Siren: Secure Data Sharing Over P2P and F2F Networks

Péter Kasza[a], Péter Ligeti[a], Ádám Nagy[a]

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

Abstract

In this paper we propose a decentralized privacy-preserving system which is able to share sensible data in a way, that only predefined subsets of authorized entities can recover the data; either after getting an alarm message or timing out. The protocol has two main communication channels: a P2P network where the encrypted information is stored, and a smaller private P2P network, which consists of the authorized parties and is called a friend-to-friend network. We describe the communication protocols fulfilling the desired security requirements and present some results on the implementation of the system. The main cryptographic building blocks of the protocol include symmetric cryptographic primitives and a t-threshold secret sharing scheme.

Keywords: private P2P network, secret sharing, symmetric cryptography

MSC: 94A62, 68P25

1. Introduction

1.1. Motivation

Smartphones and other ubiquitous smart devices have several built-in sensors, such as accelerometers, digital compasses, gyroscopes, GPS trackers, microphones and cameras. This is even more apparent in specialized self-tracking devices that are e.g. measuring medical data, like blood-pressure, pulse and body temperature. The sensible data collected by these devices are often stored by a central entity, usually a mobile or cloud service provider. 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 the large coalition of users is necessary to recover the data.

In the proposed solution the data is stored within a special private network called a friend-to-friend (or F2F) network and distributed using a given secret sharing scheme. Furthermore, it is required that the original data is 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 [2], recent examples of popular implementations are RetroShare [4] and OneSwarm [3]. 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.

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 [5]. 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.* 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

2.1. Parameters

The participants of the protocol are the following: Alice, the sender of the data and the alarm message, Alice's friends $\{F_1, \ldots, F_n\}$ 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 $\{F_1, \ldots, F_n\}$ and a P2P network. Alice is able to make a digital signature $Sign_A(m)$ and the encryption $E_k(m)$ for every message $m \in \{0,1\}^*$ and key k. Alice and her friend F_i have a symmetric secret key k_i established in the F2F network, the friend F_i can decrypt the ciphertext $E_{k_i}(m)$ using this symmetric key. Alice chooses a message m, a session key k_m and a security parameter $\lceil \frac{n+1}{2} \rceil < t \leq n$.

2.2. Security model

From the users point of view, the simplest case is when we suppose that every participant is *honest*, meaning that they always follow the steps of the protocol and compute/send nothing more. The other extremity is the *malicious* participant who may follow some steps of the protocol (possibly none), but cannot interrupt the communication. From now on we suppose that the honest friends are in majority, which is a realistic assumption in a F2F network. Here we collect the main security requirements that the proposed scheme has to satisfy:

- ***Correctness:*** if there is set of at least t honest friends, then every friend can recover the original message m.
- ***Key Privacy:*** the session key k_m 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 m is computationally secure against any coalition of participants who are not friends of the sender in the F2F network.

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 and then distributes the shares containing the encryption of the temporary key to her friends in the F2F network according to a t-threshold secret sharing scheme. The second step is the *Downloading* phase, in which the friends distribute their encrypted shares and then download the remaining parts from the P2P network, if they receive the alarm message. The last stage is the *Message recovery* step, where every friend computes the original message m from the downloaded data and then possibly execute further actions based on the recovered information.

2.3. Uploading

1. Alice chooses a session key k_m and computes the encryption of the message m, i.e.: $c_m := E_{k_m}(m)$
2. Alice chooses ID_m for the message and uploads the pair (ID_m, c_m) to the P2P network
3. Alice computes the vector

   $$v = (E_{k_i}(k_m), \ldots, E_{k_n}(k_m))$$

   (i.e. she encrypts the session key with every symmetric key) and splits it into shares $v_1, \ldots, v_n$ according to a t-threshold secret sharing scheme (this means that v can be recovered from every subset of $\{v_1, \ldots, v_n\}$ of cardinality at least t)
4. Alice gives an ID for every package containing a share with integrity protection: $P_i = (ID_i, v_i, Sign_A(ID_i, v_i))$
5. Alice sends the package P_i and the set of IDs to the friend F_i in the F2F network encrypted with the respective symmetric key, i.e.

   $$c_i := E_{k_i}(P_i,), \{ID_m, ID_1, \ldots, ID_n\}.$$

2.4. Downloading

1. the system generates and sends the alarm message to every friend F_i in the F2F network
2. every friend F_i uploads P_i to the P2P network
3. every friend F_i starts to download the other packages (ID_m, c_m) and P_j for $j \neq i$ from the P2P network

2.5. Message recovery

1. every friend F_i has at most n packages containing the shares. The packages are integrity checked and the corrupted ones are thrown away. If there are at least t correct packages, then every F_i goes to the next step, otherwise they abort.

2. every friend can recover the vector v with the help of the t-threshold secret sharing scheme

3. from v every friend F_i can find his own encrypted part, decrypt the session key k_m with its secret key and then compute the original message m.

3. Security analysis

Theorem 3.1. *If Alice uses a perfect t-threshold secret sharing scheme in the third step of Uploading 2.3 then the system fulfills the Correctness requirement.*

Proof. Let us suppose that there is set of at least t honest friends, for a sake of simplicity we suppose that $F_1, \ldots, F_t$ are honest. Then every friend can download the correct packages $P_1, \ldots, P_t$ in the Downloading phase. Now every friend has the shares $v_1, \ldots, v_t$ from which the vector v can be recovered because of the definition of perfect t_A-threshold secret sharing schemes. □

Theorem 3.2. *If Alice uses a perfect t-threshold secret sharing scheme in the third step of Uploading 2.3 and the encryption scheme E is computationally secure, then the system fulfills the Key Privacy requirement.*

Proof. Here we suppose, that the adversary controls any coalition of users of cardinality less than t, including honest friends, but the latter always follows the steps of the protocol. The adversary can eavesdrop the encrypted message c_m but the assumption ensures Key Privacy in this case. On the other hand, the honest participants neither send nor upload their packages before the Downloading 2.4. Hence, because they are in majority, then at most $\lceil \frac{n+1}{2} \rceil < t$ share v_i is known to the adversary claiming that the knowledge of the adversary is independent of the vector v containing some information related to the session key k_m, which completes the proof. □

Theorem 3.3. *If the encryption scheme E is computationally secure then the system fulfills the Message Privacy requirement.*

Proof. The adversary can gain any information from the encrypted message c_m first. However, it is not possible, because of the assumption. From now on, let us suppose that at least t honest friends upload its package in the second step of Downloading 2.4. We will prove, that the theorem is true even under this rigorous assumption. Then the adversary can compute the vector v containing the encryption of the session key which is computationally secure, hence the message encrypted with this key is computationally secure as well. □

Remark 3.4. Malicious friends are not able to fail the Correctness by jamming with fake packages, because they are not able to forge the digital signature of Alice.

Remark 3.5. Both of the Privacy requirements can be strengthen to ensure unconditional security if $E(.)$ is an unconditionally secure encryption scheme (e.g. one-time pad). Although, such kind of paranoid system can be implemented as well, its far from practical.

4. Implementation details

4.1. 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. The F2F and P2P layers provides a simple socket based interface to the underlying friend-to-friend and peer-to-peer network.

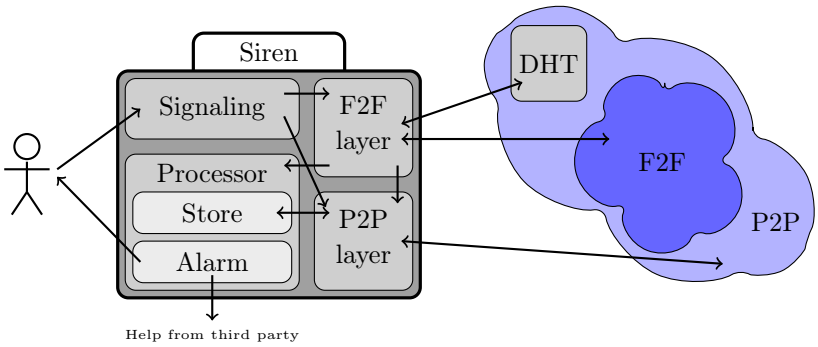

Figure 1: Modules of the Siren application

4.1.1. Signaling

The Signaling module interacts with the user through a graphical interface and sends heartbeat messages or a panic signal if requested. The heartbeat messages can be automatic (timed) or manual, requiring user interaction. This module is responsible for distributing the messages between the friends as specified in 2.3.

4.1.2. Processor

The Processor module is responsible for processing messages from other nodes in the network. It handles the message reconstruction stage as described in 2.5.

4.1.3. P2P

The peer-to-peer layer stores the addresses of all the participants of the friend-to-friend network and provides an interface for sending and receiving messages through a socket based channel.

4.2. 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 (which are calculated from the public keys). The messages sent through this layer are automatically encrypted using the appropriate keys.

The calculation of these keys is done in two step. Firstly when we first start our application we generate a RSA keypair to sign messages and identify ourself [1] and send/receive invites. When we know a friend's identifier we check out his address from the DHT and start the Diffie–Hellman key exchange. At the last phase of the key exchange we receive the public key of our friend with some data (like name, etc.) and calculate our 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.

4.2.1. Communcation between nodes

For two nodes to be able to communicate through the underlying network, first their network addresses has to be resolved. 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) tuple. These pairs are identified by the SHA-256 hash of the public key for the node.

The F2F layer maintains a cache of these (id, IP:port) pairs to speed up connection requests. Between two nodes, if only one node's network address changes, this node can notify the other of its new address. The DHT only needs to be consulted if the node identifier is not found in the cache or if both node's addresses simultaneously change.

The nodes must however continously 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 asymmetrically 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.

[1] We use a hash calculated from the public key as identifier for example in the distributed hash table.

4.2.2. 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 our friends in the F2F network, which contains the hash our 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 exchange. We can add new friends to the contact list by sending a special invite message which can take the form of an URL or a QR code. The invite contains the hash of our public key and an initial network address both of which are signed by our private key. This prevents forgeries and guarantees that the invite is sent by us.

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). The second author was partially supported by the grant OTKA PD-100712.

References

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

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

[3] 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).

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

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

[6] Loewenstern, A., Norberg, A., BitTorrent BEP0005: DHT Protocol `http://www.bittorrent.org/beps/bep_0005.html`