DICOMA: DIsaster COntrol MAnagement

Deliverable D4.8

Final implementation of security services

Project Identifier	DICOMA
Project Title	DIsaster COntrol MAnagement
Document Version	1.0
Planned Delivery Date	Dec 31, 2013
Actual Delivery Date	
Document Title	Final implementation of security services
Work Package	WP4
Document Type	Word
Abstract	
Keywords	.

Function	*Name*	*Entity*
Editor	Juan Hernández Serrano & Juan Vera del Campo	UPC
Author	WP4 partners	DiCoMa

Contents

1 Executive Summary

Embedded systems (ES) for applications such as machine-to-machine (M2M) and wireless sensor networks (WSNs) are made up of a wide range of nodes equipped with sensor devices that capture data from the medium and can autonomously intercommunicate and cooperate in order to offer a wide variety of services. The sensor capabilities and the cooperation between nodes make these networks especially suitable for activities involving monitoring, tracking and controlling. Therefore, WSNs arise as powerful solutions for a wide range of services such as industrial monitoring, fire detection, fleet management, health monitoring, and smart utility networks, to name a few, all of which require a high, albeit differing, degree of security.

In chapter 4 of Deliverable 4.7, we detailed the most used security mechanisms in sensor networks and defined new specific mechanisms and protocols developed during this project as well as recommendations and guidelines targeted to guarantee the security of the services implemented in the DiCoMa project.

In this deliverable we present a working implementation of several key management mechanisms targeted to make more efficient the management of the keying material required by the different services. The proposed rekeying protocols get a level of efficiency so high that can be used in large networks of very-constrained devices such as sensor nodes; which make them ideal for the acquisition network of the DiCoMa project.

The reference implementation has been developed in Java 7. It includes the complete implementation of the protocol, suitable for working on real devices. In addition, a stand-alone simulator is provided to demonstrate the capabilities of the protocols.

The source code of the implementation is open and released under GNU Public License v3.0. It can be downloaded from the DiCoMa website at:

http://www.dicoma.eu/images/stories/articleFiles/treebasedgkm_src.zip

The Javadoc API of the provided application is at:

http://www.dicoma.eu/images/stories/articleFiles/treebasedgkm_doc.zip

Finally, a precompiled Java 7 runnable jar can be found at:

http://www.dicoma.eu/images/stories/articleFiles/treebasedgkm.rar

The rest of document is organized as follows: chapter 2 introduces the basis of group key management and Chapter 3 provides with a user guide to the provided application.

2 Introduction to Group Key Management

In *wireless sensor networks* (WSNs), it is necessary to establish a secure communication channel between sensor nodes where no attacker can eaves-drop, modify, replay, or inject messages on. This is what is called group security, which is targeted to provide group privacy, since data should be protected just from outsiders, and group authentication, since the only sources of communication should be the members of the group. In order to achieve this goal, every member knows a set of keys that are usually classified as: keys shared by two nodes or pairwise keys, and keys shared between group nodes or groupwise keys. Pairwise keys allow secure routing by hop-by-hop encryption and authentication, and provide easy isolation of a kidnapped member since compromised keys are just not used anymore. On the other hand, groupwise keys allow secure routing without the need of costly hop-by-hop encryption but providing hop-by-hop authentication and integrity (checking message authentication codes at every link). However groupwise keys are not resilient against node kidnapping and thus must be updated whenever a member is compromised.

Nevertheless both types of keys are part of a vicious circle if asymmetric cryptography is not used: in order to securely agree pairwise keys, nodes need a pre-shared secret that it is often the groupwise key; and in order to securely update a groupwise key, secure communications must be provided often by means of non-compromised pairwise keys

Group key management (GKM) studies the generation and updating of afore-mentioned keying material during the whole group life [1] thus warranting that only the current group members can authenticate and understand or decrypt messages within the group. We are certain that the application of known GKM techniques to WSN can secure the exchanged critical data while incurring a low impact on the network performance.

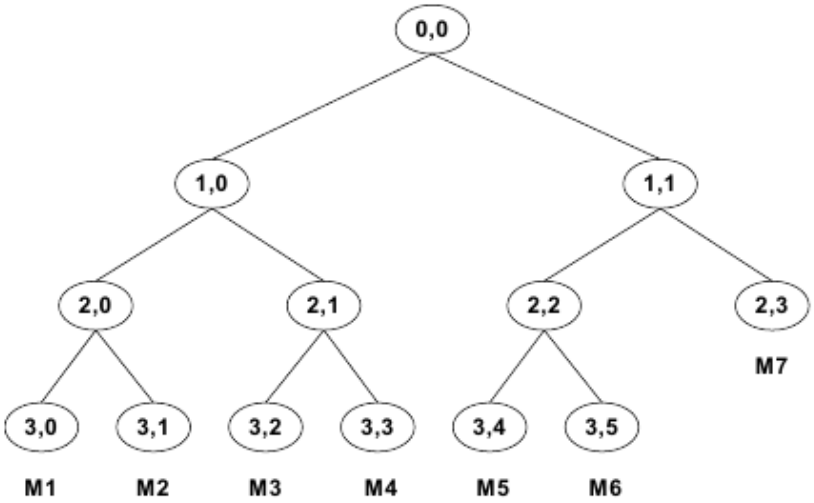

Figure 1: Logical tree of KEKs

Considering that the more energy consumption activity in WSNs is the transmission of a message over a wireless channel (reaching up to 3 orders of magnitude [2]), any GKM proposal for WSNs

must be designed to minimize the number of messages that has to be transmitted for re-keying. The most successful proposals for reducing the re-keying problem order are even nowadays based on the old well-studied logical key encryption keys (KEK) tree hierarchies [3, 4]. The simplest method for providing group security is merely based on the use of a groupwise key shared by all the group members that it is often called the network wise key, or group key or session encryption key (SEK). This key allows every group member to: 1) send encrypted data; 2) decrypt received data, and 3) authenticate itself as a group member since the knowledge of the session key guarantees that it belongs to the group. However, in order to securely update the SEK when the group membership changes, some other keys are necessary and these keys are the ones commonly called KEKs. Many GKM re-keying algorithms use these KEK trees since they substantially improve the efficiency in terms of bandwidth and latency. In this kind of methods, a key server builds a KEK tree only known to him and assigns a set of these KEKs to each of the members of the group. This set of keys corresponds to the path from the tree's leafs – where the members are- to its root. When a member leaves or joins a group only the KEKs belonging to that member need to be changed. Then new keys are delivered to the remaining members and the tree is reconstructed using the unchanged keys. In short, the cost of re-keying is reduced from O(N) to O(L), with N the number of members and L the depth of the tree.

In this document, we present a working implementation of the main tree-based GKM protocols, namely: *logical key hierarchy* (LKH) [5], which is the base of [6]; batch LKH [7], balanced batch LKH [8], *one-way function tree* OFT [9], single-message LKH [10] and batch single-message LKH [11]. Detailed descriptions of the protocols can be found in the cited sources and deliverable D4.7 of the DiCoMa project.

3 Tree-based GKM implementations: User guide

The presented application makes possible to create and test several implementations of GKM protocols based on tree-based logical key hierarchies. The graphical interface of the application makes easier to analyse the real-time performance of the protocols in terms of messages and bandwidth when a rekeying is needed (due to new memberships or leavings/expulsions). Moreover, the application allows the simulation of the protocols from an input file, which makes easier to evaluate the goodness of the implemented protocols under several predefined scenarios.

Running the application requires a java runtime environment installed and properly running. Application has been tested with Java version 7; thus Java version equal or greater than 7 is recommended (although it may work with previous versions). Several Java implementations can be found on the internet; however, Oracle Java (both the non-open-source and the open-source versions) is recommended and can be easily downloaded for the most common operating systems from the Oracle website at https://www.java.com/en/download/.

The application is provided as a runnable jar file (treebasedgkm.jar). Your operating system should detect that it is a java runnable application and execute it with Java; however, if file is not executed under Java for any reason, just “open with” your installed Java Runtime Environment.

Once opened, after the splash the application main window is as in Figure 2. In the top of the window, there is toolbar with several pull-down menus that will be later discussed. Just below the toolbar there are some shortcut icons for the most common menu options. The rest of the window is split into three panels: left, right and bottom panel. Left panel “LKH trees” shows the real-time shape of the logical tree or some simulation statistics if working with data from an input file. Right panel “Nodes info” shows tabbed sub panels with information about user-picked tree nodes. Finally, the bottom panel “LKH Log” shows a detailed log of real-time operations of the current protocol.

Next, every pull-down menu with its different options is detailed:

- File. The only option in this menu is “Exit”, which mainly exits the application the same as the window’s close button does.
- View. It contains the option “Allow animations (may cause some delay)”, which is switched on by default. The animation moves a red key over every compromised key during a rekeying. With the animation, as it slows things down, it is easier to observe the protocol behaviour during a rekeying.
- Mode. It contains two mutually exclusive options: “Real-time demonstration” (default) and “Simulation from file”. The former allows emulating the different GKM protocols and its performance during several rekeying processes. The latter allow us to evaluate the performance of the protocols from a pre-established input pattern in a file.

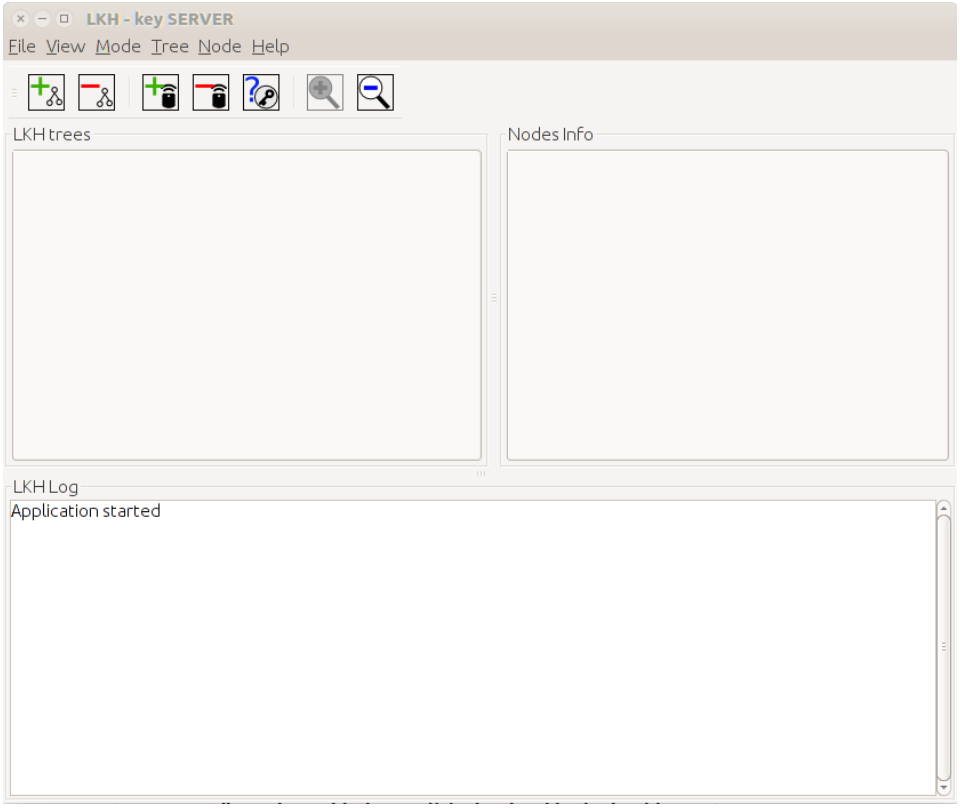

Figure 2. Application main window

- Tree. This menu has two options: “Create tree” and “Delete tree” with the same associated actions as the shortcut icons in Figure 3.

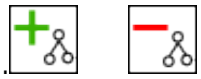

Figure 3. Shortcut icons for “Create tree” and “Delete tree”

“Delete tree” just deletes the current logical tree of keys, while “Create tree” creates a new tree. The behaviour of the latter depends on the previously chosen mode.

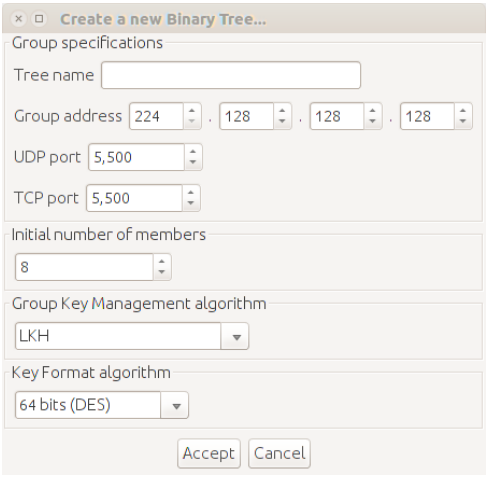

Figure 4. Create a new tree in real-time demonstration mode

On the one hand, if “Real-time demonstration” mode is chosen, a frame as in Figure 4 appears with a form to be filled with the following fields:

- Tree name. An optional string name for the new tree.
- Group address. IP address for the required multicast group.
- UDP port. UDP port used by the server to transmit new rekeying messages.
- TCP port. TCP port used by the server to listen for new join/leaving requests.
- Initial number of members. Number of members that will initially made up the group/tree.
- Group Key Management algorithm. Drop-down list that allows the user to choose among the different implemented GKM protocols: Simple LKH, Batch LKH, Balanced Batch LKH, OFT and Single-message LKH. In case the user chooses any of the "Batch" versions, a new option appears in order to let the user set the batch time (it defaults to 30 seconds).
- Key Format algorithm. A drop-down list with the currently implemented encryption algorithms: DES (64 bits), Blowfish (128 bits), DESede (192 bits) and AES (256 bits). These are the encryption algorithms and key lengths used during the rekeying process.

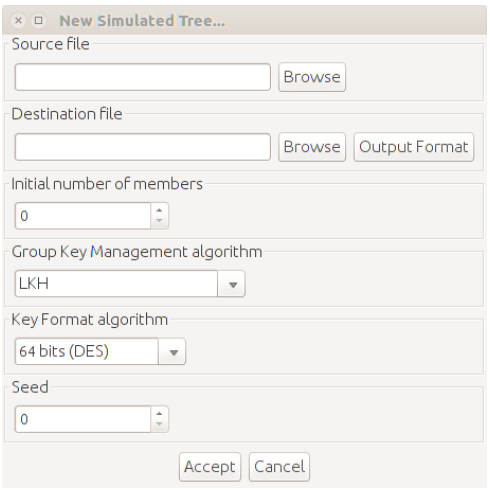

Figure 5. Tree creation when "Simulation from file" mode is chosen.

On the other hand, if the chosen mode is "Simulation from file", a frame as in Figure 5 appears with a formulary with the following fields:

- Source file. A file picker that allows the user to pick an input file with a pattern of joins and leavings. The input file must follow the next format: every single row accounts for a second in time and must hold an integer with the number of new member joins (a positive integer) or member leavings (negative integers).

- Destination File. The user can select the output file using this option. The format of the destination file is selected with the button "Output Format".

- Output Format. If the user clicks on this option, a new window appears with a selection of the information to be included in the destination file. The information to be optionally written in the destination file includes captions,

unicast and multicast messages exchanged in the protocol, amount of memory used by the server and latency of the communications. By default, all the switches are set. ¡Error! No se encuentra el origen de la referencia. shows this window.

Figure 6: information to be included in the destination file

- **Initial Number of Members**. The simulated network will start with this number of initial nodes.

- **Group Key Management Algorithm**. This drop-box list selects the specific protocol the simulator will demonstrate: LKH Simple, Batch LKH, Balanced Batch LKH, OFT and LKH Single Message. These protocols are explained in depth in D4.7 and the referenced material. If a protocol with a batch period is chosen, a new text box appears to enter this time period.

- **Key Format Algorithm**. The user selects in this list one of the available encryption algorithms available in the application: DES (64 bits), Blowfish (128 bits), DESede (192 bits) or AES (256 bits). With this option, the user selects the size of the security key and the type of the session key. In addition, if the rekeying algorithm requires the use of an encryption mechanism, the simulator is going to use the encryption scheme selected in this option.

- Seed. It is the initial seed to be used by the pseudorandom number generator. When the simulator drops off a node, the member leaving the tree is selected at random. Using the same seed will always output consistent simulations, and the same tree structure and source file with a different seed will generate different simulation scenarios.

- Node. This menu entry includes the options: “Add member (leaf node)”, “Remove member (leaf node)” and “Show node info”. These three options have the corresponding shortcut icons showed in Figure 7.

Figure 7: icons for the options: "Add member", "Remove member" and "Show node info"

- Help: This menu entry includes the item “About…”, which shows some information about the simulator.

- The icons with a Magnifier glass are used to zoom in and out the graphs, to show the complete tree or only some details about it. These icons are shown in Figure 8 .

Figure 8: icons to zoom in and out

Bibliography

[1] Li, X., Wang, Y., & Frieder, O. (2002). Efficient hybrid key agreement protocol for wireless ad hoc networks. In: Proceedings of the eleventh IEEE international conference on computer communications and networks, 2002. (pp. 404–409).

[2] J. R. Douceur, "The Sybil Attack," in 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), March 2002.

[3] Li, X., Wang, Y., & Frieder, O. (2002). Efficient hybrid key agreement protocol for wireless ad hoc networks. In: Proceedings of the eleventh IEEE international conference on computer communications and networks, 2002. (pp. 404–409).

[4] Chen, J., Kher, S., & Somani, A. (2006). Distributed fault detection of wireless sensor networks. In: Proceedings of the 2006 workshop on dependability issues in wireless ad hoc networks and sensor networks, DIWANS '06 (pp. 65–72). New York, NY: ACM. doi:10.1145/1160972.1160985.

[5] Harney, H. (1999). Logical key hierarchy protocol (lkh). Internet draft. Draft-harney-sparta-lkhp-sec-00.

[6] J. Hernández-Serrano, J. Vera-del-Campo, J. Pegueroles and C. Gañán "Low-cost group rekeying for unattended wireless sensor networks", Wireless Networks. Volume 19 Issue 1, January 2013.

[7] Li, X. S.; Yang, Y. R.; Gouda, M. G.; Lam, S. S. (2001) Batch Rekeying for Secure Group Communications. X International World Wide Web Conference (WWW10). Hong Kong, China.

[8] Pegueroles, J.; Rico-Novella, F. (2003). Balanced Batch LKH: New Proposal, Implementation and Performance Evaluation. IEEE Symposium on Computers and Communications – ISCC'2003. Antalya, Turquía.

[9] Balenson, D. M.; McGrew, D.; Sherman, A. (2000). Key Management for large Dynamic Groups: One-Way Function Trees and Amortized Initialization. Internet Draft.

[10] Pegueroles, Wang-Bin, Soriano, and Rico-Novella, Group Rekeying Algorithm using Pseudo-Random Functions and Modular Reduction. Lecture Notes in Computer Science, vol. 3032, pp. 875 – 882, 2004.

[11] Pegueroles, J., F. J. Rico, J. Hernández-Serrano, and M. Soriano. Adapting GDOI for balanced batch-LKH. Internet Draft. Proceedings of the 57th IETF Meeting, 07, 2003.