Annales Mathematicae et Informaticae
44 (2015) pp. 111–120
http://ami.ektf.hu

On a secure distributed data sharing system and its implementation

Péter Kasza, Péter Ligeti, Ádám Nagy

Eötvös Loránd University
Department of Computeralgebra
pkasza@caesar.elte.hu, turul@cs.elte.hu, spigy88@inf.elte.hu

Submitted September 23, 2014 — Accepted February 10, 2015

Abstract

In this paper we propose a decentralized privacy-preserving system which is able to share sensible data in an encrypted way, that only predefined subsets of authorized entities can recover the data after getting an additional alarm message. In the paper we give a short description of the necessary cryptographic building blocks and the communication protocol. Furthermore, we present the main communication channels and the implementation of the proposed data sharing system. The proposed system achieves the desired functionalities by using secret sharing and two communication networks: an ordinary P2P network where the encrypted information is stored, and a smaller private P2P network called friend-to-friend network, which consists of the authorized parties and handles messages that are necessary to the decryption. The main part of the paper concentrates on the implementation of the system.

Keywords: private P2P network, secret sharing, symmetric cryptography

MSC: 94A62, 68P25

1. Introduction

1.1. Motivation

The number and role of smart devices show an intensive growth nowadays, they are collecting, storing and sending a large amount of sensitive data about the owner of the device. Communication devices, like smartphones or tablets can have several built-in sensors, such as accelerometers, digital compasses, gyroscopes, GPS trackers, microphones and cameras. Dedicated self-tracking devices measuring various medical data of the user, like blood-pressure, pulse, blood-sugar level, etc. The sensible data collected by these devices are often handled (additionally stored and analyzed) by a central entity, usually a mobile or cloud service provider, raising serious privacy concerns. The main goal of this paper is to propose a privacy-preserving communication framework, wherein the sensible data is not stored by a singular trusted third party, but instead distributed to some predefined subset of users such that large coalition of users is necessary to recover the data. In the proposed solution the encrypted data packages are stored within an open P2P network and the necessary decryption key is distributed in a private network called friend-to-friend (or F2F) network. In order to decrease the communication and space consumption of the users and avoid a possible adversarial tracking, it is required that the original data can be recovered only in the presence of a special event, called an *alarm message.* Examples of such alarm messages are extremities in medical data, S.O.S. signals in case of a physical attack or stroke, etc.

1.2. Communication building blocks: P2P and F2F networks

A *peer-to-peer (P2P) network* is a type of decentralized and distributed network architecture in which the individual nodes of the network (called peers) act both as suppliers and consumers of resources, in contrast to the client-server model where client nodes request access to resources provided by central servers. In our protocol the P2P architecture is used for file sharing; the participants are able to search, upload and download messages from the P2P network.

A *friend-to-friend (or F2F) computer network* is a type of private peer-to-peer network in which the users only make direct connections with people they know. Unlike other kinds of private P2P networks, the users in a friend-to-friend network cannot find out who else is participating beyond their own circle of friends, so F2F networks can grow in size without compromising their users' anonymity. Many F2F networks support indirect anonymous or pseudonymous communication between users who do not know or trust one another. For example, a node in a friend-to-friend overlay can automatically forward a file (or a request for a file) anonymously between two friends, without telling either of them the other's name or IP address. These friends can in turn automatically forward the same file (or request) to their own friends, and so on. Historically, the first F2F system was Turtle [3], recent examples of popular implementations are RetroShare [6] and OneSwarm [5]. For the underlying P2P system we chose the BitTorrent protocol's DHT network and implemented our own friend-to-friend scheme on top of it by creating encrypted communication channels between the nodes.

1.3. Cryptographic primitives: secret sharing and symmetric cryptography

A *secret sharing scheme* is a method of distributing secret data among a set of participants so that only specified qualified subsets of participants are able to recover the secret from its parts of information called shares. In addition, if the unqualified subsets collectively yield no extra information, i.e. the joint shares are statistically independent of the secret, then the scheme is called perfect. For a given positive integer t a secret sharing scheme is called t-threshold, if every subset of participants with cardinality at least t can recover the secret.

Secret sharing was first introduced independently by Blakley [1] and Shamir [7]. In both papers the authors constructed perfect t-threshold schemes. Here we present the method of Shamir, which can be easily implemented due to its simplicity.

Example 1.1 (Shamir)**.** Let the participants indexed by the non-zero elements of a finite field $\mathbb{F}$ and let p be polynomial of degree at most $t-1$ over $\mathbb{F}$ chosen uniformly at random. The share of participant i is $p(i)$ and the secret is the the constant term of $p(x)$, i.e. $p(0)$.

In the proposed protocol we assume that at least some of our friends in the F2F network are trustful i.e. can be expected to follow the protocol. We use the pairwise secret channels between the participants for communication. The security of these end-to-end communication channels are guaranteed by *symmetric cryptographic primitives*, especially by symmetric encryption schemes. Informally, a *symmetric encryption scheme* consist of three algorithm: the *key-generation*, where the common secret key is established and sent on a secret channel; the *encryption* where the sender computes the ciphertext of a given message using the symmetric key and the *decryption*, where the receiver computes the message from the ciphertext using the symmetric key. A message (key) of an encryption scheme is *computationally secure* if any probabilistic polynomial time adversary is able to learn some information about the message (key) with negligible probability only. Note that, this is rather an informal definition, but here we just highlight the main cryptographic ideas and results: we use that most of the widely used encryption schemes are proven to be computationally secure. For example, in the case of the OneSwarm network, RSA is used: every user generates a 1024 bit public/private RSA key pair when installing the client, with the public key serving as its identity. After a key-exchange between the friends, the participants can connect to one another using secure sockets (SSLv3) bootstrapped by their RSA key pairs. Furthermore forward security can be achieved by establishing ephemeral Diffie-Hellman keys between the participants.

2. Protocol description

Within this section we present an informal description of the proposed protocol together with the desired security requirements. This paper concentrates on the communication channels and the implementation details, hence the exact protocol description and the proof of security is contained in a separate paper [4].

2.1. Parameters

The participants of the protocol are the following: Alice, the sender of the data and the alarm message, Alice's friends in the F2F network and further participants using an open P2P network. We suppose two communication channels: a F2F network consists of Alice and her friends and a P2P network. Alice is able to make a digital signature for integrity protection and the encryption for every message and key. Alice has a secret symmetric key with every friend of her in the F2F network.

2.2. Protocol description

The protocol has three main phases: the first one is *Uploading*, where the sender first generates a temporary key and uploads the encrypted message in the P2P network. Next, the sender distributes the collection of encrypted temporary keys together with the list of identifiers of other shares and the message according to a t-threshold secret sharing scheme. Finally, the shares are sent to her friends in the F2F network.

The second step is the *Downloading* phase, in which the alarm message is sent first to the friends. After getting the alarm signal, the friends distribute their encrypted shares into the P2P network and then download the remaining parts from the P2P network based on the identifiers of the shares and the message.

The last stage is the *Message recovery* step, where every friend checks the integrity and authenticity of the downloaded packages, computes the encrypted temporary key from the correct shares and decrypt the session key with the symmetric key and the original message with this decrypted key as well.

2.3. Security model

We suppose two opposite user behaviors. The first kind of participants is called *honest*, meaning that they always follow the steps of the protocol and compute/send nothing more. The other extremity is a *malicious* participant who is not supposed to send messages according to the protocol, but is not able to interrupt the communication. Because the connections in a F2F network are based on real personal relationships and trust, we will suppose that the honest friends are in majority.

Intuitively, we need that if there are "many good friends" then every friend is able to get the message when the protocol finishes (even the "bad" ones). Furthermore, it is necessary that no small subset of participants can learn any information about session key (including non-friends) as long as Alice doesn't send the alarm signal and any subset of entities out of the F2F network learns nothing about the message. Here we collect the precise security requirements that the proposed scheme has to satisfy:

- ***Correctness:*** if there is a set of at least t honest friends, then every friend can recover the original message at the end of the protocol.
- ***Key Privacy:*** the session key used for encryption/decryption of the message is computationally secure against any coalition of participants of cardinality less than t, before getting the alarm message.
- ***Message Privacy:*** the message is computationally secure against any coalition of participants who are not friends of the sender in the F2F network.

2.4. Security analysis

Here we only present the main theorems providing the above requirements without proofs, the particular analysis can be found in [4].

Theorem 2.1. *If Alice uses a perfect t-threshold secret sharing scheme in the Uploading then the system fulfills the Correctness requirement.*

Theorem 2.2. *If Alice uses a perfect t-threshold secret sharing scheme in the Uploading and a computationally secure encryption scheme, then the system fulfills the Key Privacy requirement.*

Theorem 2.3. *If Alice use a computationally secure encryption scheme, then the system fulfills the Message Privacy requirement.*

From the implementation point of view, it is enough to use a perfect secret sharing scheme, like Shamir's scheme 1.1 and computationally secure encryption in the implementation of the used F2F network.

3. Implementation details

The created application – called *Siren* – realizes the requirements described above: the users can send and receive encrypted information (location, special message, etc.) to and from their friends and it can be restored by them if and only if the communication breaks unexpectedly or there is an alarm message. In the case of emergency (i.e. some friend sent or triggered an alarm message) the application will automatically put out the known shared pieces to the peer-to-peer network and will try to gather enough piece to restore the key for the encrypted information. The decrypted information will be shown to the user as a pop-up warning message on the device.

3.1. Structure

Each instance of the Siren application is made up of a Signaling, a Processor, a F2F network layer and a P2P network layer module. The modules are responsible for different aspects of the program. The Signaling and Processor modules implement the core protocol described above in 2.2. The F2F and P2P layers provides a simple socket based interface to the underlying friend-to-friend and peer-to-peer network.

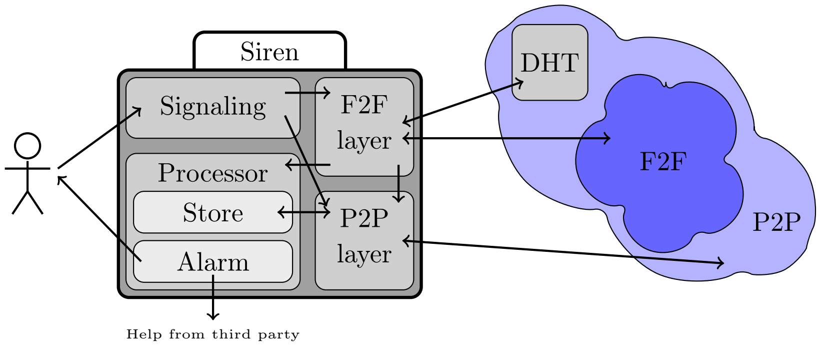


Figure 1: Modules of the Siren application

3.1.1. Signaling

The Signaling module interacts with the user through a graphical interface, makes and distributes heartbeat messages as specified in the Uploading phase or sends a panic signal if requested. After starting the module it keeps making and distributing heartbeat messages with a given frequency until the user stops the module or it becomes impossible to send any message. When the user stops the module, it will send a special closing message to the friends (not an alarm), hence they will don't start the Upload phase.

The heartbeat messages contain shared pieces of an unique key – which was used to encrypt information about the user – as well as some parameters about when and how the alarm message will be sent or should be triggered. Among others every message contains an expiration time and a deadline too. After the expiration time the receiver's Processor module will drop the message; while after the deadline an alarm will be triggered unless another newer message arrives from the sender.

3.1.2. Processor

The Processor module has two important tasks. The first task is to store the messages sent by friends from both F2F and P2P networks. It will organize and keep the messages until they are expired (the expiration time sent within the heartbeat message) and can announce them on the peer-to-peer network.

The other task of this module is to listen to alarm messages. If a friend has sent or triggered such message then it will gather the shared data from the P2P network, decrypt the information after recovering its key as described in the Message recovery phase in 2.2 and show it to the user. Because there can be more non-expired message from the same sender, the module will simultaneously try to gather all the required number of pieces for every message. Hence if there is a chance to recover a message that is not expired then it will be.

3.1.3. F2F Layer

The F2F layer abstracts away the F2F network, providing a simple socket interface where the individual nodes can be addressed by their identifiers calculated from their public keys.

To build up communication between two F2F nodes, their network addresses has to be resolved first. In our case, the node addresses are resolved used the BitTorrent DHT network. The network address for each node consists of an (IP address, port) pair. These pairs are identified by the SHA-256 hash of the node's public key. The F2F layer maintains a cache of these (id,IP,port) triples to speed up connection requests. Between two nodes, if only one node's network address changes, this node can notify the other one of its new address. The DHT only needs to be checked if the node identifier is not found in the cache or if both node's addresses change simultaneously. The nodes must however continuously keep advertising their network addresses through the DHT network, because the DHT has a tendency to "forget" information as old nodes leave and new ones enter the cloud.

Once the network address for a node is known, an encrypted channel can be established using the public key for the node. Through this channel, the nodes negotiate an ephemeral Diffie-Hellman key, which they use to encrypt their further messages. This provides better performance then using the asymmetric encryption and also achieves forward security.

Having a fully functional P2P layer, one can calculate and advertise F2F node identifier. The identifier of a F2F node is the SHA-256 hash of the node's public key. The calculation of these keys are done in two step. Firstly at the first start of the application it generates a new RSA keypair to sign messages and identify the host[1] and send/receive invites. When a friend's identifier known it can check out his address from the DHT and start the Diffie–Hellman key exchange. At the last phase of the key exchange it receives the public key of the friend with some data (like name, etc.) and calculates the common key for the communication channel.

Separating this functionality from the main application promotes modularity and encourages reuse. We hope that our implementation of the F2F layer will provide a guideline for developing similar cryptographic software.

[1]It uses this hash calculated from the public key as identifier in the distributed hash table too.

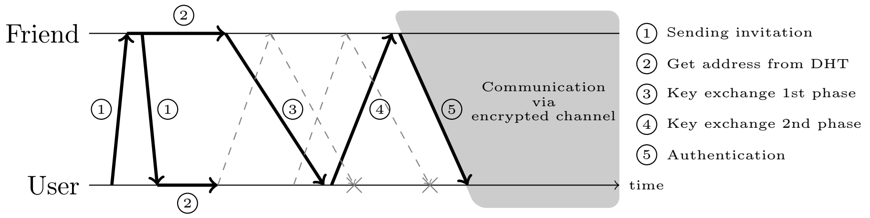


Figure 2: The process of establishing communication channel

3.1.4. P2P Layer

Bootstrapping the P2P layer. As seen in figure 1, the modules are highly interdependent, with the P2P layer being the most fundamental part of the application. The P2P layer provides a reliable address resolution mechanism for the F2F network. Because of this, the application needs to bootstrap the P2P network first to be able to use the F2F network. Bootstrapping from a handful of nodes can take some time to finish (it usually takes a few minutes), however in most cases we only need to find a few nodes only, because the addresses of the nodes are cached and stored between sessions. The cache contains a selected subset of the P2P nodes. The node selection based on their uptime and node identifier.

The nodes' uptime is usually thought of as a decreasing failure rate system (see [2]) in which the nodes with a longer uptime have a higher probability of being available for some fixed amount of time from now. If the cache is empty, we bootstrap the DHT from the following "master" nodes:

- `udp://router.bittorrent.com:6881`
- `udp://router.utorrent.com:6881`
- `udp://dht.transmissionbt.com:6881`

Communication via P2P. This implementation uses the peer-to-peer network for simple data announcing too. The encrypted data – which key was shared via the F2F network – is sent to the P2P network as well as all of the key pieces when a friend is in an emergency.

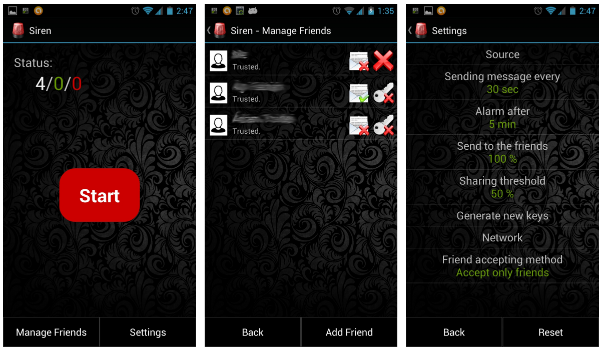
3.2. Key management

Our keypair is generated on the first run of the application and stored on the local disk for further use. The program manages a contact list of friends in the F2F network, which contains the hash of partners' public keys so that their network addresses can be resolved. The actual public keys are not stored, but they can be retrieved from an alive node after the Diffie-Hellman key exchange. One can add new friends to the contact list by sending a special invitation message which can take the form of an URL or a QR code. The invitation contains the hash of the public key and an initial network address. Both of them are signed with the private key. This prevents forgeries and guarantees the authenticity of the invitation.

3.2.1. User interface

The application was made for Android operation systems so it can be used on the majority of mobile devices. The user interface can be separated into three parts: main, friends and settings.

- In the main part of the UI the user can start/stop the signaling module and can see some numerical information how: many friend added to his network, how many is active and how many turned on the signaling module. Also the other interfaces can be reached from here.
- The friends UI was made to handle friends so here the user can add, delete, invite or disable[2] friends. Reached this interface the user will see a list which items contain friends and their connection information like invitation states (sent/received) and that the friend is active or sending heartbeat messages.
- With the settings UI the user can set the parameters of Signaling module, the F2F and P2P networks and also can change the information of the encrypted message that the others will known in case of emergency.



Source
Sending message every
30 sec
Alarm after
5 min
Send to the friends
100 %
Sharing threshold
50 %
Generate new keys
Network
Friend accepting method
Accept only friends

Figure 3: The three main graphical interfaces of the application

Acknowledgements. The research was carried out as part of the EITKIC_12-1-2012-0001 project, which is supported by the Hungarian Government, managed by the National Development Agency, financed by the Research and Technology Innovation Fund and was performed in cooperation with the EIT ICT Labs Budapest Associate Partner Group. (`www.ictlabs.elte.hu`). This research has been partially supported by the Lendület program of the Hungarian Academy of Sciences. The second author was partially supported by the grant OTKA PD-100712.

[2]Disable friends means they won't participant in the user's secret sharing but the application will receive messages from them and will warning the user if they are in emergency.

References

[1] Blakley, G. R., Safeguarding cryptographic keys *Proceedings of the National Computer Conference* Vol. 48 (1979) pp. 313–317.

[2] Carrda, D., Building a Reliable P2P System Out of Unreliable P2P Clients: The Case of KAD (2007) `http://www.eurecom.fr/fr/publication/2430/download/ce-carrda-071210.pdf`

[3] Isdal, T., Piatek, M., Krishnamurthy, A., Anderson, T., Privacy-preserving P2P data sharing with OneSwarm `http://www.oneswarm.org/f2f_tr.pdf`

[4] Kasza, P., Nagy, Á., Ligeti, P., Siren: secure data-sharing over P2P and F2F networks *submitted to Studia Scientiarum Mathematicarum Hungarica*

[5] Popescu, B.C., Crispo, B., Tanenbaum, A. S., Safe and Private Data Sharing with Turtle: Friends Team-Up and Beat the System *12th International Workshop on Security Protocols* (2004).

[6] RetroShare: secure communications with friends, available online at `http://retroshare.sourceforge.net/`

[7] Shamir, A., How to share a secret *Communications of the ACM* Vol. 22 (1) (1979) pp. 612–613.