

The SKEMASNET project

Session Key Management in a Spontaneous Network

CS642 - Distributed Systems

PARK Hyunho - 20094045

RICCIARDI Gianni M. - 20096093

ALAUZET Pierre - 20096699

December 16th, 2009

HGP Team

Introduction

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 handheld 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.

Related works

A *Group Key Agreement* (GKA) [4] 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 provides 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.

Since in our design a so called *secure side channel* is necessary, some works about the topic have been considered. The choice of a secure side channel depends of course on features available on devices used to establish and access spontaneous networks we are considering. Nowadays we have several examples of these kinds of channels: the most common one is the IrDA (Infrared Data Association) port present on almost all laptops, netbooks and mobile phones on the market. Other interesting methods to exchange some key material have been described in the literature as well.

In [6] the authors show the use of *location-limited channels* for such a purpose, experimenting in their prototype with audio, infrared and contact-based channels.

In [7] some authentication methods based on data from physical channels are showed: shaking devices together and compare data from accelerometers (to derive common crypto material), using spatial reference or a visible laser, etc.

A comparative survey of different methods to pair devices through the use of side channels is presented in [8]: some examples are use of audio subsystems, photo cameras, vibration etc.

Another option is to make users compare short and simple audiovisual patterns, or use an auxiliary device (e.g. a personal camera phone) to reach this goal, as described in [9].

Different kinds of channels have a different impact on the set-up of the network, since each of them requires a particular interaction to users. So the network establishing can be more or less *explicit* (and

require a shorter or longer time) according to the ease of use of the chosen side channel.

Design specifications

1. 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 signed 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.

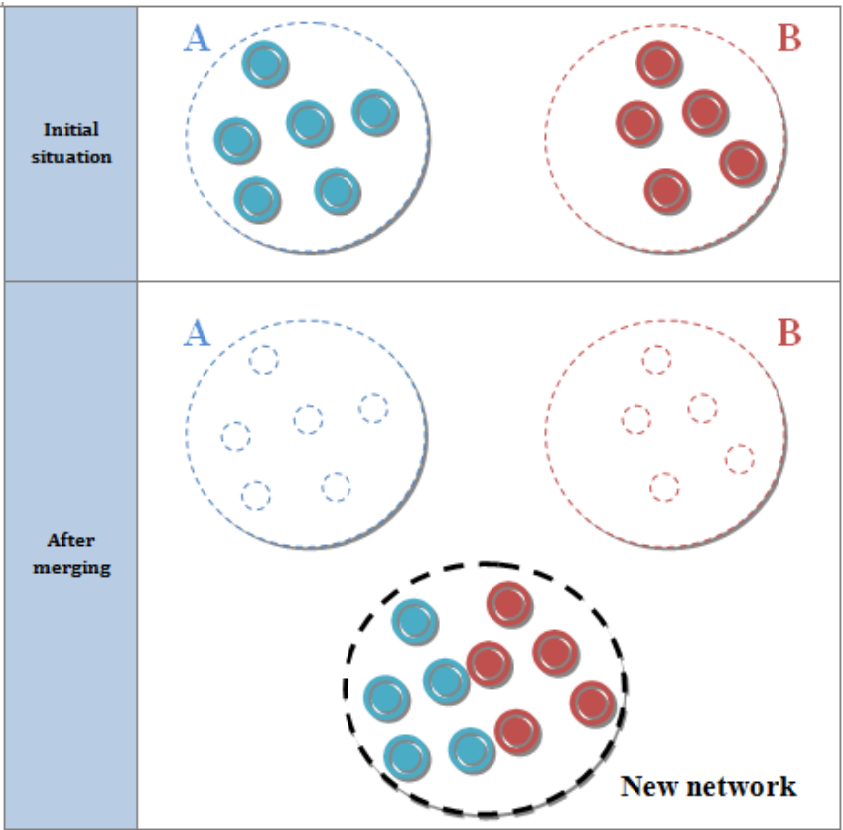
Each host owning the new key is able to share it again using the aforementioned *secure side channel*: in this way new users can join the network in the usual way (i.e. obtaining the session key from a connected user through the use of the side channel and broadcasting their own public key).

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

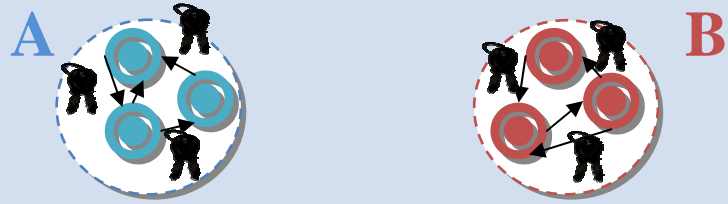


Let us give the scenario we want to follow for the merging of two networks:

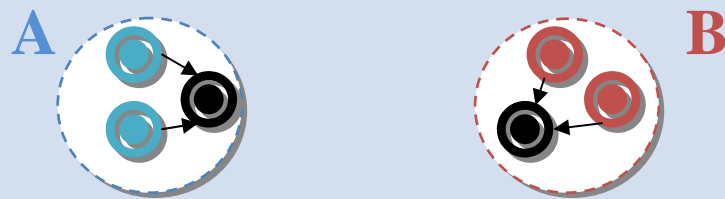
*Decision making comes from human interaction.
Two groups meet and decide to merge their networks and then they choose two leaders (one per network).*

Preface

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

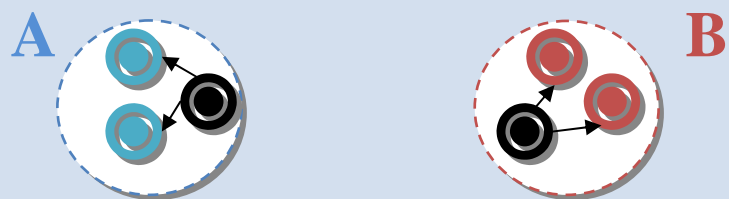



All users of each network, after a *social* agreement, select the leader on the users list.

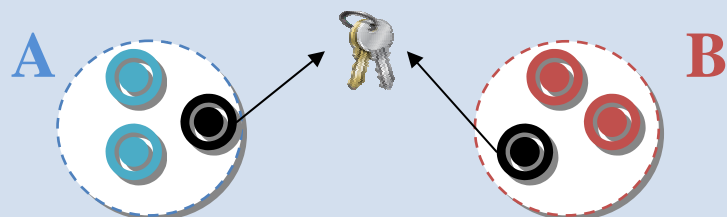


Initial phase

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.

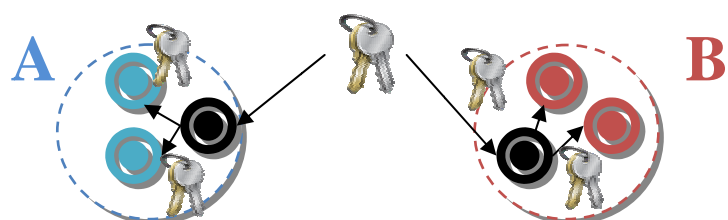


The two leaders meet **face to face** and share a new session key  using a *secure side channel*.



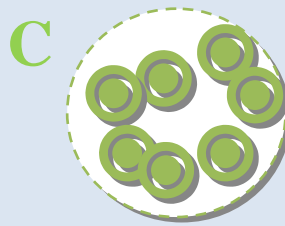
Each leader propagates the new key  to his/her members of pre-existing network through the network itself using a signed message.

Propagation phase




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 able to share it


Public key  of the connected user who

users; the session joining user.



B

new user is sent to the broadcasts it to all other

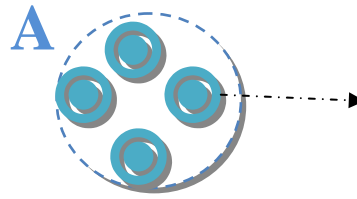
key  is delivered to

The new node joined the network and communicates with the others.

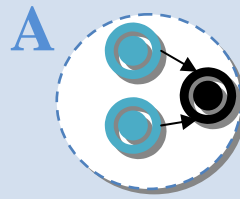
2. Leaving users

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.

A user decides to leave the network.

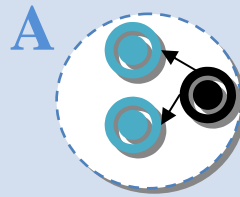



All users of each network, after a *social* agreement, select the leader on the users list.



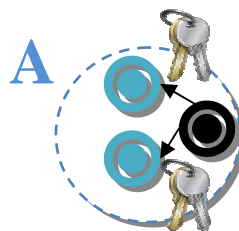
Initial phase

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.



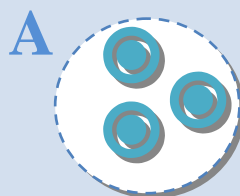
The leader creates the new key  and sends it to all other users, encrypting it with their public keys.

Propagation phase



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

Communication phase



3. Comparison to GKA

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

$$\text{SizeOfSessionKey} * 2 * N$$

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

$$\text{SizeOfSessionKey} + \max(N1, N2) * \text{SizeOfPubKey}$$

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

To summarize, let us compare the

1. Number of exchanged messages:

	GKA	Skemasnet
Merging Networks	$2 * \max(N1, N2) + (N-1)$	$3(N-1)$
Leaving User	$2(N-1) + (N-1)$	$2(N-1) + (N-1)$
Joining User	$2(N-1) + (N-1)$	$N-1$

Figure 1: Comparisons between KGA & Skemasnet in term of size of messages

2. Size of exchanged messages to deliver a new session key:

GKA	$\text{SizeOfSessionKey} * 2 * N$
Skemasnet	$\text{SizeOfSessionKey} + \max(N1, N2) * \text{SizeOfPubKey}$

Figure 2: Comparisons between GKA & Skemasnet in term of number of messages

Implementation

In order to implement our solution and demonstrate our algorithm, we used the *ns-2* simulator. Written in C++ with an OTcl simulation interface, the *Network Simulator version 2* 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, our development consisted in

1. Implement our *Skemasnet* and the *GKA* algorithm
2. Simulate 3 scenarios: merging between 2 networks, the joining and the leaving of a node
3. Compare our implementation with *GKA* in term of number and size of exchanging messages

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

We have implemented *GKA* and *Skemasnet* algorithms in C++ and simulate for 3 scenarios (up to 5 tests per scenario). Concerning the merging of two networks and the joining of a node into a network, we considered between 10 and 50 nodes per network. Concerning the leave of a node from a network, we executed some tests with networks up to 50 nodes.

Results

We show our results using diagrams measuring the overhead in networks using *GKA* and *Skemasnet* algorithms. We compare *GKA* and *Skemasnet* results of the 3 different scenarios taking in account:

1. The number of exchanged messages versus number of nodes
2. Total size of exchanged messages versus number of nodes

1. The number of exchanged messages versus number of nodes

The results of this part will show the number of exchanged messages between nodes from a network, using *Skemasnet* and *GKA* algorithms. We will add more and more nodes at each test and display diagrams of results:

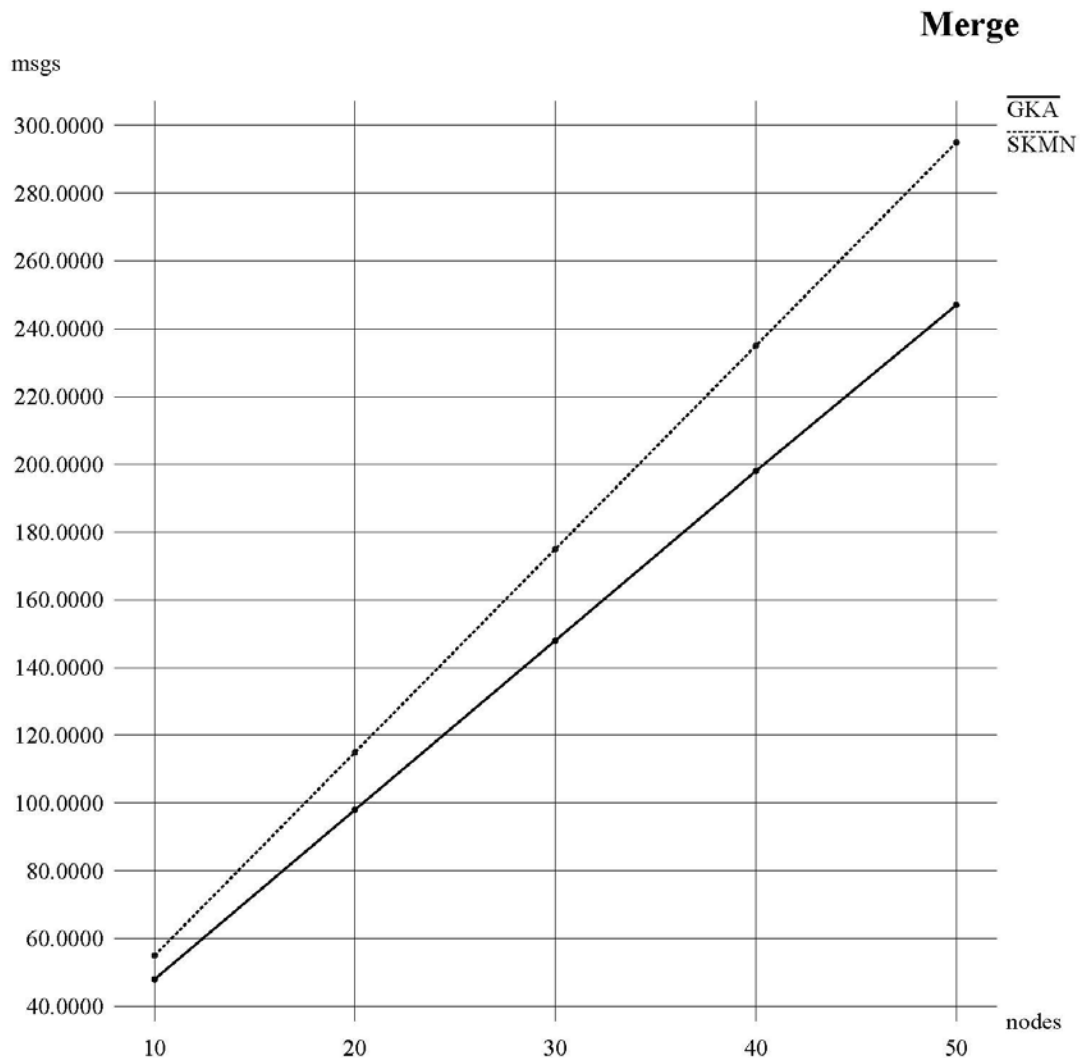


Figure 3: Number of messages versus number of nodes in GKA & Skemasnet when merging networks

The graph shows that GKA needs less exchanged messages than SKMN. But we would like to specify that we implemented the GKA merge phase as a GKA joining phase, as suggested in [4].

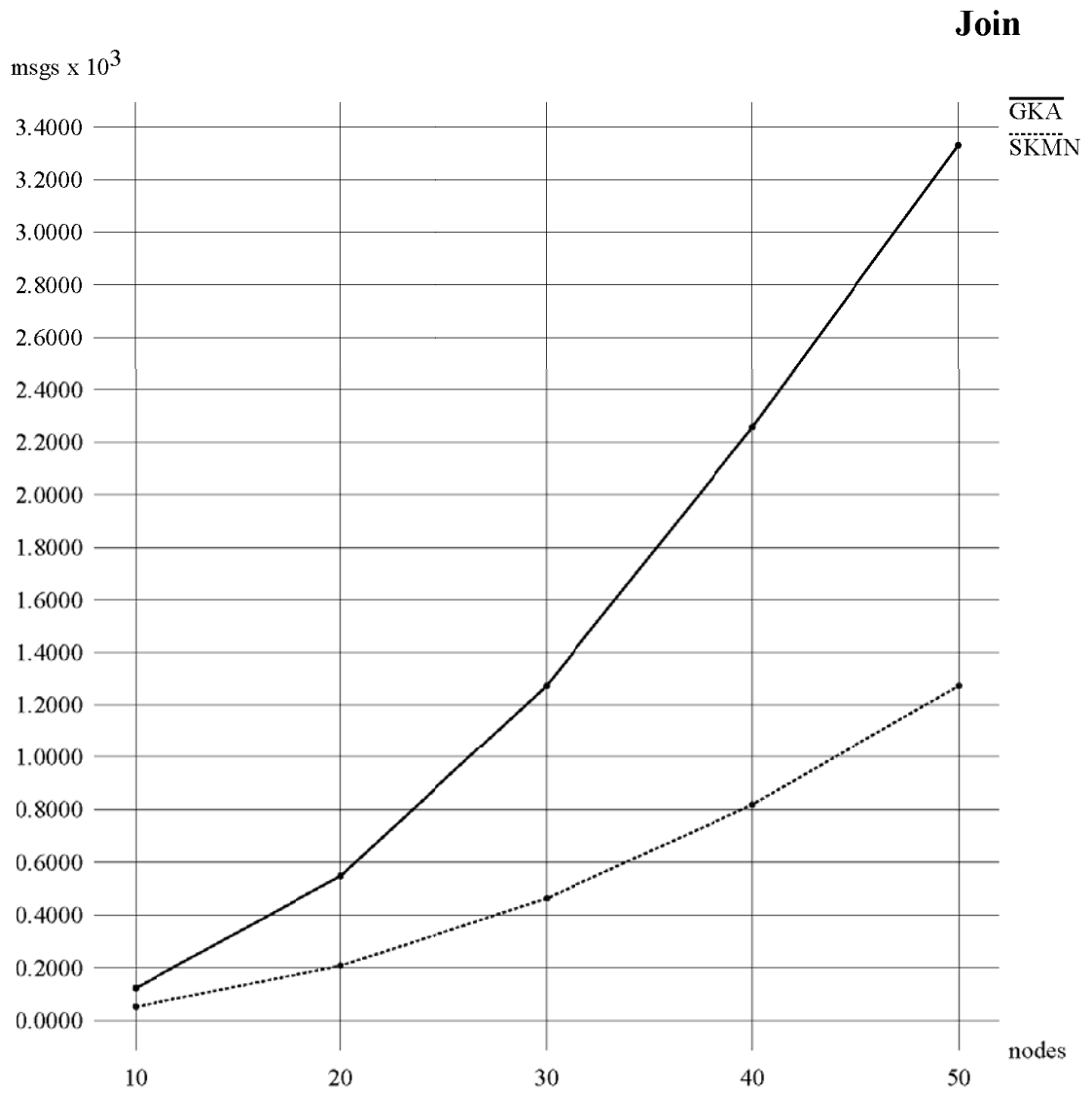


Figure 4: Number of messages versus number of nodes in GKA & Skemasnet when a node joins a network

In the case of joining, *Skemasnet* provides better results than *GKA*, as in our design we do not create a new session key every time a user joins a network.

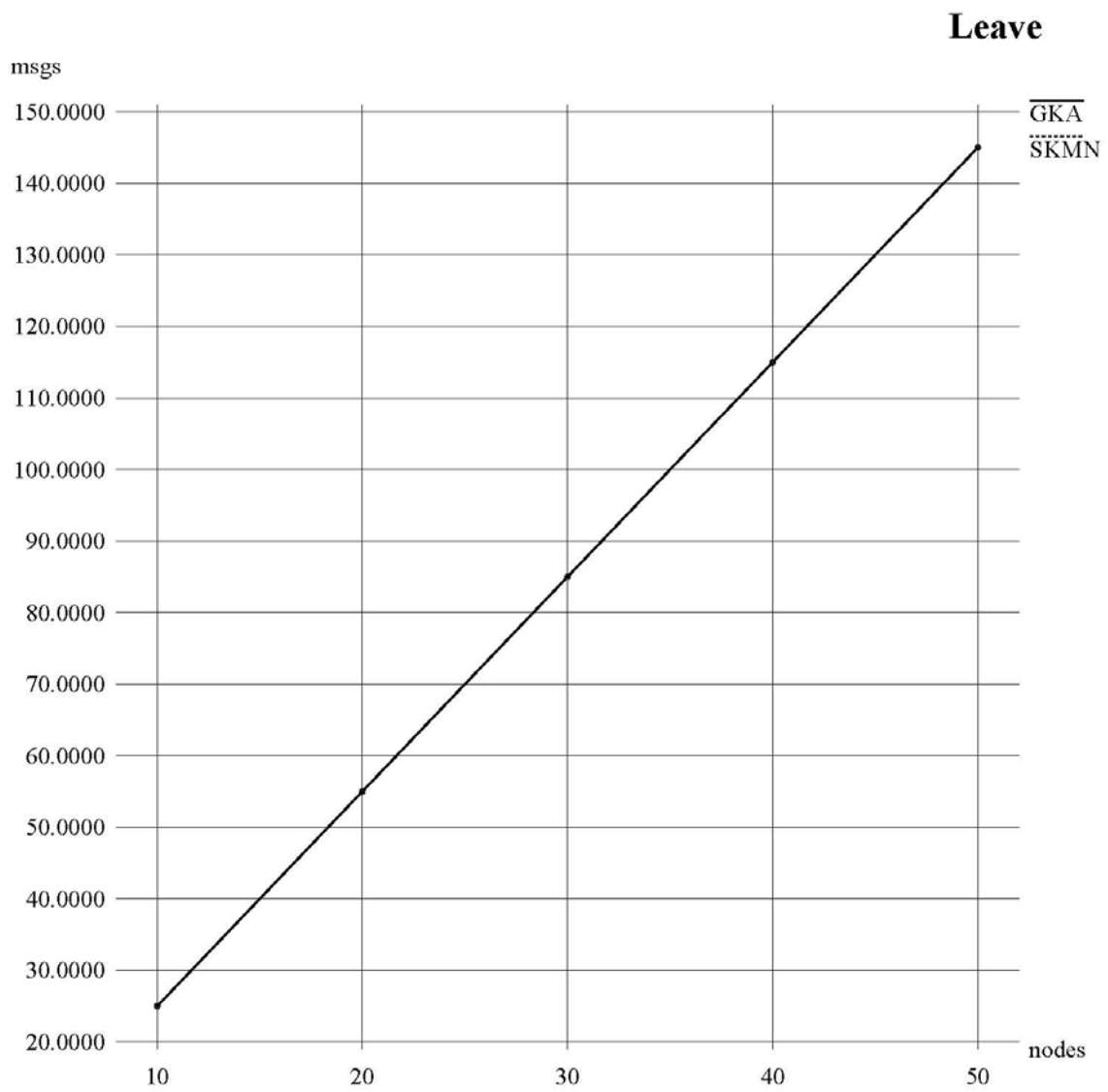


Figure 5: Number of messages versus number of nodes in GKA & Skemasnet when a node leaves a network

As shown on the graph, the results correspond to our expectation. GKA and SKMN require the same number of messages for leaving nodes.

2. Total size of exchanged messages versus number of nodes

We compare the size of messages exchanged using *Skemasnet* and *GKA* algorithms in order to have an idea of the **network usage**.

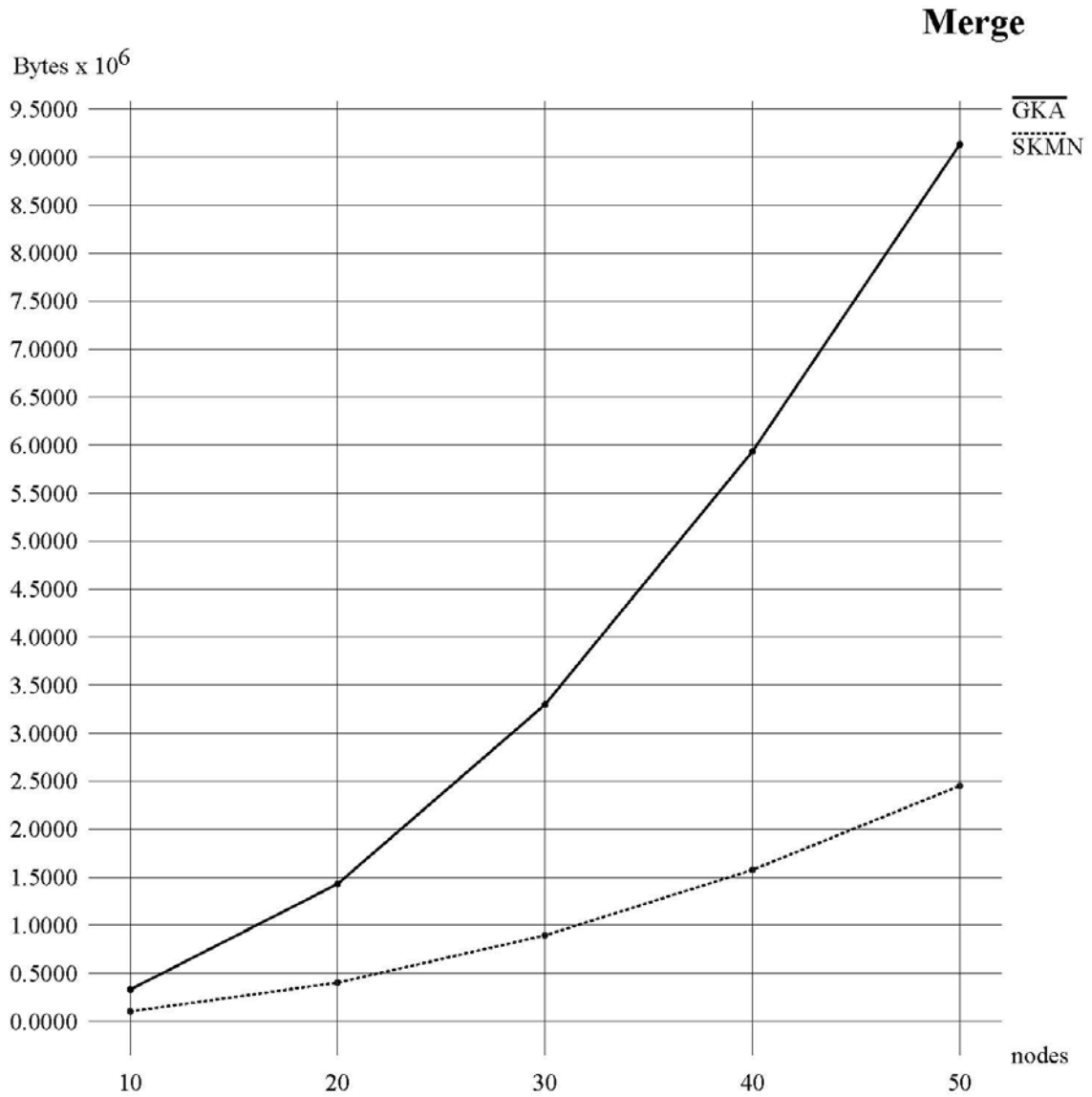


Figure 6: Total size of messages versus number of nodes in GKA & Skemasnet when merging networks

In terms of the number of exchanged messages for merging two networks into a single network, GKA requires fewer messages than SKMN. However, in terms of total size of exchanged messages for merging, GKA requires much more than SKMN. GKA needs bigger sized messages in order to create and share a common session key.

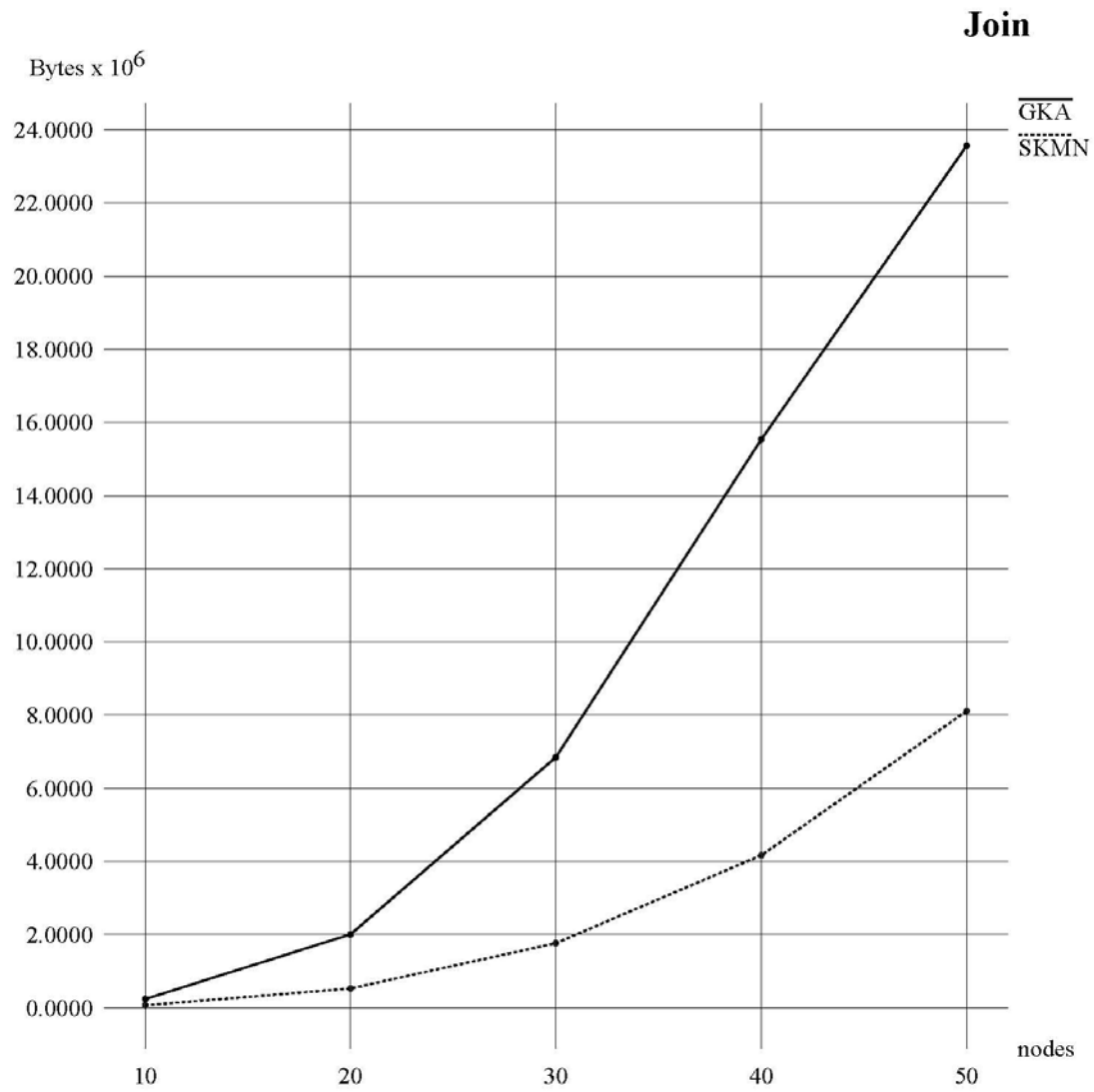


Figure 7: Total size of messages versus number of nodes in GKA & Skemasnet when a node joins a network

Similar to previous results, even in the case of joining phase, SKMN requires smaller messages than GKA.

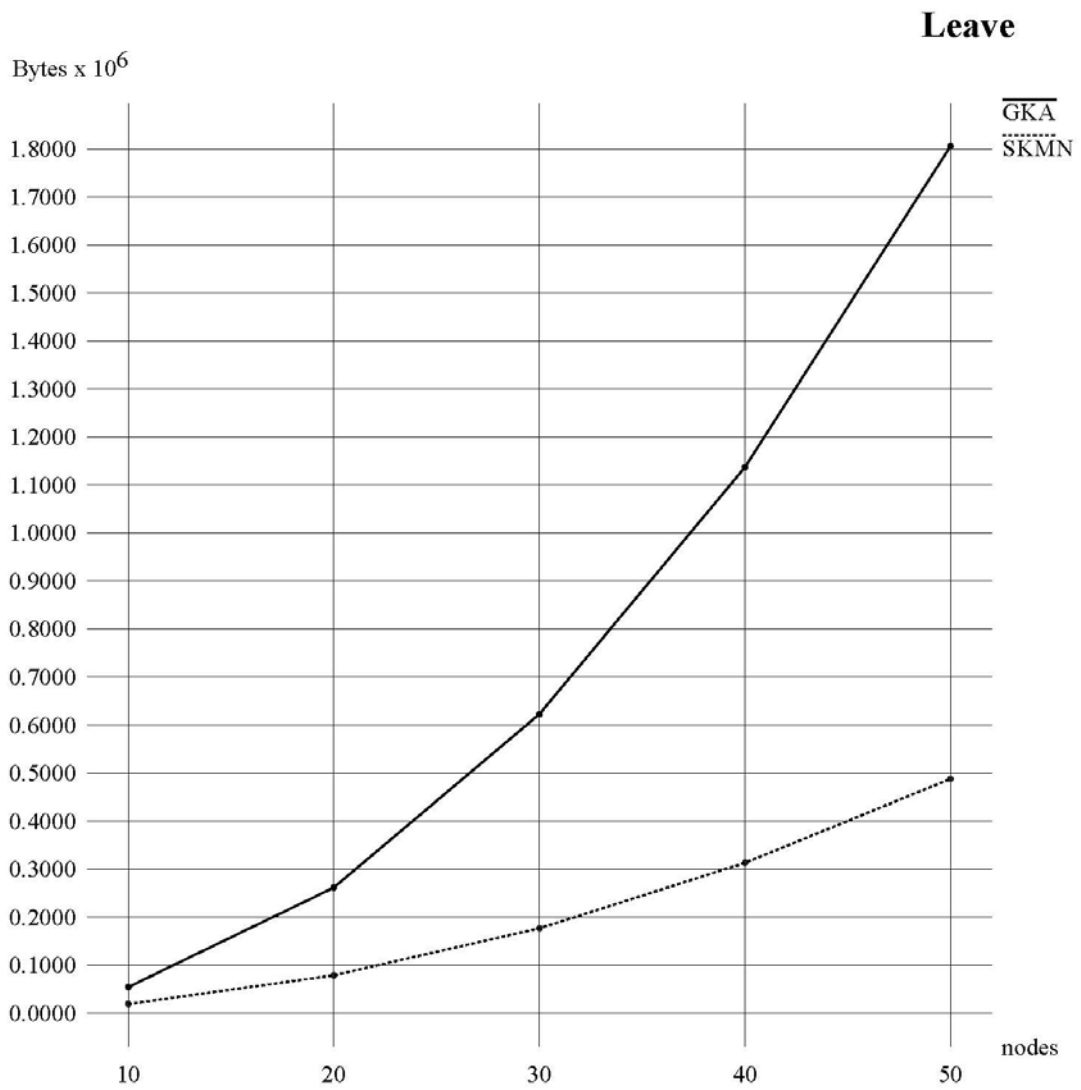


Figure 8: Total size of messages versus number of nodes in GKA & Skemasnet when a node leaves a network

Similar to previous results, even in the case of leaving phase, SKMN requires smaller messages than GKA.

Conclusion

We achieved positive results with *Skemasnet* algorithm concerning number and size of messages. *Skemasnet* does use smaller messages and, in the joining phase, fewer messages than GKA algorithm.

Skemasnet algorithm cannot replace a strong and powerful algorithm like GKA as it may just be used in a special context. *Skemasnet* may be applied in small networks with some tens of nodes, as it leverages social interaction among users to merge networks.

During this work, we did not consider the encryption during our simulations. So a further investigation can be devoted to the analysis of its computation complexity, especially considering that this kind of protocol is run on resource poor devices.

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
- [5] Official website of ns-2 simulator, accessed on November 29th 2009
Ref: <http://www.isi.edu/nsnam/ns/>
- [6] Dirk Balfanz, D. K. Smetters, Paul Stewart, H. Chi Wong
Talking To Strangers: Authentication in Ad-Hoc Wireless Networks
- [7] Mayrhofer Rene, Gellersen Hans
Spontaneous mobile device authentication based on sensor data
Information Security Tech. Report Volume 13, Issue 3 (August 2008)
Ref: <http://dx.doi.org/10.1016/j.istr.2008.10.005>
- [8] Yasir Arfat Malkani, Lachhman Das Dhomeja
Secure Device Association for Ad Hoc and Ubiquitous Computing Environments
2009 International Conference on Emerging Technologies
- [9] Nitesh Saxena, Md. Borhan Uddin, Jonathan Voris
Universal Device Pairing using an Auxiliary Device
CM International Conference Proceeding Series; Vol. 337
Proceedings of the 4th symposium on Usable privacy and security
Ref: <http://doi.acm.org/10.1145/1408664.1408672>