authentic and confidential) channel

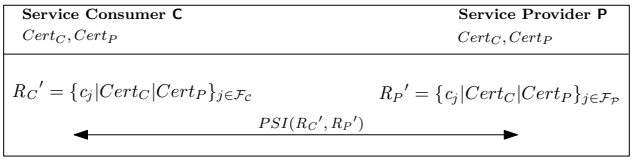

Fig. 6: `FoF Finder` protocol: Binding PSI inputs to certificates

established based on $Cert_P, Cert_C$ (as was shown in Fig. 4). The friend's capability lists are bound to this channel by appending $Cert_C$ and $Cert_P$ to each capability c_j in the list, and using these lists $R_C{}'$ and $R_P{}'$ as inputs to the PSI/PSI-CA protocol, cf. Fig. 6. The performed transformations on capability lists has negligible impact on performance, as PSI/PSI-CA protocols hash each element in the list before further processing.

`FoF Finder` can identify direct friends as well by using the set containing only its own capability as input to PSI/PSI-CA. Further, it can be easily generalized to find contacts which are more than two hops away in one's social graph. However, this generalization may leak information about the social graph that was not already available to the users.

5 Security Considerations

In the following we provide an informal security analysis that demonstrates that the security requirements of §2 are fulfilled.

5.1 CrowdShare Services

Channel protection. For channel protection, all protocols are executed over a secure (i.e., confidential and mutually authenticated) channel. Particularly, the registration protocol runs over a channel where the server S is authenticated based on the server certificate $Cert_S$, while the user is authenticated by verifying user's phone number. Capability upload/download process is secured by means of a secure connection established based on a server certificate and password-based authentication against the social network server. The provider discovery, `FoF Finder`, and accountability protocols run over the secure channel established based on mutually exchanged certificates between P and C. The resource delivery is protected by a data channel established between C and P or C and S. The former is used in use cases which are not sensitive to eavesdropping by P, e.g., in case of file sharing (a file originating from P is already known to P). The latter is applied in case if P is a subject for confidentiality requirement, e.g., when sharing Internet connectivity (to ensure P cannot eavesdrop or manipulate traffic downloaded by C).

Pseudonymity. Pseudonymity is fulfilled by deploying pseudonymous user certificates which do not include any user specific information. The only entity which

can map certificates to user identities is the server S, which is trusted to keep this information confidential.

Accountability. Accountability is satisfied by deploying an accountability service which protects P from framing attacks. The signed resource quota request submitted during the accountability protocol can be used by P as evidence toward possible misuse by C. Further, the signature can be mapped to a user identity with the help of the server S, which keeps the mapping between pseudonymous user certificates and user phone numbers. Hence, the real user identity can be traced back in case of illegal usage (e.g., accessing illegal content).

Access Control. We use two access control mechanisms: (i) membership-based access control which allows users to deny non-members access to resources of members, and (ii) `FoF Finder` service which allows users to restrict access to resources based on their social relationships.

5.2 FoF Finder

Server-aided ***`FoF Finder`*** *Service.* Connection to S is not needed during the resource provider discovery phase, but only during the registration phase. Thus S does not learn which C connects to which P (cf. decentralization requirement in §2). As only P checks whether the C certificate is in his list of friends (and friends of friends) and this list is received over the secure connection from S, the authenticity and privacy requirement of §2 are also fulfilled.

Privacy Preserving ***`FoF Finder`*** *Service.* The solution is private as the only interaction between C and the P is the run of the PSI or PSI-CA protocol which by definition does not reveal any information on the inputs. It provides authenticity because if the server behaves correctly, it is infeasible for an attacker to acquire the capability needed to fake a friend relationship. It is decentralized as the server is involved only in the beginning, but not during the discovery phase, so it does not learn interactions between users. Hence the solution meets all requirements for friend-of-friend access control in §2.

6 Implementation

In this section we describe the implementation of the trusted server S and the mobile device which integrates functionality of the resource consumer C, resource provider P, and a relay device F in one. Further, we describe the implementation of a simple tethering app which makes use of the `FoF Finder` service and allows the user to configure access control to tethering functionality based on social relationships.

Server. The server S provides the following main functionalities: (i) registration of new `CrowdShare` community members, (ii) server-side components of `FoF Finder` service described in §4.2, and (iii) a database that includes persistent information from other services (e.g., mapping from user identities to certificates). The functionality of S is implemented in Java 1.5 and stores objects in MySQL[6] using the Hibernate framework[7]. The implementation has 5 188 lines of Java code (LoC).

Mobile Platform. Our implementation targets Android-based devices. We used Google's Nexus One and the HTC Desire smartphones with Cyanogenmod 7.0 images for our development. The code is written in Java (for the API level 10) and makes use of a Bouncy Castle crypto library v. 147 (written in Java). Stock Android includes an older version of this library, which, however, is not complete (e.g., does not support certification request generation), and, hence, we included a newer library version which provides all the required functionality. For the implementation of cryptographic primitives, we used RSA 1024 and AES 128. We used standard SSL for the establishment of the secure channel used in provider discovery and OpenVPN (from the Cyanogenmod image) for the protection of the data channel. Our Android app has a modular design. Particularly, MeshNet and FoFService are implemented as separate components with well-defined interfaces which can be replaced when necessary or re-used in other applications. The overall implementation excluding the MeshNet component has 9 441 LoC.

We adapted the implementation of the Serval open source project [11] for the instantiation of the MeshNet component. Serval allows mobile devices to establish mesh networking on top of ad-hoc WiFi connections. It integrates BATMAN [18], a proactive distance vector routing protocol for wireless mesh networks. Further, it supports voice calls and text messages between mesh modes (hence, this functionality is also inherited by our implementation), but it does *not* provide Internet connectivity sharing and does *not* address possible security threats, which are our main focus.

Generally, stock Android devices cannot be configured to operate in WiFi ad-hoc mode without root access. Root access is required for loading a WiFi driver, configuring it to operate in ad-hoc mode and for configuring IP settings. However, root access is not required for the usage of our `FoF Finder` service. To support this claim, we implemented a simple (one-hop) tethering app which uses `FoF Finder` service for access control. The app uses Bluetooth to run `FoF Finder` protocols and WiFi for tethering and does not require root privileges.

`FoF Finder` Service. `FoF Finder` implemented as a simple Android service that exposes its interfaces via Android Interface Definition Language (AIDL)

[6] http://www.mysql.com
[7] http://www.hibernate.org

declarations. `FoF Finder` service is used by an application as follows: the application instances on an initiator device $\mathcal{I}$ and a responder device $\mathcal{R}$ first establish a communication channel and initiate the `FoF Finder` Service protocol engines on each side indicating the *type* of protocol variant to be used. The *type* indicates (1) use of *PSI* vs. *PSI-CA* and (2) the protocol output: finding direct friends vs. finding common friends-of-friends). Thereafter, the applications act as the transport channel for passing the protocol messages between the two protocol engines. The protocol messages are containers implemented as *Parcelable* and *Serializable* Android classes. They are opaque to the calling applications. The protocol engines first create and exchange ephemeral Diffie-Hellman public keys, append the keys to each element in the respective capability set (see §4.2), and follow the chosen PSI protocol variant. They also compute the Diffie-Hellman shared key K_{IR}. At the end of the protocol run, the application on each device receives K_{IR}. In addition, the application at $\mathcal{I}$ receives the result of PSI. Application developers can easily start embedding `FoF Finder` functionality into their code by adding the `FoF Finder` service **.aidl** interface declaration to their application source tree together with the container classes. For more details, see Appendix A.

WiFi Tethering App. We implemented a WiFi Tethering App that allows a device P to turn itself into a WiFi access point and enforce access control based on invoking `FoF Finder` service (as initiator $\mathcal{I}$) run over Bluetooth. It also allows a device C to scan the neighborhood for provider devices, connect to them using Bluetooth, and invoking `FoF Finder` service (as responder $\mathcal{R}$). If `FoF Finder` service determines the existence of common friends, P will transfer the access parameters (SSID and the password for the WiFi access point) over the secure channel resulting from `FoF Finder`. We use Java reflection to discover and use the standard Android API from the *android.net.wifi* package to set up, configure and use WiFi access points.[8]

7 Performance Evaluation

For our performance tests we used a HTC Desire device as a resource provider P and Nexus One devices for the resource consumer C and the relaying node F.

Multihop re-transmissions. Fig. 7 illustrates the performance with and without multihop re-transmissions. To perform this test, we sent a ping packet to a remote server (*www.google.de*) and estimated the delay of the received response. The test was done for the direct Internet connection (i.e., no mesh re-transmissions are required), as well as for 1 hop and 2 hop indirect connections[9]. We sent 200 ping packets for each case. The delay increases with rising hop counts in the multi hop connection, which is reasonable, as each additional hop imposes

[8] http://stackoverflow.com/a/7049074/1233435
[9] Our tests were limited by the number of available devices.

additional packet delay due to re-transmissions. Further, the context switch between 3G and WiFi transmissions also adds overhead. The several peaks for the 2 hop tethering are imposed by packet loss and subsequent re-transmissions required to perform packet delivery successfully. To summarize, a hop count of 2 introduces a little delay in the range of milliseconds, which is acceptable.

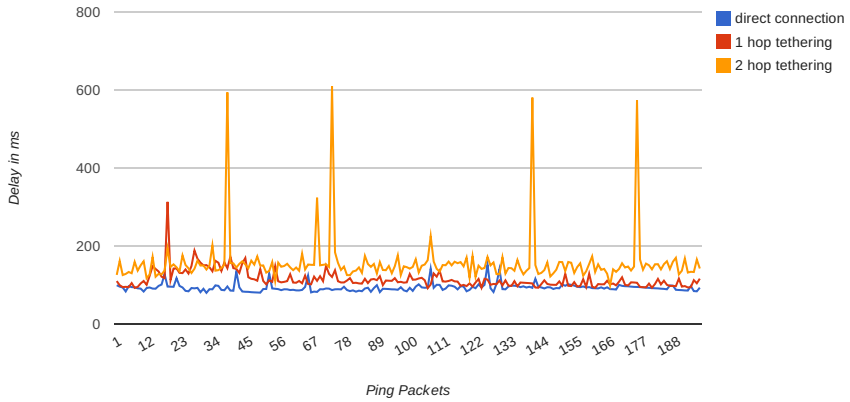

Fig. 7: Performance with and without multihop re-transmissions

Connection Establishment. We evaluated the time needed for the connection establishment with different configuration settings: without `FoF Finder`, with server-aided `FoF Finder` and with `FoF Finder` based on PSI, cf. Fig. 8.

The lower curve shows measurements for the connection establishment without `FoF Finder`. The performance for the server-aided `FoF Finder` is nearly the same as without `FoF Finder`, hence we didn't include it into the plot. The other curves demonstrate connection establishment time with FoF PSI with different input sizes (from 50 to 400 elements). The more entries are used as input for the FoF PSI algorithm, the longer it takes to calculate the set intersection, delaying overall time for the connection establishment.

To summarize, the connection time without FoF service takes 4.8 s in average. Server-aided FoF Finder adds negligible overhead, while FoF with PSI adds more significant delay up to 13 s for the input size of 400 elements.[10]

We find our performance results promising, as our implementation demonstrates feasibility of private set intersection algorithms even for mobile devices. Further, the input size for regular users are in practice smaller [22]. FoF based on PSI with the realistic input size of 100 elements adds overhead of only 1.7 s resulting in 6.5 s of overall time for the connection establishment.

[10] We note that our PSI protocol implementation is substantially faster than the garbled circuit-based PSI protocol of [15] which takes 598 s for 256 elements.

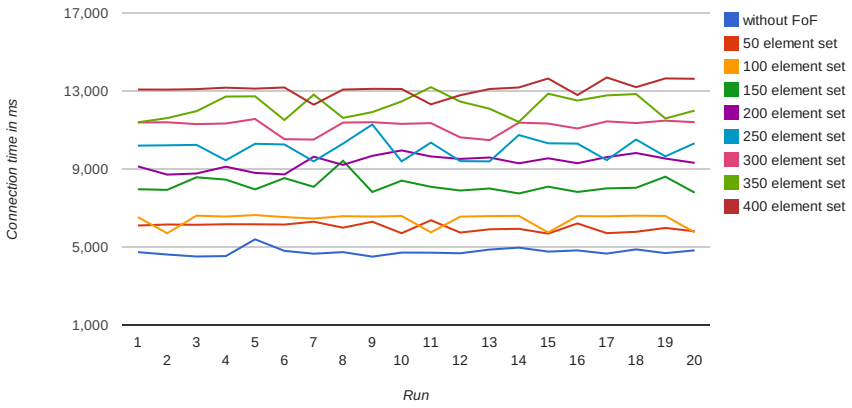

Fig. 8: Timings for the connection establishment

8 Usability Study

We conducted a preliminary user study in order to identify if any major usability issues can be uncovered. The study was conducted with 12 participants divided into three groups (e.g., four participants in each group). Participants are 11 male and 1 female, all between 25 and 29 years old. They all have technical background and feel comfortable with new technology and gadgets. Three of them had experience with Android-based smartphones. While this is not a representative sample, we deemed it sufficient given our goal of detecting major usability issues. In each group, the participants were handed test devices (three Nexus One devices and one HTC Desire) with pre-installed `CrowdShare` apps and were asked to register at Facebook and to establish the following friend relationships: If participants are referred to as A, B, C and D, then A is a direct friend with B and B is a direct friend of C and D. Participants were asked to perform four tasks. After completing each task, they answered a set of questions about the task. The possible answer options were scores from 0 (negative) to 5 (positive). At the end, there was a free form interview where the participants had the opportunity explain their responses.

Task 1. Registration. First participants were asked to register at `CrowdShare` server. Some questioned the need for the registration, but others accepted it as many apps on the market require registration. The corresponding menu was found to be easy to find and registration was performed efficiently. Results are shown in Fig. 9(a).

Task 2. Setting up a one-hop tethering connection. Participants were located in one office and were asked to establish one-hop tethering connection (particularly, A $\Leftrightarrow$ B, A $\Leftrightarrow$ C and A $\Leftrightarrow$ D). They tried out different connection settings: with and without FoFService and accountability services. The delays in finding the service provider and establishing the connection were thought to be bearable,

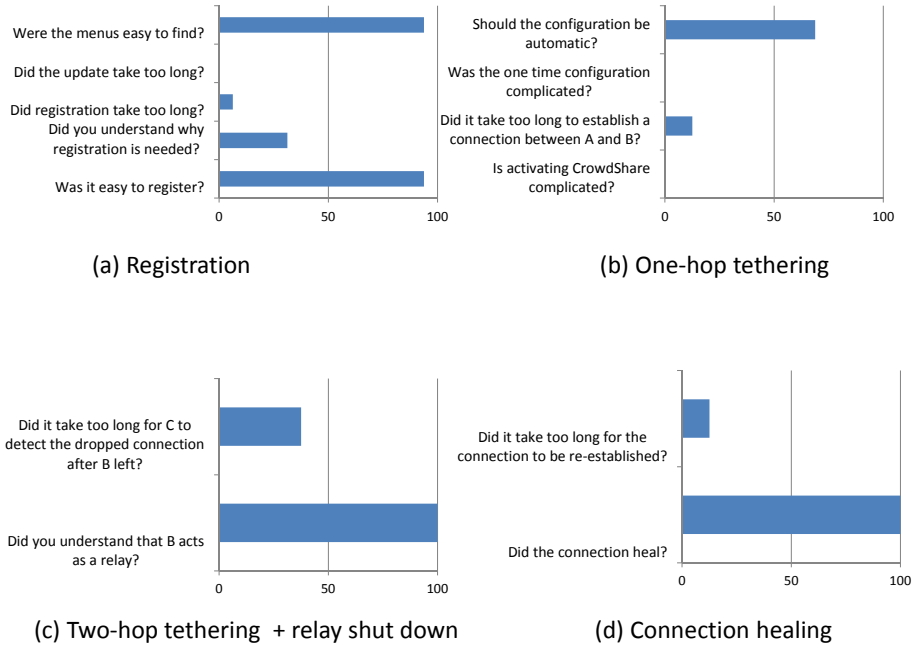

Fig. 9: Usability study results (Scores are normalized to 0% - 100%)

but a majority of the participants would prefer reasonable default options which would simplify configuration process. The results are shown in Fig. 9(b).

Task 3. Setting up a 2-hop tethering connection and shutting down the relay node. The test was performed in the office environment. Participants A, B and C were asked to establish 2-hop connection A $\Leftrightarrow$ B $\Leftrightarrow$ C. The total distance between A and C was about 100 meters, and they had doors and walls in between. Further, the B was asked to switch off his device. Results are shown in Fig. 9(c).

All participants understood the role of B as a relay. Some users found out that it took too long for C's device to realize that the connection was gone.

Task 4. Connection healing. To heal the interrupted 2-hop connection, participant D was placed in the position of B. This led to the establishment of the two-hop connection A $\Leftrightarrow$ D $\Leftrightarrow$ C. All the participants understood connection healing and found the self-healing process to be fast enough. Results are shown in Fig. 9(d).

In summary, we concluded that `FoF Finder` Service needs to have reasonable defaults. No other major usability issues were identified. After refining the design, we intend to carry out a larger usability study.

9 Related work

VENETA [6] is a mobile social networking platform that, among other features, allows decentralized SMS-messaging via Bluetooth (up to 3 hops) and privacy-preserving matching of common entries in the users' address books using private set intersection. To cope with the threat of the "fake friends" attack they suggest to limit the size of the input sets to 300. In contrast, we provide a more flexible solution to this attack. The combination of privacy-preserving profile matching and establishment of a secure channel was considered recently in [23]. Their solution allows a user to establish a shared key with another user only if their profiles match in a pre-determined set of attributes. Privacy-preserving discovery of common social contacts was considered in [8], where friends issue mutual certificates for their friendship relation.

A number of projects developed ad hoc communication and resource sharing on top of mesh networks like Serval [3, 11] and OpenGarden [2]. SCAMPI [20] provides generic discovery and a routing framework for opportunistic networks for developing versatile applications and services on top of it. Ad hoc communication has also found use cases in extreme situations where normal infrastructures are inaccessible, e.g., mines [12] and disaster-recovery scenarios [13]. Our work is different as we emphasize privacy and security, and allow privacy-preserving access control based on social relations.

10 Conclusion and Future Work

We present `CrowdShare`, a generic service design and its Android implementation that allows users to *share connectivity* with the nearby devices through WiFi channels. Compared to the existing application on tethering and mesh networking, `CrowdShare` incorporates a security architecture that provides privacy-preserving access control based on social relationships, pseudonymity for users, and accountability of usage. Access control based on social relationships is built on the `CrowdShare` *privacy-preserving friends-of-friends finder* (`FoF Finder`) service which is based on private two-party set intersection (PSI) in semi-honest adversary model, and of independent interest. It allows two devices to verify if their owners are direct friends or share common friends in a social network and provides them with a shared key to use for subsequent access control based on the verified friend relationship. `FoF Finder` provides a clear interface such that app developers can easily integrate it into their applications, and also allows new PSI protocols to be plugged into the `FoF Finder` service easily. Currently we are conducting extended usability tests.

Acknowledgements. This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within EC SPRIDE, and by the Hessian LOEWE excellence initiative within CASED.

References

[1] Gtalk. http://www.google.com/talk/.

[2] OpenGarden project. http://opengarden.com/ourstory.php.

[3] Serval - comunicate anywhere, anytime. http://www.servalproject.org/, visited 07/26/12.

[4] Skype service. http://www.skype.com/intl/en/home.

[5] Whatsapp messenger. http://www.whatsapp.com/.

[6] M. von Arb et al. VENETA: Serverless friend-of-friend detection in mobile social networking. In *WiMob*, pages 184–189. IEEE, 2008.

[7] E. De Cristofaro and G. Tsudik. Experimenting with fast private set intersection. In *Trust and Trustworthy Computing (TRUST)*, volume 7344 of *LNCS*, pages 55–73. Springer, 2012.

[8] E. De Cristofaro et al. Private discovery of common social contacts. In *ACNS*, volume 6715 of *LNCS*, pages 147–165. Springer, 2011.

[9] E. De Cristofaro et al. Fast and private computation of cardinality of set intersection and union. In *CANS*, volume 7712 of *LNCS*, pages 218–231. Springer, 2012.

[10] Fon Networks. http://corp.fon.com/en/.

[11] P. Gardner-Stephen. The serval project: Practical wireless ad-hoc mobile telecommunications, 2011.

[12] P. Ginzboorg et al. DTN communication in a mine. In *ExtremeCom*, 2010.

[13] T. Hossmann et al. Twitter in disaster mode: Security architecture. In *CoNEXT*. ACM, 2011.

[14] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed Security Symposium (NDSS)*. The Internet Society, 2012.

[15] Y. Huang et al. Privacy-preserving applications on smartphones. In *HotSec*. USENIX, 2011.

[16] S. Jarecki et al. Fast secure computation of set intersection. In *SCN*, volume 6280 of *LNCS*, pages 418–435. Springer, 2010.

[17] JoikuSpot, 2007. http://joikusoft.com/.

[18] A. Neumann et al. B.A.T.M.A.N.: Better approach to mobile ad-hoc networking. In *IEFT Draft*, 2008.

[19] Nokia. Nokia Instant Community. Article in Nokia Conversations Blog, May 2010. http://conversations.nokia.com/2010/05/25/nokia-instant-community-gets-you-social/.

[20] M. Pitkänen et al. SCAMPI: Service platform for social aware mobile and pervasive computing. *Computer Communication Review*, 42(4), 2012.

[21] A. S. Tanenbaum et al. Using sparse capabilities in a distributed operating system. In *ICDCS*, pages 558–563, 1986.

[22] J. Ugander et al. The anatomy of the facebook social graph. *arXiv:1111.4503*, 2011.

[23] L. Zhang et al. Message in a sealed bottle: Privacy preserving friending in social networks. *CoRR*, abs/1207.7199, 2012.

Appendix A `FoF Finder` Service Interface

Name	Input	Output	Invoker	Description
$InitInitiatorEngine$	*type*	*IIC*	$\mathcal{I}$	Initiate protocol engine
$InitResponderEngine$	*IIC*	*RIC*	$\mathcal{R}$	Initiate protocol engine; compute K_{IR}
$FinalizeInitiatorEngine$	*RIC*		$\mathcal{I}$	compute K_{IR}
$InitiatorFirstStep$		*IDC*	$\mathcal{I}$	Trigger PSI
$ResponderStep$	*IDC*	*RDC, RRC*	$\mathcal{R}$	Trigger PSI; extract K_{IR}
$InitiatorFinalStep$	*RDC*	*IRC*	$\mathcal{I}$	extract PSI result, K_{IR}

Table 1: `FoF Finder` service interface

Notation	Description	Constituent data
type	Type of `FoF Finder` service needed	PSI type, hop length
IIC	Initiator Initial Container	type, PK_I
RIC	Responder Initial Container	type, PK_R
IDC	Initiator Data Container	payload from PSI state machine
RDC	Responder Data Container	payload from PSI state machine
IRC	Initiator Result Container	$K_{\mathcal{IR}}, CommonFriends$
RRC	Responder Result Container	$K_{\mathcal{IR}}$

Table 2: Parameters in the `FoF Finder` service interface