The SKEMASNET project

Session Key Management in a Spontaneous Network

CS642 - Distributed Systems

PARK Hyunho - 20094045
RICCIARDI Gianni M. - 20096093
ALAUZET Pierre - 20096699

November 30th, 2009

HGP Team

Background

The **Skemasnet** project goal is to find an efficient way to merge several private networks in terms of number of exchanged messages and their size. This protocol aims to leverage pre-existing secure networks to share new session keys. We are considering scenarios such as meetings or multi-player games on handled devices, where some tens of users, divided into two independent groups, decide to merge their spontaneous networks (e.g. to share same documents in an enlarged meeting or to play the same match). Users are almost always in visual contact one to each other.

Usually in this kind of context *Group Key Agreement* (GKA) is used to create and share a new session key when needed. We surveyed several papers to understand how it works and to compare it to our idea.

Merging networks

We are assuming the presence of a membership management system, so each member has the list of all other users in the same spontaneous network. Moreover, during the establishing phase of each network, as soon as a user join it taking the session key from a connected user (using a *secure side channel*, e.g. the infrared ports on their laptops or portable devices), he/she broadcasts his/her public key to all users in the network; in this way, once the spontaneous network is established, each user has the public key of all other users in the same network.

In case of merging networks, one user is chosen as a *leader* in each network in order to manage the creation of the new key; all users, after a *social* agreement, select the leader on the users list: when the leader receives a signed *election message* from each user, he assumes the role of leader and sends a signed *confirmation message* to all users. From now on all other users will wait for a new key from the leader, sent as a message they can check using his/her public key.

The two leaders meet face to face, create the new key and share it using a *secure side channel*. Once the new key is available, each leader sends it to all users belonging to his/her former network, using the latter and signing the message containing the new key. In the message containing the new session key, all public keys belonging to users of the other network are also attached.

Using the new key a new common spontaneous network is established, and users from both original networks can communicate one to each other. In order to get the new key an attacker should have one of the keys used to establish one of the merging networks.

Since we are not using a *Public Key Infrastructure* (PKI) with a common *CA* in our scenario (considering it not a suitable requirement for our cases of use), it is not possible to check the identity of

each user in a trustful way. Our idea is to leverage human interaction in order to ensure a level of security for membership management that is acceptable for applications running on top of this kind of networks.

Each host owning the new key is able to share it again using the aforementioned *secure side channel*: in this way new users can join the network in the usual way (i.e. obtaining the session key from a connected user through the use of the side channel and broadcasting their own public key).

We were considering 2 cases of merging networks:
1. **Merging of networks A & B and creation of a new network**
2. **Group B joins the group A**

As an update and revision, we will focus on the case 1.

Case 1: *Merging of networks A & B and creation of a new network*

Initial situation	
After merging	
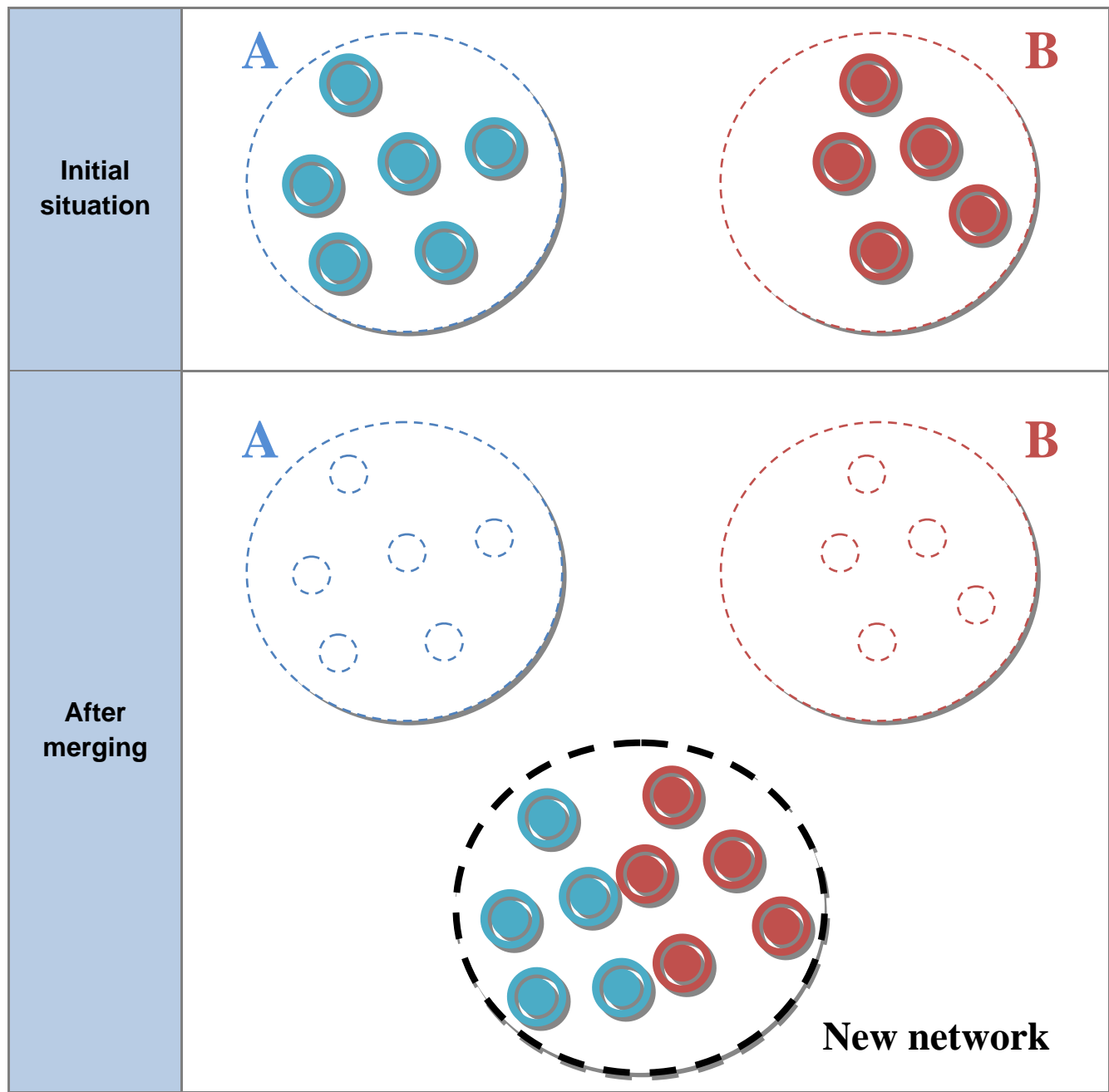
Let us give the scenario we want to follow for the merging of two networks:
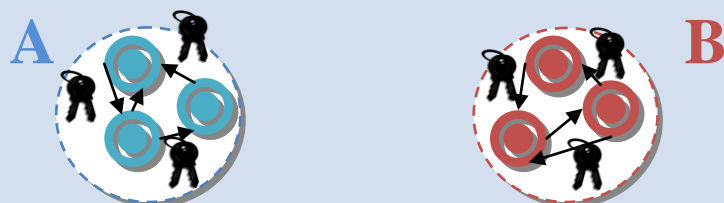
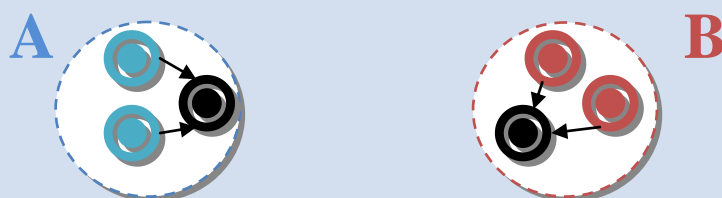|	
Preface	*Decision making comes from human interaction.* *Two groups meet and decide to merge their networks and then they choose two leaders (one per network).*

Initial phase

Thanks to the joining procedure, each user has the public key 🗝 of all other users in the same network.
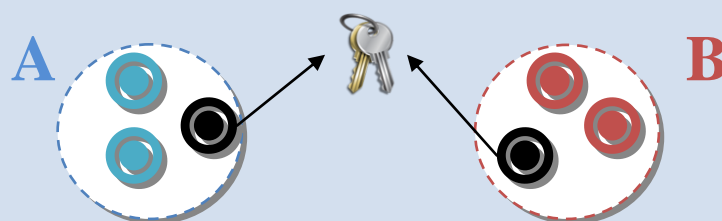

All users of each network, after a *social* agreement, select the leader on the users list.

When the leader receives a signed *election message* from each user, he assumes the role of leader and sends a signed *confirmation message* to all users.
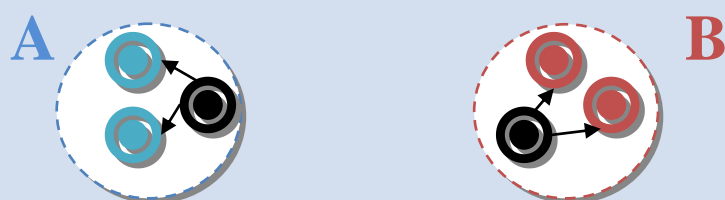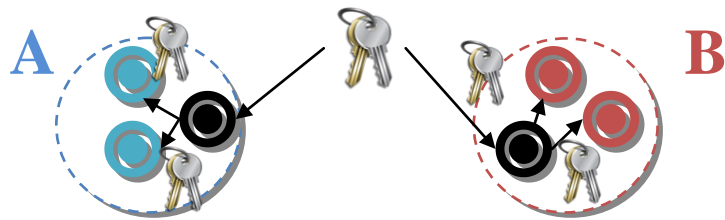

The two leaders meet **face to face** and share a new session key 🗝 using a *secure side channel*.
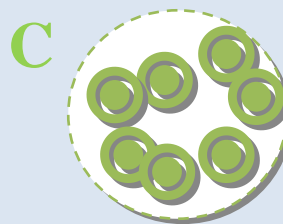
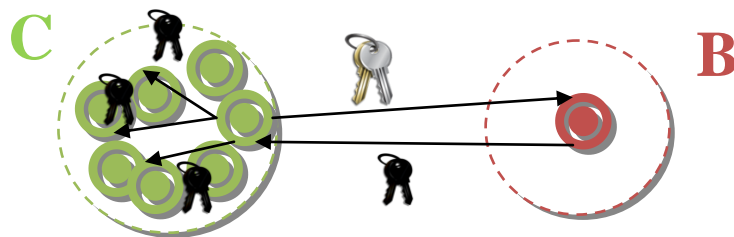|	
Propagation phase	Each leader propagates the new key 🔑 to his/her members of pre-existing network through the network itself using a signed message.
Creation & communication phase	Users from both original networks can communicate one to each other.
Joining phase	If a new node requests to join the new network, it is performed as a common joining process. Each host owning the new key is able to share it again. Public key 🔑 of the new user is sent to the connected user who broadcasts it to all other users; the session key 🔑 is delivered to joining user. The new node joined the network and communicates with the others.

Leaving users

When a user leaves the network and there is the need to allow the access to the network itself only to remaining users, a new session key has to be generated: a leader is elected with the same procedure used for merging networks; the leader generates the new key and send it to all other users (of course not considering the leaving user) encrypting it with their public key. Using the new session key a new spontaneous network is established without the leaving user being able to access it.

Preface

Decision making comes from human interaction.
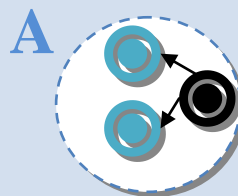*A user decides to leave the network.*
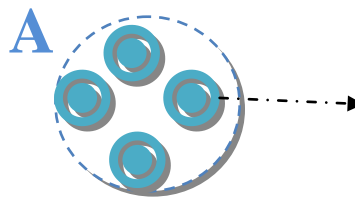
All users of each network, after a *social* agreement, select the leader on the users list.
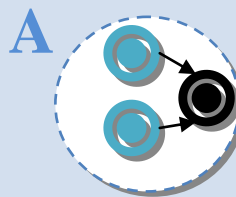
Initial phase

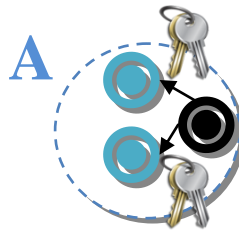When the leader receives a signed *election message* from each user, he assumes the role of leader and sends a signed *confirmation message* to all users.
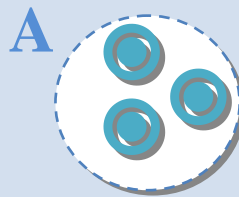
Propagation phase

The leader creates the new key and sends it to all other users, encrypting it with their public keys.

Communication phase

Nodes can now communicate once again, in a new secure network that the previous node cannot see anymore.

Comparison to GKA

Group Key Agreement (GKA) is a mechanism to create and share a common session key for a group of users without the use of a *secure side channel.* Each member is involved in the creation of a new key and provide a public contribution.

However it requires to create a new session key when a change in the network occurs: when a user joins, when a user leaves and in the case of merging networks.

For the creation and the distribution of a new session key 2N-1 messages are required to be exchanged to elect a leader, and 2N-1 messages to create and share a new session key (N to collect public contributes and N to deliver it) and the size of each message used to deliver a session key is

SizeOfSessionKey * 2 * N

Instead in our idea when a user join a message (of SizeOfSessionKey bytes) with the session key is exchanged on the side channel and, once the user is on the network, N messages (with a size depending on the size of public keys) are required to send his/her public key to all other users. In the case of a leaving user we need 2(N-1) messages to elect the leader (N-1 election messages from users to the future leader, and N-1 confirmation messages from the elected leader to all the users) and N-1 messages to deliver the new key to all users (SizeOfSessionKey). In the case of merging networks, we need 2(N-1) to elect the leader, as above, and N-2 messages (total number of users after the merger excluding the two leaders) to share the new key: the size of the latter is

SizeOfSessionKey + max(N1, N2)*SizeOfPubKey

where N1 and N2 are the respective numbers of users in the merging networks.

Status of the implementation so far

In order to implement our solution and demonstrate our algorithm, we will use the *ns-2* simulator. Written in C++ and Tcl, the <u>N</u>etwork <u>S</u>imulation version <u>2</u> is popularly used in the simulation of routing and multicast protocols, among others, and is heavily used in **ad-hoc networking research**. Ns supports an array of popular network protocols, offering **simulation results** for wired and wireless networks alike. It can be also used as limited-functionality **network emulator**. One advantage of using ns-2 is that the result will be visible and understandable.

Thanks to ns-2, we will be able to:
1. **Implement** our *Skemasnet* **algorithm**
2. **Emulate** a bunch of **spontaneous networks**
3. **Simulate** a **merging** between 2 networks, the **joining** and the **leaving** of several nodes
4. **Compare** our implementation with **GKA** in term of **number & size of exchanging messages**

Up to know we implemented the *joining*, *leaving* and *merging* phase for the GKA protocol. Only the election phase is omitted as we assume that the node that receives requests is the leader.

Since we are focusing about number and size of messages, we are not implementing encryption algorithm.

Current implementation does not reflect the overhead of message size.

What still we have to do is:
1. Implement the election phase in GKA
2. Implement all the *Skemasnet* algorithm phases
3. Emulate a bunch of spontaneous networks
5. Simulate a merging between 2 networks, the joining and the leaving of several nodes
6. Compare our implementation with GKA in term of number & size of exchanging messages

References

[1] Johann Van Der Merwe, Dawoud Dawoud, and Stephen Mcdonald
   A Survey on Peer-to-Peer Key Management for Mobile Ad Hoc Networks

[2] Dirk Balfanz, D. K. Smetters, Paul Stewart and H. Chi Wong
   Talking To Strangers: Authentication in Ad-Hoc Wireless Networks

[3] Michael Steiner, Gene Tsudik, Michael Waidner
   Diffie-Hellman Key Distribution Extended to Group Communication

[4] Daniel Augot, Raghav Bhaskar, Valerie Issarny and Daniele Sacchetti
   An Efficient Group Key Agreement Protocol for Ad hoc Networks