PeerShare: A System Secure Distribution of Sensitive Data Among Social Contacts

Marcin Nagy[1], N. Asokan[2], Jörg Ott[1]

[1] Aalto University, Finland
{marcin.nagy, jorg.ott}@aalto.fi
[2] University of Helsinki, Finland
asokan@acm.org

Abstract. We present the design and implementation of the PeerShare, a system that can be used by applications to securely distribute sensitive data to social contacts of a user. PeerShare incorporates a generic framework that allows different applications to distribute data with different security requirements. By using interfaces available from existing popular social networks, PeerShare is designed to be easy to use for both end users as well as developers of applications. PeerShare can be used to distribute shared keys, public keys and any other data that need to be distributed with authenticity and confidentiality guarantees to an authorized set of recipients, specified in terms of social relationships. We have used PeerShare already in three different applications and plan to make it available for developers.

Keywords: Data distribution, social networks, access control

1 Motivation

Key management has been one of the challenging problems in guaranteeing secure communication between parties on the Internet. Although *public key technology* holds the promise of simplifying key management, multiple technical, economic, legal and social reasons prevented PKIs to be successfully deployed in the Internet, thus leaving the problem of secure key distribution still wide open [11].

Recent years have also brought a tremendous increase in the popularity of social networks. Social networks (like Facebook[1], or Twitter[2]), by giving users opportunity to share data among their friends and by providing APIs for third party developers, open new possibilities for creation of applications using social graph data. The most important aspects of social networks in the context of data distribution are:

- the possibility of common user authentication by means of the *Single Sign-on* service and the *OAuth* protocol [12],
- the extensive scale of deployment of popular social networks, and
- the ability for users to express social relationships in an intuitive manner (e.g., "friends", "colleagues", "friends of friends" etc.)

Our starting point is the observation that one can use social networks to facilitate the distribution of authentic public keys [15]. One can generalize this to design a generic framework that allows distribution of arbitrary application-specific sensitive data with specific security requirements (like authenticity-only or authenticity and confidentiality) to a specific set of social contacts. The result is a system we call PeerShare, which we describe in this paper.

Our goal and contribution. In this paper we present PeerShare: the design and implementation of a system that allows users to distribute data securely. PeerShare distinguishes itself from other data distribution systems through:

- incorporating **a generic framework for data distribution** that can be used by different applications to distribute different types of data (e.g., shared secret keys, public keys, other sensitive data) to a specified set of social contacts with different security guarantees.
- improving **usability both for end-users and application developers** by taking advantage of existing and popular social network tools for the user authentication and distribution of data inside a specific social context.

In our implementation, we use on Facebook as the social network. The social network server is used for user authentication and for users to define social groups either as pre-defined lists (like "friends" or "friends-of-friends") or custom lists. However, our system is generic and can use any social network that supports a single sign on (SSO) and authorization mechanism (like OAuth 2.0) and provides an interface for apps access user's social graph information. Given the scale of social networks deployment, the SSO using the social network greatly increases the usability of user authentication. Social graph information is used only for obtaining user specific friend lists which can be used to specify access control for the data being distributed.

The social network server is not involved in actual data distribution; that is done through the PeerShare server containing database with data to be distributed to specified users. PeerShare client-side implementation, called the PeerShare Service, is responsible for uploading new data to the server, deleting old data items, and periodically querying the server to check if there are any new data for them uploaded by other devices. PeerShare Service exposes an API towards applications which allow them to make use of PeerShare functionality. The communication between the client and the server is via an authenticated secure channel. PeerShare server is assumed to be a trusted entity. In Section 5, we discuss ways of reducing this trust assumption.

Outline. We describe usage scenarios in Section 2 which motivate usage of PeerShare. Section 3 presents system requirements. Section 4 includes detailed system design, while Section 5 describes security considerations of the system. Section 6 presents the related work. Finally, section 7 concludes the paper.

2 Usage scenarios

Currently PeerShare is already used in three example applications, namely PeerSense [10], SCAMPI [16] and CrowdShare [3][4].

PeerSense [10] is a service on a mobile device that senses the presence of nearby friends. Applications can query PeerSense for the set of nearby friends at any given moment(e.g. the camera application can query the set of nearby friends at the time when the shutter is pressed and attach this information as metadata to the resulting picture [17]. PeerSense uses `PeerShare` to distribute the binding between the Bluetooth Device Address (BDADDR) and the social network identifier of the device user to the set of users who the device user wants to be visible to. Although BDADDR itself is not secret, the binding is. Hence, the data share via `PeerShare` needs secrecy and authenticity. At any given time a device is associated with at most one user, whereas a user may be associated with multiple devices. Thus, the data is device-specific.

The *SCAMPI* platform [16] allows mobile devices to communicate in the opportunistic network. Mobile devices discover themselves through multicasting their SCAMPI identifiers, which are hashes of their public keys. As such identifiers are not very meaningful to use, the SCAMPI platform uses `PeerShare` for mapping the SCAMPI identifiers to social identifiers. Similarly to PeerSense, the exchanged data is a binding, thus it is private and also device-specific.

In the *CrowdShare* project [3][4], devices in the network are able to share their resources with one another based on existing social relationships. CrowdShare presents privacy-preserving friend of friend finder service based on the private set intersection (PSI) algorithm [7]. The input to PSI consists of a set of "bearer tokens". A bearer token is generated by the device of a user and is distributed to all friends of that user. It serves as a capability for proving the friend relationship. The data shared using `PeerShare` are the bearer tokens. They are user-specific and require both authenticity and integrity.

Furthermore, CrowdShare can optionally make use of user-specific public keys. Distribution of public keys is done similarly to the SocialKeys project [15], but using `PeerShare`. The exchanged data would then be a binding between a public key and a social identifier. Thus, such binding is public and user-specific.

Table 1 presents a short summary of existing `PeerShare` use cases.

Table 1: Summary of existing `PeerShare` use cases

Use case	**Type of data**	**Security need**	**Specificity**
PeerSense	BDADDR:social-ID binding	private	device-specific
SCAMPI	SCAMPI-ID:social-ID	private	device-specific
FoF finder	bearer token	private	user-specific
Public key distribution	public key	public	user-specific

Finally, there are also possible situations in which, we are interested in making a binding between a data item and a specific user that does not use the `PeerShare` system. To do this, we introduce also the notion of data binding type. If data is uploaded normally by the application, we call it a owner-asserted binding. However, if a user decides to add a binding for another user, such binding is called user-asserted, and is only visible to the user that has created it. An example of such a situation is present in the PeerSense application. A user can tag a device on the list of scanned devices and assign a name of his/her friend to it. Since there is no evidence for the correctness of such a user-asserted binding, it is not distributed using `PeerShare`, but is still available via the `PeerShare` API in the devices of the user who asserted the binding.

Given the common aspects of these different cases of secure data distribution, it is evident that designing a generic data distribution framework would improve ease of development, use and security.

3 System requirements

3.1 `PeerShare` goals

Our vision of a successful data distribution system sets three basic goals for it to fulfil:

1. **Data security** ⇒ assurance of data security is the most critical requirement to convince users to the distribution system.
2. **Usability** ⇒ the system should be as easy to use as possible for users, thus necessary user interaction should be minimised.
3. **Deployability** ⇒ the system should be scalable to allow various application developers to easily distribute their data though it.

3.2 Assumptions

Our threat model assumes that each device has platform security that:

- isolates applications from one another (both during execution time and in terms of persistent storage),
- allows a service on the device to learn a platform-specific identity of a calling application that wants to access the service.

3.3 Threats

We need to provide protection against the following threats:

1. **Man-in-the-middle Attacks** ⇒ any network devices that route messages between the mobile device and the server should not be able to act as man-in-the-middle that eavesdrops on or modifies messages.
2. **Unauthorized Usage** ⇒ only the person that has created the data item should be able to later modify or erase it. Furthermore, as data are created by applications that use the `PeerShare` system, only the application that has created the particular data item should be able to access, modify or delete it. Finally, data should be distributed by the `PeerShare` server only to users that are eligible to obtain them.

3.4 Security requirements

Communication channel protection is required to prevent a man-in-the-middle attack. The threat of unauthorized usage motivates usage of server authentication, mobile application authentication, user authentication and application access control.

4 System design

The PeerShare system allows for creation, storage and distribution of application specific data securely. The system consists of two main components: (1) PeerShare Service, and (2) PeerShare Server, which are described below. Figure 1 illustrates the system overview.

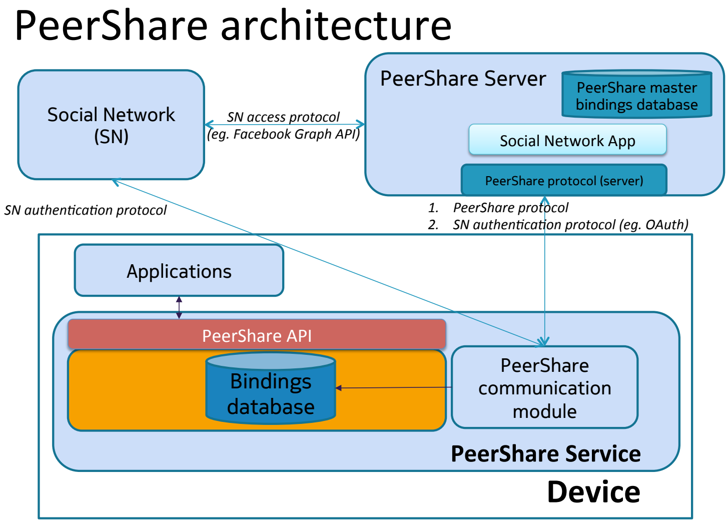

Fig. 1: PeerShare architecture.

4.1 PeerShare Service

The PeerShare Service is the encapsulation of the PeerShare functionality on the mobile device. It is exposed to other applications via the PeerShare API described later in the section. It is also responsible for communication with the PeerShare Server, which is described in the section 4.3, along with the protocol between the client and the server.

The heart of the service is internal database that stores data together with their mappings to social identities, which are obtained from the server. The database security is guaranteed by the mobile platform security. Data are bound to social identities by means of social network authentication.

The service stores a data item inside the AppData data structure whose attributes are described in the table 2.

Furthermore, the service provides application level data access control which guarantees that only authorized applications can modify or delete existing data. Any application can create data that it intends to share using the system. During the initial data upload process, the service records the calling application platform specific identifier (e.g. pair of Android package name and developer key) and appends it to the created data. As a result, if an application wants to modify or delete existing data, the service learns the calling application identifier and verifies it against the application identifier recorded in the initial upload process.

Table 2: Summary of PeerShare data attributes

Data attribute	Description
Data type	Mapping of data to particular type, or application
Data value	Actual value of data
Data description attributes	Provides more detailed description of a data item by indicating algorithm used for its creation, specificity, sensitivity and binding type
Data sharing policy	Specifies sharing policy for data
Timestamps	Indicates timestamps for data creation and its validity
Social information	Social information that is bound to data
Creator application identifier	Identifies mobile platform and application that owns a data item

The PeerShare API is the second part of the mobile application. It provides the interface for third-party applications for creation, modification and removal of data that an application wishes to share with other application users. The most important methods are described in the table 3:

Table 3: Summary of PeerShare API

Method	Description
long **addData**(*AppData* data)	Stores application data and uploads it to the PeerShare Server as soon as the network connectivity to the server is established. Returns object identifier which is used later to modify/delete it.
int **updateData**(*long* objectID, *AppData* data)	Modifies already existing data. The object ID obtained in the *addData* method identifies data to update.
int **removeData**(*long* objectID)	Deletes existing data. The object ID obtained in the *addData* method identifies data to remove.

4.2 PeerShare Server

The PeerShare Server is the trusted entity that is primarily responsible for secure storage of data. Every data item is bound to a social identifier of the user (e.g. Facebook user ID) who has created the item. The server authenticates users by requiring them to provide a valid social network user access token and verifying its correctness through interaction with the social network server.

The second crucial responsibility of the server is enforcement of access control policies for stored data. The system allows users to specify who is eligible to access stored data by allowing them to state the sharing policy from all available social network user lists of the user. The server queries the social network server for custom friend lists created in the social network by a user. Such lists are returned to the PeerShare service and can be further accessed by other applications to allow them specify the sharing policy. On creating a new data item, or updating sharing policy for the existing item, the server queries the social network server to obtain the list of social user IDs applicable to download particular data item. If an application uploading a new data item does not specify its sharing policy, the data item is by default shared among all user's friends.

Furthermore, the server updates lists of users assigned to particular friend list in case of their modification by means of the realtime updates provided by the social network. If the social network does not have this functionality, the server must regularly poll the social network server to learn about such changes.

4.3 PeerShare Protocol

The PeerShare service communicates with the server through the JSON encoded protocol that runs on top of standard HTTPS protocol. It involves following operations: user registration and unregistration, data upload and update, and data download. Because data exchanged between the service and the server are sensitive, communication security is guaranteed by the TLS layer. Furthermore, to protect against fake social network application attack, each request includes user access token of the social network, whose validity is verified by the server.

In the ***REGISTER*** method, the user registers for the service by informing the server about his/her social information. In response, the server generates (or finds if the user is not a new one) user's PeerShare identifier that is needed in all subsequent transactions with the server to uniquely identify the correct person. The PeerShare ID is necessary to properly correlate possible multiple social identities of the same person. This may happen if someone uses more than one social network in the PeerShare system (e.g. Facebook and Twitter).

In the ***UPLOAD*** method, the service sends to the server all data items which have been added to the database on the user's device, but have not been uploaded on the PeerShare server. The message contains also the PeerShare ID to map uploaded data to a specific user. Content of each data item is consistent with data description provided in the section 4.1. In response to the *UPLOAD* request, the server sends an array of object IDs that are later used to modify or delete every data item from the server database.

The ***UPDATE*** method is very similar to *UPLOAD*. The only difference is that it is not adding any new data on the server, but only updating existing ones. The object ID returned in the *UPLOAD* operation is needed to properly identify the item on the server to modify. If the service wants to update a non-existing item (i.e., the one that has already been deleted by the user), the server ignores the request to do it, and sends back in response notification that the data item does not exist and should be removed from the local database.

The ***DOWNLOAD*** method allows the service to fetch all data items that the registered user is eligible to obtain. It requires the service to provide user's PeerShare ID to correlate the request with the correct user. In response, the server returns an array of data items that contain detailed information about each item in a format similar to the one used in the *UPLOAD* or *UPDATE* request. The only difference is that personal information (i.e., sharing policy, and object ID) is not included unless the downloading user owns the item.

The ***DELETE*** method is used to erase old and no longer needed data from the server. Similarly to all other methods, it must include the PeerShare ID to correctly correlate user with data. In addition, it also contains an array of object identifiers that are to be deleted. In response, the server sends just status information that is either *OK* if there are no errors, or is an error message.

UNREGISTER is the final method defined in the protocol. It is used to unregister the user from the PeerShare and delete all data associated with the user. In the request, the PeerShare ID together with the social network identifiers are provided. The server responds with *OK* status, or an error message if operation fails.

4.4 Implementation

Our server implementation is written in PHP, and uses PostgreSQL database and the Facebook PHP SDK. Currently the server supports only Facebook as the social network to authenticate with, but the architecture is generic enough, so that in the future it can be easily extended to support other social networks. Finally, the server takes advantage of Facebook Realtime Updates functionality to learn about modifications of user's lists.

The service implementation is more complex, as it includes the communication module as well as API for third party applications. Currently we have the Android implementation of the client package. The service is implemented as a standard Android background service that runs as an independent process. It contains internal SQLite database, where PeerShare data are stored. Applications using the service bind to it through the AIDL interface. SSO user authentication is currently provided by native Facebook library. The service uses also Binder interface functionality to learn about service calling application identifiers. It allows matching calling applications with data they create, which is critical to provide application access control.

4.5 Deployment and performance considerations

The PeerShare system has up to now been successfully deployed in three projects, as described in the section 2.

To evaluate performance of the PeerShare, we have tested average time needed for upload and download of data in the WiFi network that uses ADSL connection. In our test scenarios, a user uploads 1 data item, and downloads 5 data items, as 5 friends share sample data in a test application. Average upload time measured in 30 runs is 2.02 seconds with standard deviation of 1.33 seconds. Download operation performance is more stable, as average time is 1.18 seconds with standard deviation of 0.12 seconds. To compare these numbers with standard web browsing activities, we conducted similar experiments for downloading mobile Facebook web page. Average download time for Facebook webpage measured in 30 runs is 1.50 seconds with 0.21 seconds of standard deviation.

Furthermore, PeerShare has been designed to allow multiple applications use the same server. However, if application performance is limited due to the server scalability, each application developer may decide to run its own server.

Such a solution is further discussed in the section 5.3 describing possibility of minimizing the need of trust for the PeerShare Server.

5 Security considerations

In this section, we present our security analysis showing that the security requirements presented in the section 3 are fulfilled.

5.1 Channel protection

In order to guarantee channel protection, PeerShare Protocol is executed over a secure (i.e., confidential and mutually authenticated) channel. The user is authenticated by the OAuth protocol via the native Android Facebook application. Therefore the system relies on correct behaviour of the native Android Facebook application.

PeerShare Server is authenticated via a TLS certificate, We use a form of "certificate pinning" by embedding the TLS server certificate of PeerShare Server in the client implementation. This protects against a rogue server from masquerading as PeerShare Server even if the rogue server has succesfully obtained a certificate for its TLS keypair from one of the tens of Certification Authorities that are normally trusted for TLS. If there are many PeerShare Servers, then instead of hardwiring the TLS server certificate, we can use standard certificate pinning [9].

5.2 User and application authentication

User authentication is obtained through the native Facebook Android library. Prior to invoking any interaction with the server, the service asks the native Facebook application (through the library interface) for a valid access token associated with the authenticated user. Such a valid token must be included in every message exchanged with the server. The PeerShare Server uses the Facebook graph API token debug tool to examine its validity by checking the following:

- does application identifier encoded inside the token correspond to the PeerShare Facebook application identifier
- does user identifier encoded inside the token correspond to the social identifier included in the sent message

The former checking protects the server against allowing a fake Facebook application to modify data on the server. The latter one prevents other users from modifying or deleting data that do not belong to them. Only if both above conditions are fulfilled, the PeerShare server proceeds with the request. Otherwise, it responds to the sender with authentication error.

User access control The user access control must guarantee that only eligible users are able to obtain data from the PeerShare server and that only the user that has created a particular data item can modify or delete it. Correct data distribution is secured by the server that learns data sharing policy from the request to store/update the data item. For each created/updated sharing policy, the server interacts with the Facebook graph API to fetch the list of social user IDs associated with the given policy. Having obtained such list, the server stores information about social IDs eligible to download particular data item.

The second problem is resolved on the device side. Whenever a third party application makes a request to modify or delete the particular data item, the PeerShare service checks in the local database if the item has been created by the user trying to modify it. If this is true, the service grants application permission to edit or remove the item. Otherwise, it denies application access to the given item.

Application access control As multiple applications on one mobile device can use the PeerShare system, there is a threat that a malicious application using the system can modify or delete a data item that does not belong to it. In order to protect against this threat, the PeerShare service has built-in application level access control enforcement. When an application creates a new data item, and wants to have it distributed through the PeerShare system, the service tries to infer the platform specific calling application identifier and appends it to the uploaded data. For Android operating system, the application identifier is a tuple of package name and developer public key that can be obtained through the Binder interface. On subsequent requests to update/delete the data, the service again obtains the identifier of the calling application and compares it with the one associated with object as the creator. In the Android operating system, this function is performed by verification if package signatures match. Unfortunately, some operating system may not permit for the service to infer the calling application identifier. In such case, the caller must explicitly specify the application identifier.

5.3 Minimizing the need to trust PeerShare Server

Since PeerShare Server has access to all the sensitive data, it needs to be trusted by all participants. This is a rather strong assumption. There are two ways to reduce the extent to which PeerShare Server needs to be trusted:

Use of trusted hardware: If PeerShare Server is equipped with a hardware security module (HSM) like the Trusted Platform Module (TPM)[3], then PeerShare server database can be encrypted using a HSM-resident key. HSM will decrypt the plaintext and make it available to a process if and only if the host computer is in the correct configuration (i.e., running the correct PeerShare Sever software). An attacker will have to subvert PeerShare Server process at runtime. If the client devices also have a hardware-based trusted execution environment (like On-board Credentials [14], then the server HSM can encrypt the

[3] http://www.trustedcomputinggroup.org/resources/tpm_main_specification

sensitive data so that it is accesible only within the client TEE, thereby not exposing it to the PeerShare Server at all.

Application-specific PeerShare Server: Although we designed PeerShare in such a way that multiple application developers could use the same PeerShare Server, in practice, each application developer could decide to host her own independent PeerShare Server. This would still allow the benefit of developer ease of use because developers can re-use our PeerShare implementation, without asking all developers to trust the same server.

6 Related work

The concept of data sharing with social networks support is present also in other works. SocialKeys [15] project proposes the idea of distributing public keys via social networks. PeerShare extends this concept to: various types of data and multiple applications.

Backes et al. [5] presents a generic cryptographic framework that allows social relations establishment and resource sharing with user anonymity, secrecy of resources, privacy of social relations and access control secured. Unlike PeerShare that uses social network specified sharing policies, it requires users to explicitly establish social relationships with other users which makes it less intuitive for users in real deployments.

Baden et al. has implemented Persona [6], a distributed social network with distributed data storage. It provides data access control by employing a combination of traditional public key cryptography and attribute-based encryption (ABE) scheme that involves more complex key management. Safebook [8] is the implementation of the distributed social network that improves privacy protection mechanisms in comparison to other existing social networks. Improved privacy results from taking advantage of cooperation between users inside a peer-to-peer overlay network, named matryoshka, and trust relations among users to achieve integrity and privacy properties. Unlike concepts presented in these works, our goal is not to build a new social networks, but to make use of existing social networks, as one of the requirements of the system is its deployability. Obviously, if any of these social networks proves to be successful, we are interested in taking advantage of their security and privacy mechanisms in PeerShare.

Jahid et al. has implemented DECENT [13] that is a decentralized social network system providing confidentiality and integrity of data that due to cryptographic mechanisms can be stored in untrusted nodes. Unlike our system, it requires user to explicitly specify data sharing policy and build their social relationships, thus it is more difficult to use in real life than PeerShare.

7 Conclusion

We have described the design and implementation of PeerShare, the system that allows users for secure and privacy preserving data distribution system. It implements a generic framework for data distribution that can be used by different applications and provides different levels of security guarantees. Furthermore, the system enhances general usability for end-users and application developers by using existing and popular social network mechanisms for the user authentication and data distribution inside a specific social context.

PeerShare has been used by three different applications. We plan to make it available to other application developers.

References

1. Facebook. http://www.facebook.com.
2. Twitter. http://www.twitter.com.
3. N. Asokan, Alexandra Dmitrienko, Marcin Nagy, Elena Reshetova, Ahmad-Reza Sadeghi, Thomas Schneider, and Stanislaus Stelle. Crowdshare: Secure mobile resource sharing. Technical Report TUD-CS-2013-0084, TU Darmstadt, April 2013. http://www.trust.informatik.tu-darmstadt.de/publications/publication-details/?no_cache=1&tx_bibtex_pi1%5Bpub_id%5D=TUD-CS-2013-0084.
4. N. Asokan, Alexandra Dmitrienko, Marcin Nagy, Elena Reshetova, Ahmad-Reza Sadeghi, Thomas Schneider, and Stanislaus Stelle. CrowdShare: Secure Mobile Resource Sharing. In *ACNS*, 2013. to appear.
5. Michael Backes, Matteo Maffei, and Kim Pecina. A Security API for Distributed Social Networks. In *NDSS*, 2011.
6. Randolph Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. In *SIGCOMM*, pages 135–146, 2009.
7. E. De Cristofaro, P. Gasti, and G. Tsudik. Fast and private computation of cardinality of set intersection and union. In *Cryptology and Network Security (CANS)*, volume 7712 of *LNCS*, pages 218–231. Springer, 2012.
8. L. A. Cutillo, R. Molva, and T. Strufe. Safebook: A privacy-preserving online social network leveraging on real-life trust. *Comm. Mag.*, 47(12):94–101, December 2009.
9. C. Evans, C. Palmer, R. Sleevi, and Google Inc. Public Key Pinning Extension for HTTP, 2013. IETF Internet Draft, draft-ietf-websec-key-pinning-04.
10. Aditi Gupta, Markus Miettinen, Marcin Nagy, N. Asokan, and Alexandre Wetzel. Peersense: Who is near you? In *PerCom Workshops*, pages 516–518, 2012.
11. Peter Gutmann. PKI: It's not dead, just resting. *IEEE Computer*, 35(8):41–49, 2002.
12. D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749 (Proposed Standard), October 2012.
13. Sonia Jahid, Shirin Nilizadeh, Prateek Mittal, Nikita Borisov, and Apu Kapadia. DECENT: A decentralized architecture for enforcing privacy in online social networks. In *PerCom Workshops*, pages 326–332, 2012.
14. Kari Kostiainen, Jan-Erik Ekberg, N. Asokan, and Aarne Rantala. On-board credentials with open provisioning. In *ASIACCS*, pages 104–115, 2009.
15. A. Narayanan. Social keys: Transparent cryptography via key distribution over social networks. In *The IAB Workshop on Internet Privacy*, 2010.
16. Mikko Pitkänen et al. SCAMPI: Service platform for social aware mobile and pervasive computing. *Computer Communication Review*, 42(4):503–508, 2012.
17. Chuan Qin, Xuan Bao, Romit Roy Choudhury, and Srihari Nelakuditi. Tagsense: a smartphone-based approach to automatic image tagging. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 1–14, New York, NY, USA, 2011. ACM.