SKiMPy: A Simple Key Management Protocol for MANETs in Emergency and Rescue Operations

Matija Pužar[1], Jon Andersson[2], Thomas Plagemann[1], Yves Roudier[3]

[1] Department of Informatics, University of Oslo, Norway
{matija, plageman}@ifi.uio.no
[2] Thales Communications, Norway
jon.andersson@no.thalesgroup.com
[3] Institut Eurécom, France
yves.roudier@eurecom.fr

Abstract. Mobile ad-hoc networks (MANETs) can provide the technical platform for efficient information sharing in emergency and rescue operations. It is important in such operations to prevent eavesdropping, because some the data present on the scene is highly confidential, and to prevent induction of false information. The latter is one of the main threats to a network and could easily lead to network disruption and wrong management decisions. This paper presents a simple and efficient key management protocol, called SKiMPy. SKiMPy allows devices carried by the rescue personnel to agree on a symmetric shared key, used primarily to establish a protected network infrastructure. The key can be used to ensure confidentiality of the data as well. The protocol is designed and optimized for the high dynamicity and density of nodes present in such a scenario. The use of preinstalled certificates mirrors the organized structure of entities involved, and provides an efficient basis for authentication. We have implemented SKiMPy as a plugin for the Optimized Link State Routing Protocol (OLSR). Our evaluation results show that SKiMPy scales linearly with the number of nodes in worst case scenarios.

1 Introduction

Efficient collaboration between rescue personnel from different organizations is a mission critical element for a successful operation in emergency and rescue situations. There are two central requirements for efficient collaboration, the incentive to collaborate, which is naturally given for rescue personnel, and the ability to efficiently communicate and share information. Mobile ad-hoc networks (MANETs) can provide the technical platform for efficient information sharing in such scenarios, if the rescue personnel is carrying and using mobile computing devices with wireless network interfaces.

Wireless communication needs to be protected to prevent eavesdropping. The data involved should not be available to any third parties, for neither publication or malicious actions. Another important requirement is to prevent inducing of false data. At

This work has been funded by the Norwegian Research Council in the IKT-2010 Program, Project Nr. 152929/431. It has been also partly supported by the European Union under the E-Next SATIN-EDRF project.

the application layer this might for example lead to wrong management decisions. At the network layer it has been shown that a very few percent of misbehaving nodes easily can lead to network disruption and partitioning [17]. In both cases, efficiency of the rescue operation will be drastically reduced and might ultimately cause loss of human lives. In order to prevent such a disaster, all data traffic should be protected, allowing only authorized nodes access to the data. Given that devices carried by the rescue personnel will mostly have limited resources, any security scheme based solemnly on asymmetric cryptography will be too costly in terms of computing power, speed and battery consumption. Therefore, the use of symmetric encryption with shared keys is preferable for MANETs in emergency and rescue scenarios. Agreeing on a shared key in a highly dynamic and infrastructure-less MANET is a non-trivial problem and requires establishing trust relations between all devices. It is important for emergency and rescue scenarios that corresponding solutions are simple, efficient, robust, and autonomous. User interactions should be kept at an absolute minimum.

This paper describes a simple key management protocol, called SKiMPy, that can be used to establish a symmetric shared key between the rescue personnel's devices. By this, SKiMPy will set up a secure network infrastructure between authorized nodes, while keeping out unauthorized ones. It may be decided at the application layer whether the established shared key is robust enough for achieving some degree of data confidentiality as well. The basis for this simple and efficient solution is the fact that rescue personnel are members of public organizations with strict, well defined hierarchies. This hierarchy can be mirrored into a certificate structure installed a priori on their devices, i.e., before the accident or disaster actually happens. As a result, it is possible for the nodes during the rescue activity to authenticate each other on a peer-to-peer basis, without need for contacting a centralized server or establishing trust in a distributed approach.

The organization of the paper is as follows. Section 2 gives a detailed description of our protocol. In Section 3 we show some design considerations and respective solutions. Section 4 describes an implementation of the protocol together with evaluation results. In Section 5 we present related work. Finally, conclusion and future work are given in Section 6.

2 Protocol Description

SKiMPy makes use of the existing traffic in the network to trigger key exchange. Periodic routing beacons (HELLO), sent by proactive routing protocols, are such an example. The following two messages are specific to SKiMPy:

- *Authentication Request* (AUTH_REQ): sent by a node after it detects traffic from a node having a key that is *worse* than its own one. The message is used to inform the remote node that the sending node is willing to transfer its key.
- *Authentication Response* (AUTH_RESP): sent by a node, as a result of a received AUTH_REQ message. The message is used to inform the remote party that the node is willing to perform the authentication and receive the remote and *better* key.

The protocol consists of three phases, namely *(I) Neighborhood Discovery*, *(II) Batching* and *(III) Key Exchange*.

During phase I, a node listens to all traffic sent by its immediate neighbors. If it detects a node using a *worse* key (explained in detail in Section 3.2), it will send an *Authentication Request* message to it, saying it is willing to pass on its key. Upon receiving such a message, the other node enters the phase II, waiting for possible other authentication requests before sending a response. This batching period is used for optimization - a node will only perform authentication with the *best* of all neighbors. All the other keys will, due to the transitiveness property of the *better than* relation, at some point get overruled and therefore there is no point in getting them. After the node has chosen its peer, it sends an *Authentication Response* after which its peer initializes the actual authentication procedure, that is, exchange of certificates, establishing a secure tunnel, and finally transfer of the key. The reason for having such a handshake procedure is to ensure that the nodes can indeed communicate. In some standards, such as 802.11b [19], traffic like broadcast messages can be sent on a lower transmitting rate with larger transmission range than data messages. Thus, broadcast messages might reach a remote node and trigger a key exchange, even though the nodes cannot directly exchange data packets.

Figure 1 shows an example of the key exchange between three nodes (*A*, *B* and *C*) and indicates the different phases of the key exchange for node *A*. Node *A* enters phase I when turned on. Nodes *B* and *C* do not directly hear each other's traffic and are only able to communicate through node *A*, once the shared key is fully deployed.

The initial states of the three nodes are as follows: *A* has the key K_A, *B* has K_B and *C* has K_C. In this example, K_C is the *best* key, whereas K_A is the *worst* key.

Phase I:

1. Node *A* is turned on. All nodes send periodic HELLO messages which are part of the routing protocol.
2. *A* receives a HELLO message from *B*, notices a key mismatch, but ignores it because K_A is *worse* than K_B.
3. *A* receives HELLO from *C*, notices a key mismatch, but ignores it because K_A is *worse* than K_C.
4. *B* and *C* receive HELLO from *A*, they both notice they have a *better* key than K_A, and after a random time delay (to prevent traffic collisions), send an AUTH_REQ message to *A***.**

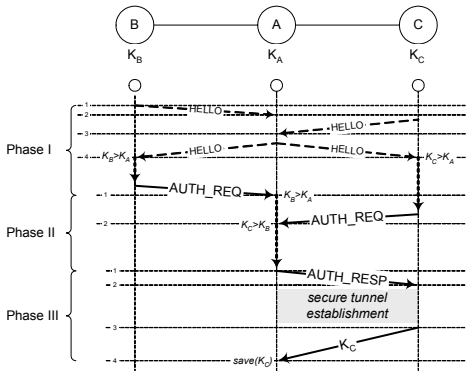

Fig. 1. Message Flow Diagram

Phase II:

1. *A* receives AUTH_REQ from *B* notices that *B* has a *better* key and schedules authentication with *B*. The authentication is to be performed after a certain waiting period, in order to hear if some of the neighbors has an even *better* key.
2. *A* receives AUTH_REQ from *C* as well, sees that *C* has a key *better* than K_B, and therefore decides to perform authentication with *C* instead.

Phase III:

1. *A* sends an AUTH_RESP message to *C*, telling it is ready for the authentication process
2. *C* initiates the authentication procedure with *A*, they exchange and verify certificates; the secure tunnel is established.
3. *C* sends its key K_C to *A* through the secure tunnel.
4. *A* receives the key and saves it locally; the old key K_A is saved in the key repository for eventual later use; *A* sends the new key further, encrypted with K_A.

In the next round, that is, after it hears traffic from node *B* signed with K_B, node *A* will use the same procedure to deliver the new key K_C to node *B*, hence establishing a common shared key in the whole cell.

There are two important parameters which influence the performance of the protocol and therefore have to be chosen carefully. The delays used before sending AUTH_REQ are random, to minimize the possibility of collisions in the case when more nodes react to the same message. On the other hand, the delay from the moment a node receives AUTH_REQ to the moment it chooses to answer with AUTH_RESP is a fixed interval and should be tuned so that it manages to hear as many neighbors as possible within a reasonable time limit. By this, all nodes that have been heard during the waiting period can be efficiently handled in the same batch.

3 Design Considerations

Our protocol is designed for highly dynamic networks, where nodes may appear, disappear and move in an arbitrary manner. Topology changes are inevitable. The key management protocol must have low impact on the available resources, i.e. battery, bandwidth and CPU time. Here, we analyze the different security and performance issues that had to be considered while designing the protocol, as well as respective solutions integrated into SKiMPy.

3.1 Authentication

An important characteristic of an emergency and rescue operation is that the organizations involved (police, fire department, paramedics, etc.) are often well structured, public entities. Before the rescue personnel comes to the disaster scene, all devices are prepared for their tasks. One task in the preparation phase, which we call *a priori*

phase [23], is the installation of valid certificates. The certificates are signed by a commonly trusted authority, such as the ministry of internal affairs, ministry of defense, etc., on the top of the trust chain. This gives nodes the possibility to authenticate each other without need for contacting a third party.

Certificates on the nodes can identify devices, users handling them, or even both. The users would then present their certificate to the device by means of a token, i.e. smartcard. The decision for this does not impact the key management in SKiMPy, but it impacts the way how lost and stolen nodes are handled, i.e., revoking certificates and/or blacklisting of such nodes. We explain this issue later, in Section 3.5.

3.2 Choosing Keys

The main task of SKiMPy is to make sure that all the nodes agree on a shared key. When a node is turned on, it generates a random key with a random ID number. The uniqueness of the key IDs must be ensured by e.g. using the hash value of the key itself as part of the ID, by including the nodes MAC address, etc. The final shared key is always chosen from nodes' initial keys. To achieve this, we introduce the notions of *better* and *worse* keys, together with the relation ">" representing *better than*. There are several possible schemes for deciding which of the keys is *better* or *worse* and all schemes can be equally valid, as long as they cannot cause key exchange loops, are unambiguous and transitive: $(A > B$ and $B > C) \Rightarrow A > C$. The necessary control information, which depends on the scheme chosen, is always sent with the message signature.

We briefly describe two schemes and their advantages and drawbacks.

The first scheme uses arithmetic comparison of two numbers, i.e. the key having a higher or lower ID number, timestamp or a similar parameter, is considered to be *better*. The advantage of this scheme is that it is unambiguous, transitive and easy to implement. In addition, it can be "tweaked" in a way that would prevent a single node to cause re-keying of an already established network cell. For example, if the scheme defines that the lower ID number means a better key, the highest bit of the ID number can be always set to "1" when the node is turned on, and cleared once two nodes merge. Assuming that nodes in a certain area will in most cases pop up independently, this simple and yet efficient method might prevent a lot of unnecessary re-keying traffic. If we use the keys' timestamps instead of the ID numbers, choosing a lower timestamp could imply that the key is older and that more nodes have it already. SKiMPy does not require the clocks of different devices to be synchronized and therefore, the given assumption might not necessarily be true, especially if the key creator's clock was heavily out of sync. One major drawback of the presented scheme is that a small cell (consisting of, for example, 2 nodes) could easily cause re-keying of a much bigger cell (having, for example, 100 nodes), which would be a waste of resources.

The second scheme takes care of this problem by using the number of nodes in each network cell as the decisive factor. The simple rule for this scheme is to always re-key the smaller cell, i.e. the one with the lower number of nodes, thus minimizing resource consumption for the necessary re-keying. The approximate number of nodes can be either retrieved from the routing protocol state information (if, for example, the OLSR routing protocol [7] is used) or maintained at a higher protocol layer, as it is

done in our project. However, if not all of the nodes have exactly the same information (which is to be expected in a dynamic scenario), and for some obscure reason we have more simultaneous merging processes between the same two cells, a key exchange loop may occur. One approach to this problem is to adjust in each node the state information of the number of nodes in its cell, always increasing it when new nodes join, but never decreasing it upon partitioning of the cell.

At the present, we use the first scheme, choosing always a key with a lower ID number. An in-depth study of both schemes and their variations is subject to ongoing and future work.

3.3 Key Distribution

Once a node gets a new key as a result of network merging, the key should be deployed within its previous network cell. There are several ways to achieve this:

- *Proactively* - each node receiving the key immediately forwards it to the others. This approach ensures prompt delivery of the key to all nodes, but it also generates a lot of unnecessary network traffic.
- *Reactively* - when a node receives a key, it does nothing. Only after detecting a message sent by a neighbor and signed with the old key, the node sends the new key further. This approach uses less resources, but it takes more time for the whole cell to get a stable key.
- *Combination* - the first node getting the new key (that is, the node which performed the merge) immediately forwards the key to its one-hop neighbors, since it knows that no other node in its previous cell has it yet. The other nodes do not distribute it right away, but rather when (if) they notice that a node still uses an old key. This approach keeps the number of necessary broadcast messages containing the key at a minimum.

In any of the given cases, the new key is encrypted using the old one before sending, giving all the other nodes the possibility to immediately start using it. The old key is saved for a short period of time, for possible latecomers. This can be done because in this particular case the key change was not performed explicitly for the purpose of preventing traffic analysis attacks.

In our implementation, described in Section 4.1, we use the *combination* approach.

3.4 Key Update

When created, each key has a companion key (called *update key*) used to periodically update it. The update key is never used on traffic that goes onto the network and therefore it is not prone to traffic-analysis attacks. The nodes must periodically update the main key. The new key can be computed using one-way hash functions such as SHA-1 [15] or MD5 [25], ensuring backward secrecy in the case the key gets broken at some stage. In addition to the ID of the key used to sign it, a message contains also the update-number saying how many times the key on the sender-node has been updated. That way, the receiver can easily compute the new key if it notices a mismatch,

which could happen since we can't expect all the nodes to perform the update at exactly the same time. The local update will not take place if the received message has an invalid signature.

3.5 Exclusion of Nodes

Once authenticated, a node is a fully trusted member of the network. This poses the evident problem of how to exclude such a node once the device has been lost or, even worse, stolen by a malicious third party. At the present, exclusion of already authenticated nodes is not solved in SKiMPy and is part of ongoing and future work. Here, we describe some ideas on measures to be taken in order to ensure that such a node stays out of the network.

First, the node's certificate must be revoked, preventing the node from re-authenticating later at some stage. Since there is no central authority, a decision is reached on which node or person can perform the task of revoking certificates. If the certificates contain also additional attributes such as rank or role of the persons (assuming that the certificates do in fact represent persons, not devices), it can be decided that only certain roles/ranks (such as *leader*) can perform revocation and blacklisting. In theory, the leaders' devices might also be stolen, but in practice they should normally be physically well protected. It is important to ensure that the compromised node itself does not revoke and blacklist legitimate ones or, even worse, the whole network.

Next, the node's IP address should be put on a common blacklist. Assuming that IP addresses are bound to the certificates (as presented in e.g. [22]), the nodes would be unable to change their IP address. However, relying on fixed IP addresses might introduce new issues and should be considered carefully. Traffic coming from blacklisted nodes must be discarded at the lowest possible layer and, in case legally signed traffic coming from a blacklisted node is detected, the compromised key must be removed.

Additional methods might be used to ensure that devices cannot be used by unauthorized persons. One such example is a system relying on short range wireless authentication tokens. A token is installed into the personnel's vests or watches, ensuring confidentiality of the data and denying unauthorized access to the devices when they get out of their token's range [8].

3.6 Batching

To save resources as much as possible, our protocol makes the nodes learn about their neighborhood before acting, reducing the number of performed authentications and thus reducing directly CPU and bandwidth consumption. This is possible due to the fact that all nodes directly trust the same certificate authority and, therefore, if a node has been successfully authenticated before and has received the shared secret, we implicitly trust it.

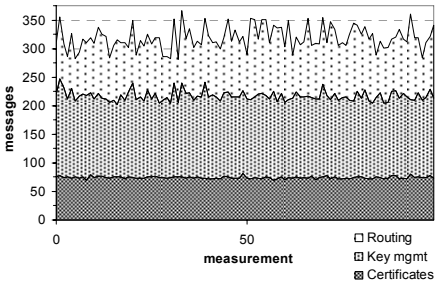

Fig. 2. Traffic analysis of the first, non-optimized protocol implementation

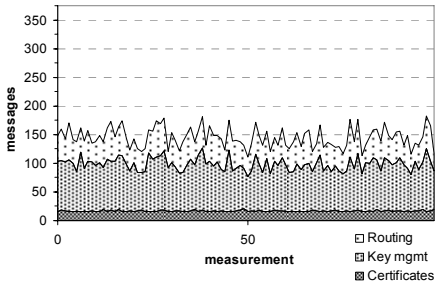

Fig. 3. Results for the same scenario, after introducing the batching process

Emphasis has been put on optimization with regards to number of messages sent out in the air. We measured the number of certificates and key management messages exchanged, and compared these figures to the number of routing messages needed from the moment when the nodes were turned on, up to the moment when a stable shared key was established. To perform these measurements, we used a static, wired test bed with 16 nodes.

Figures 2 and 3 show that introducing neighborhood awareness approximately halved the total number of messages and, proportionally, the time needed to reach a stable state. Moreover, the number of messages carrying certificates, whose size is much larger than other key management messages, has been reduced to approximately 23% of the initial number. The authentication was considered to be done after the exchange of certificates. Therefore, the results shown here are only an approximation, and might be slightly different when an actual authentication algorithm is used.

3.7 Additional Issues

The protocol's goal is to establish a secure network infrastructure. SKiMPy makes it impossible for a misbehaving node to induce a key that has either expired, or that would not have been selected in a normal operation. Such keys will be immediately discarded.

Timeouts are used during the *Key Exchange* phase (explained in Section 2) to ensure that a node does not end up in indefinite wait states or deadlocks as a result of possible link failures. Care must be taken for possible Denial-of-Service attacks in any of these cases.

In the *closing* phase of the rescue operation [23], the keys must be removed to prevent them from being possibly reused afterwards on a different rescue site.

4 Protocol Implementation and Evaluation

4.1 Implementation

Optimized Link State Routing Protocol (OLSR) [7] is a proactive routing protocol for ad-hoc networks which is one of the candidates to be used in our solution for the emergency and rescue operations. The olsr.org OLSR daemon [28] is the implementation we decided to test, since it is portable and expandable by means of loadable plugins. One example of such a plugin, present in the main distribution, is the Secure OLSR plugin [16]. The plugin is used to add signature messages to OLSR traffic, only allowing nodes that possess the correct shared (pre-installed) key to be part of the OLSR routing domain. One important functionality this plugin lacks is a key management protocol. Even though SKiMPy is mainly designed to protect all traffic and not only routing, it is still a good opportunity to test and analyze it in a realistic environment with a real routing protocol.

The key management protocol has been coded directly into the security plugin, although the plans are to make it as a separate one. X.509 certificates [18] and OpenSSL [27] are currently used to perform node authentication.

4.2 Evaluation Results

To facilitate development of this and other protocols, we created an emulation test bed, called NEMAN [24]. Routing daemons run independently, each attached to a different virtual Ethernet device. We use the monitoring channel of the emulator to analyze the keys used by each of the routing daemons. In order to test performance and scalability the protocol, we have made measurements from 2 to 100 nodes, with two very different kinds of scenario: chain and mesh. Figures 4 and 5 show example screenshots taken from the GUI, representing the two different scenarios.

In a chain scenario, the nodes are lined up in a single chain and the distance between all nodes in the chain is such that only the direct neighbors can communicate in a single hop with each other. We consider this to be the worst case scenario still giving full network connectivity. Given that all the nodes have to perform authentication with both their neighbors, this leaves no place for optimization, i.e. batching during the waiting period.

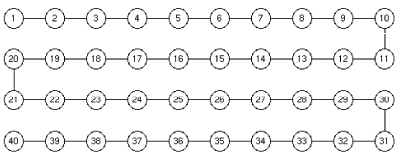

Fig. 4. Example of a chain scenario

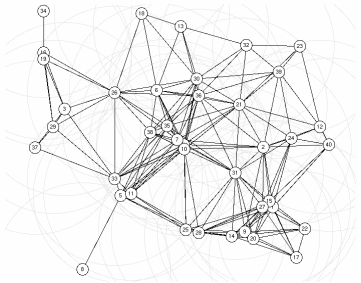

Fig. 5. Example of a mesh scenario

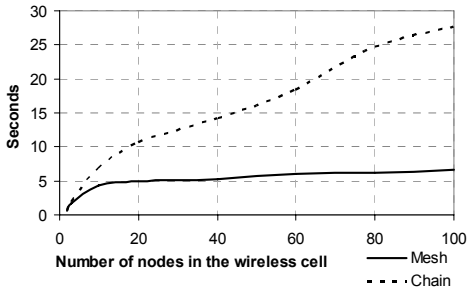

Fig. 6. Time needed to achieve a stable shared key

In a mesh scenario, however, nodes have multiple, randomly scattered neighbors, as it is natural in ad-hoc networks. Having multiple neighbors allows the protocol to exploit the batching phase, reducing traffic and resource consumption.

Ten independent runs were performed for each number of nodes and each scenario. All the nodes were started simultaneously (which we assume is the worst case for our protocol), with a random key and key ID. To be able to meaningfully compare the results, the nodes were static and the density was constant. The delay in the batching period was set to be 1 second, i.e. half of the interval used by OLSR to send HELLO messages.

One important fact that the results on Figure 6 immediately show is that the protocol scales linearly with linear increase of the number of nodes and physical network area accordingly (thus giving the same density of nodes). After approximately 10 nodes, the total time became almost independent on the network size. By the fourth second, most authentications have already been performed and the key distribution process came into place. In some additional measurements, we introduced node movement using the random waypoint mobility model. As long as all of the nodes remained reachable and the density was constant, movement did not induce a notable delay.

We also proved that having multiple neighbors does in fact lower the time necessary to reach a stable state. This scenario gives less deviation as well, which is understandable since in the case of chain there is more fluctuation of keys, nicely seen in the GUI.

5 Related Work

Different authentication schemes are available as a starting point for key management.

Devices can exchange a secret or pre-authentication data through a physical contact or directed infrared link between them [3, 26]. Another way is for the users to compare strings displayed on their devices (a representation of their public key, distance between them, etc. as presented in [9]). Since user interaction in a rescue operation should be kept as minimum, we need a different approach.

Threshold cryptography schemes, such as [20] and [31] require all nodes that are going to perform signatures to carry a share of the group private key. The full signature is acquired by a certain, predefined number of nodes who present partial signatures computed using their shares. These schemes allow a small number of nodes to be compromised and still not to present a threat for the network. However, since we do not know the number of nodes that can be expected at the rescue scene and small partitions might always be present, this approach is not suited for our scenario.

Čapkun et al. [10] present a fully self-organized public-key management system that does not rely on trusted authorities, developed mainly for networks where users can join and leave without any centralized control. This is not applicable to networks used in rescue operations, where only authorized nodes are allowed to participate. In [11], they present a solution similar to ours, explained in Section 3.1, allowing nodes to authenticate each other by means of pre-installed certificates with a common authority. The advantages of such a system are twofold: first, the data in the network is more secure. Second, establishing trust and agreeing on a shared key is much more efficient, i.e., faster and less resources are consumed.

Related key management protocols can be roughly divided into the following three categories [6].

The first one relies on a fixed infrastructure and servers that are always reachable. Since we never know where accidents will happen, we should expect them to happen at places where we cannot rely on the fact that fixed infrastructure will be present.

The next category comprises contributory key agreement protocols, which are not suited for our scenario for several reasons. Such protocols ([1, 5, 12, 29, 30], to name a few) are based on Diffie-Hellman two-party key exchange [13] where all the nodes give their contribution to the final shared key, causing re-keying every time a new node joins or an existing node leaves the group. In an emergency and rescue operation, we can expect nodes to pop up and disappear all the time, often causing network partitioning and merging. Therefore, using contributory protocols would cause a lot of computational and bandwidth costs which cannot be afforded. Besides, most of these protocols rely on some kind of hierarchy (chain, binary tree, etc.) and a group manager to deploy and maintain shared keys. In a highly dynamic scenario this approach would be quite ineffective. Another reason why such protocols are not suited for us, is that in order for the nodes to be able to exchange keys, a fully working routing infrastructure has to be established prior to that. Since the routing protocol is one of the main things we need to protect, this is a major drawback. Asokan and Ginzboorg [2] present a password-based authenticated key exchange system. A weak password is known to every member and it is used by each of them to compute a part of the final shared key. This approach shares some already mentioned drawbacks and introduces new ones which conflict with our scenario and requirements. User interaction is needed and it is assumed that all the members are present when creating the key.

The last category are protocols based on key pre-distribution. The main characteristic of such protocols is that a pair or group of nodes can compute a shared key out of pre-distributed sets of keys present on each node. These sets of keys are either given by a trusted entity before the nodes come to the scene [4, 14, 21], or chosen and managed by the nodes themselves, as it is done in DKPS [6].

SKiMPy is different in the sense that it uses pre-installed certificates to perform direct authentication between two nodes. This makes it more simple and efficient.

6 Conclusion

In this paper, we presented a simple and efficient key management protocol, called SKiMPy, developed and optimized especially for highly dynamic ad-hoc networks. The protocol relies on the fact that there will be an *a priori* phase of rescue and emergency operations, within which certificates will be deployed on rescue personnel's devices. Pre-installed certificates are necessary due to the fact that highly sensitive data may be exchanged between the rescue personnel. The certificates make it possible for the nodes to authenticate each other without need for a third party present on the scene.

We described a proof-of-concept implementation, as well as evaluation results. The results show that SKiMPy performs very well and it scales linearly with the number of nodes. As part of further work we will analyze more in-depth different key selection and distribution schemes, authentication protocols, and fine tune certain protocol parameters, like the delays described in Section 2. Open issues like exclusion of compromised nodes, duplicate key ID numbers, denial of service attacks, etc. are also subject of further investigation.

References

1. Alves-Foss, J., "An Efficient Secure Authenticated Group Key Exchange Algorithm for Large And Dynamic Groups", Proceedings of the 23rd National Information Systems Security Conference, pages 254-266, October 2000
2. Asokan, N., Ginzboorg, P., "Key Agreement in Ad Hoc Networks", Computer Communications, 23:1627-1637, 2000
3. Balfanz, D, Smetters, D. K., Stewart, P, Wong, H. C., "Talking To Strangers: Authentication in Ad-Hoc Wireless Networks", Proceedings of the 9th Annual Network and Distributed System Security Symposium (NDSS'02), San Diego, California, February 2002
4. Blom, R., "An Optimal Class of Symmetric Key Generation System", Advances in Cryptology - Eurocrypt'84, LNCS vol. 209, p. 335-338, 1985
5. Bresson, E., Chevassut, O., Pointcheval, D., "Provably Authenticated Group Diffie-Hellman Key Exchange - The Dynamic Case (Extended Abstract)", Advances in Cryptology - Proceedings of AsiaCrypt 2001, pages 290-309. LNCS, Vol. 2248, 2001
6. Chan, Aldar C-F., "Distributed Symmetric Key Management for Mobile Ad hoc Networks", IEEE Infocom 2004, Hong Kong, March 2004
7. Clausen T., Jacquet P., "Optimized Link State Routing Protocol (OLSR)", RFC 3626, October 2003
8. Corner, Mark D., Noble, Brian D., "Zero-Interaction Authentication", at The 8th Annual International Conference on Mobile Computing and Networking (MobiCom'02), Atlanta, Georgia, September 2002
9. Čagalj, M., Čapkun, S., Hubaux, J.-P., "Key agreement in peer-to-peer wireless networks", to appear in Proceedings of the IEEE (Specials Issue on Security and Cryptography), 2005
10. Čapkun, S., Buttyán, L., Hubaux, J.-P., "Self-Organized Public-Key Management for Mobile Ad Hoc Networks", IEEE Transactions on Mobile Computing, Vol. 2, No. 1, January-March 2003
11. Čapkun, S., Hubaux, J.-P., Buttyán, L., "Mobility Helps Security in Ad Hoc Networks", In Proceedings of the 4th ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'03), Annapolis, Maryland, June 2003

12. Di Pietro, R., Mancini, L., Jajodia, S., “Efficient and Secure Keys Management for Wireless Mobile Communications”, Proceedings of the second ACM international workshop on Principles of mobile computing, pages 66-73, ACM Press, 2002
13. Diffie, W., Hellman, M., “New directions in cryptography”, IEEE Transactions on Information Theory, 22(6):644-652, November 1976
14. Eschenauer L., Gligor, Virgil D., “A Key-Management Scheme for Distributed Sensor Networks”, Proceedings of the 9th ACM Conference on Computer and Communication Security (CCS’02), Washington D.C., November 2002
15. Federal Information Processing Standard, Publication 180-1. Secure Hash Standard (SHA-1), April 1995
16. Hafslund A., Tønnesen A., Rotvik J. B., Andersson J., Kure Ø., “Secure Extension to the OLSR protocol”, OLSR Interop Workshop, San Diego, August 2004
17. Hollick, M., Schmitt, J., Seipl, C., Steinmetz, R., “On the Effect of Node Misbehavior in Ad Hoc Networks”, Proceedings of IEEE International Conference on Communications, ICC'04, Paris, France, volume 6, pages 3759-3763. IEEE, June 2004
18. Housley, R., Ford, W., Polk, W. and D. Solo, “Internet X.509 Public Key Infrastructure”, RFC 2459, January 1999
19. IEEE, “IEEE Std. 802.11b-1999 (R2003)”, http://standards.ieee.org/getieee802/download/802.11b-1999.pdf
20. Luo, H., Kong, J., Zerfos, P., Lu, S., Zhang, L., “URSA: Ubiquitous and Robust Access Control for Mobile Ad-Hoc Networks”, IEEE/ACM Transactions on Networking, October 2004
21. Matsumoto, T., Imai, H., “On the key predistribution systems: A practical solution to the key distribution problem”, Advances in Cryptology - Crypto’87, LNCS vol. 293, p. 185-193, 1988
22. Montenegro, G., Castelluccia, C., “Statistically Unique and Cryptographically Verifiable (SUCV) Identifiers and Addresses”, NDSS'02, February 2002
23. Plagemann, T. *et al*., “Middleware Services for Information Sharing in Mobile Ad-Hoc Networks - Challenges and Approach”, Workshop on Challenges of Mobility, IFIP TC6 World Computer Congress, Toulouse, France, August 2004
24. Pužar, M., Plagemann, T., “NEMAN: A Network Emulator for Mobile Ad-Hoc Networks”, Proceedings of the 8th International Conference on Telecommunications (ConTEL 2005), Zagreb, Croatia, June 2005
25. Rivest, R., “The MD5 Message-Digest Algorithm”, RFC 1321, April 1992
26. Stajano, R., Anderson, R., “The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks”, 7th International Workshop on Security Protocols, Cambridge, UK, 1999
27. The OpenSSL project, http://www.openssl.org/
28. Tønnesen A., “Implementing and extending the Optimized Link State Routing protocol”, http://www.olsr.org/, August 2004
29. Wallner, D., Harder, E., Agee, R., “Key management for Multicast: issues and architecture”, RFC 2627, June 1999
30. Wong, C., Gouda, M. and S. Lam, “Secure Group Communications Using Key Graphs”, Technical Report TR 97-23, Department of Computer Sciences, The University of Texas at Austin, November 1998
31. Zhou, L., Haas, Z., “Securing Ad Hoc networks”, IEEE Network, 13(6):24-30, 1999