Security analysis and resource requirements of group-oriented user access control for hardware-constrained wireless network services

D. Ventura[1], Aitor Gómez-Goiri[2], V. Catania[1], Diego López-de-Ipiña[2], J.A.M. Naranjo[3], and L.G. Casado[4]

[1] Dpt. of Electrical, Electronic and Computer Engineering,
University of Catania, Italy
{daniela.ventura, vincenzo.catania}@dieei.unict.it
[2] Deusto Institute of Technology - DeustoTech,
University of Deusto, Spain
{aitor.gomez, dipina}@deusto.es
[3] OAK Team, INRIA-Saclay, and
LRI, Université Paris-Sud, France
juan-alvaro.munoz-naranjo@inria.fr
[4] Dpt. of Computer Science,
University of Almería,
Agrifood Campus of International Excellence (ceiA3), Spain
leo@ual.es

Abstract. We extend and analyze a previous access control solution for wireless network services with group-based authorization. Authentication and encryption are provided, and access control relies on user identity, group membership and time intervals. Both the basic solution and the extension focus on minimizing computation, energy, storage and communications on the sensor side: computations involved rely on symmetric cryptography and key derivation functions, and no additional messages between user and sensor are needed. The performance of our solution is proven by experiments on a highly constrained platform such as Arduino. Finally, its security is validated against the AVISPA tool.

Keywords: access control, group-based authorization, wireless network services, Internet of Things, ubiquitous computing

1 Introduction

Together with mobile computing, IoT constitutes the clearest sign of the Ubiquitous Computing prominence in our current lives [1]. On the other hand, security has been an ever-present concern in Internet communications, and will keep being in the new scenario: if we want the IoT paradigm to reach all its possibilities then we need to provide reliable routines for information encryption and user authentication and authorization. Furthermore, these routines must be able to run seamlessly in very constrained hardware: small and cheap devices with limited processing capabilities and sometimes energy restrictions. For example, a typical mote in a wireless sensor network is not able to make use of public key cryptography (on a frequent manner at least) given the high computational and energy demands of the latter. Hence very lightweight security routines are needed. In [2] we presented an access control solution for wireless environments in which users access services offered by constrained devices (e.g., wireless sensors). This solution provides efficient encryption, authentication and authorization on a per-user basis, i.e. a given user can access the services offered by a given sensor based on her identity. Furthermore, it needs no additional messages in the user-sensor communication. In [3], we extend the work in [2] in order to differentiate groups of credentials in the authorization process, i.e., users can access the services offered by a group of sensors when they have the corresponding group credentials. The groups can be either hierarchical or non-hierarchical. In the latter, members in different privilege groups enjoy different non-hierarchical sets of services. In the former, members in higher privilege groups enjoy more services than lower level users. Here, we study the basic protocol and the group credentials extension in terms of security and resource requirements. A security analysis is performed with the AVISPA tool [4] and experiments are conducted in the Arduino platform, thus complementing earlier results obtained with a Raspberry Pi unit [2]. At the time of publication the source code used for the experiments is being consolidated with the aim of offering an open-source solution that can be used by the community in real deployments [5].

The article is organized as follows. Section 2 discusses some proposals from the literature. Sections 3 and 4 present the scenario we are addressing here and recall the basic protocol, respectively, while Section 5 describes the groups extension. Sections 6 and 7 shows experimental results in the Arduino platform and analyses the security of the protocol with the AVISPA tool. Finally, Section 8 concludes the article.

2 Related work

The popular SPINS solution [6] provides lightweight symmetric encryption and authentication in wireless sensor networks where a Base Station is actively involved. It is composed of two sub-protocols: SNEP, which provides encryption, authentication and data freshness evidence between two parties, and μTESLA, used for authenticating broadcast messages to the whole network. LEAP+ [7] proposes an authentication and encryption framework for similar scenarios. Apart from its own protocols, μTESLA is used for authentication of broadcast messages from the Base Station. Ngo et al [8] proposed an access control system for the scenario we address here: wireless networks that provide services to users supported by an Authorization Service. It provides both individual and group-based authentication thanks to the combination of user keys and group keys. The recent MAACE [9] also focuses on the same scenario with individual and per-group authentication. However its storage requirements at every sensor are very large (sensors must store all keys shared with online users at a given time). The authors solve the storage problem by involving the Base Station in frequent communications, which is not a proper solution from our point of view since sending information is by far the most energy-consuming operation for sensors.

3 Scenario

The scenario we address in this work involves three kinds of players: sensors, Base Stations and user devices (e.g. smartphones), interacting together in a given facility (buildings, factories, greenhouses, homes, etc).

Sensors are extremely constrained wireless devices, frequently battery-powered and with reduced computational capabilities, which provide users with services of any kind.

Base Stations are better equipped devices that handle groups of sensors for message routing purposes, data collection and also for key management in our case. They are assumed to have a more powerful hardware and a permanent (or at least much larger) power supply and large storage space.

Finally, users communicate with Base Stations and sensors through their powerful smart devices, such as mobile phones or tablets.

The key point here is that sensors need to perform access control on users but face several limitations: 1) they are not able to handle complex public-key authentication nor encryption routines and 2) they do not have enough memory space so as to keep large sets of user keys. In consideration of those constraints the basic protocol in [2] provides an access control mechanism with symmetric encryption and authentication routines which minimizes storage requirements. On top of that, the groups extension introduced in [3] allows to manage users on a per-group basis: each user group has a different set of privileges, meaning that they can access different sets of the services provided by the sensors. For the sake of completeness, both methods are shown here. Table 1 shows the notation used throughout the article.

4 The basic protocol

Here we briefly summarize the initial version of the protocol as showed in [2]. It provides encryption and user access control to user $\leftrightarrow$ sensor one-to-one communications. The Base Station, a more powerful device, performs high-level authentication on the user (with authorization certificates based in public key cryptography, for example) and provides her with two symmetric keys (for encryption and authentication, respectively) and parameters for their generation at the sensor. If those parameters are attached to the first message of a conversation then the sensor can input them to a Key Derivation Function in order to obtain an identical pair of symmetric keys that make communication possible. Figure 1 depicts the message exchange in the protocol. Let us explain it with more detail.

1. At the time of sensor deployment, the latter receives a master secret MS_S, which is *secretly shared* by the Base Station BS and the sensor S (see the end of this section for secret channels).
2. Upon arrival, user A sends her credentials (e.g. an authorization certificate) to BS so high-level access control can be performed, and the list of sensors she wants to communicate with (in Fig. 1 we only consider S). This step is run only at user arrival.

$A,\ BS,\ S$	User, Base Station and Sensor identifiers, respectively
MS_S	Master secret for sensor S
$Kenc_{S,A},\ Kauth_{S,A}$	Encryption and authentication keys for communication between sensor S and user A
$Kenc_{S,A}\{x,\ ctr\}$	x is encrypted in counter mode using key $Kenc_{S,A}$ and counter ctr
$MAC_{Kauth_{S,A}}(x)$	A MAC is done on x using $Kauth_{S,A}$
$KDF(x,\ \{a,\ b\})$	A Key Derivation Function is applied to master secret x using a as public salt and b as user-related information
$H(x)$	A hash function is applied to x
$x\|\|y$	Concatenation of x and y
ID_A	Identifier of user A
a	Random integer salt
$init\_time,\ exp\_time$	Absolute initial and expiration time of a given key
MS_p	Master secret for privilege group p
$Kenc_{p,A},\ Kauth_{p,A}$	Encryption and authentication keys between sensors offering services for group p and user A
ID_p	Identifier of privilege group p
$A \rightarrow *$	User A sends a message to any listening sensor
$S_p \rightarrow A$	One sensor giving services from privilege group p sends a message to A

Table 1. Notation

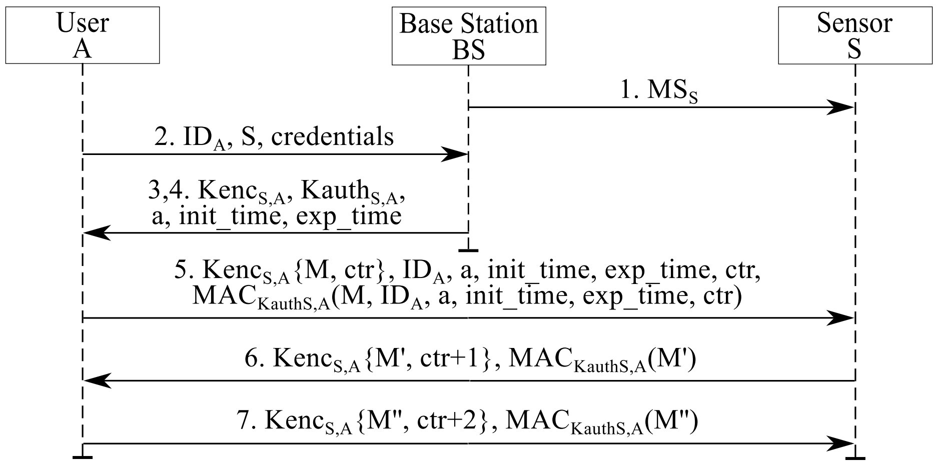

User
A
Base Station
BS
Sensor
S
1. MS_S
2. ID_A, S, credentials
3,4. Kenc_S,A, Kauth_S,A, a, init_time, exp_time
5. Kenc_S,A{M, ctr}, ID_A, a, init_time, exp_time, ctr, MAC_KauthS,A(M, ID_A, a, init_time, exp_time, ctr)
6. Kenc_S,A{M', ctr+1}, MAC_KauthS,A(M')
7. Kenc_S,A{M'', ctr+2}, MAC_KauthS,A(M'')

Fig. 1. Messages involved in the original protocol

3. BS computes:
   (a) a, random integer salt
   (b) $(init\_time,\ exp\_time)$, keying material validity interval
   (c) $Kenc_{S,A},\ Kauth_{S,A} = KDF(MS_S, \{a,\ ID_A || init\_time || exp\_time\})$

4. BS sends the information generated in the previous step to A under a secure channel (see the end of this section).
5. A encrypts her first message to S with $Kenc_{S,A}$ in counter mode (thus using a fresh counter ctr), attaches parameters ID_A, a, $init\_time$, exp_time, ctr in plain text and a MAC obtained with $Kauth_{S,A}$.
6. Upon reception of the message, S obtains the key pair $Kenc_{S,A}$, $Kauth_{S,A}$ by feeding the Key Derivation Function with the attached parameters; S can now decrypt the message. The reply is encrypted in counter mode with $Kenc_{S,A}$ and $ctr + 1$ and authenticated with a MAC using $Kauth_{S,A}$.
7. Any subsequent message is encrypted and authenticated with the same key pair after increasing the counter by one.

When the message exchange finishes the sensor may delete all information related to the user since it can be recomputed at the beginning of the next exchange, thus saving space at the sensor. Caching techniques can be applied though, as we will see in Section 7.

The sensor is sure of the authenticity of the user since the only way of knowing ($Kenc_{S,A}$, $Kauth_{S,A}$) is either knowing MS_S (which is kept secret) or obtaining it from the Base Station (which is actually the case). What is more, the MAC at the end of the message provides integrity assurance in addition to authentication. We refer the reader to [2] for more considerations on security, efficiency, message overhead and storage of the basic protocol.

Regarding the trasmission of MS_S from BS to S in Step 1, a secure channel between them can be easily established by pre-installing a shared symmetric key in S before deployment. Having a secure channel allows also to renew MS_S to enhance security (note that changing MS_S will affect the keys generated in Steps 3c and 6 so active users should receive a new pair). For the secure channel between A and BS mentioned in Step 4 we assume both the user device and the base station can use public-key cryptography (e.g. SSL/TLS).

5 A groups extension

In this section we address a scenario with different groups of users, each group giving its members access privilege to a given set of services provided by sensors. Services provided by a sensor may (but not necessarily) belong to more than one group. The associated access control routines should not be intensive in terms of computations or message exchanges.

Let us assume that there are $l > 0$ groups. The main idea is that there exists a different master secret MS_p for every privilege group $p \in [1,\ l]$, hence sensors should only reply to service requests encrypted and/or authenticated with a key pair derived from the corresponding master secret. In [3] we propose two different approaches based on how services are arranged into groups. In Approach 1 privilege groups are not hierarchical, like in the case of employees that are allowed to enter different areas of a facility based on their activity (though some services might be in more than one group). In Approach 2 privilege groups are hierarchical, hence a user with privilege level p should enjoy all privileges from groups [1, p]. An example of this scenario is a smart house with different privilege groups based on age: children would have access to certain services of the house, while parents should have full control of the house.

Due to lack of space we refer the reader to [3] for theoretical considerations about security and overhead in message length and storage.

5.1 Approach 1: non-hierarchical privilege groups

In this case, the Base Station generates l independent random master secrets $MS_1, \ldots, MS_l$ assuming there exist l different privilege groups. Sensors offering services from any privilege group p receive MS_p from the Base Station under a secure channel. In this scenario, users will typically belong to one group only, and sensors will provide services to one group as well. Figure 2(a) shows an example with three users and three sensors. However, if a sensor offers services to different privilege groups (or if a given service is included in more than one group), then the sensor should store each group's master secret. In a similar way, users assigned to more than one group (if that occurred) should receive a different pair of keys per group, and use the appropriate one to the requested service.

When user A arrives at the system the Base Station authenticates her and generates a different pair of symmetric keys ($Kenc_{p,A},\ Kauth_{p,A}$) for the privilege group A belongs to (group p in this case). These keys are generated by the BS and sensors assigned to group p in the same way as in the basic protocol: the user identifier, a random salt a and a key validity interval ($init\_time,\ exp\_time$) are fed to a Key Derivation Function along with the corresponding master secret as shown in Eq. (1).

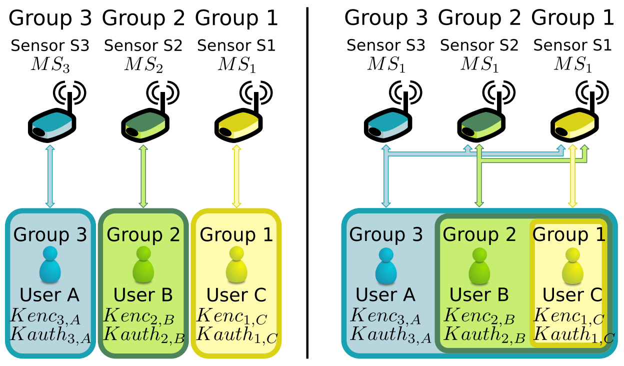
$$Kenc_{p,A},\ Kauth_{p,A} = KDF(MS_p, \{a,\ ID_A||init\_time||exp\_time\}) \qquad (1)$$

These keys are sent to A by the BS under a secure channel (see Section 4). When user A wants to request a service from privilege group p she needs to encrypt and authenticate her message with that pair of keys like in the basic protocol (note that ID_p has been added).

$$\begin{aligned} A \rightarrow * : [&Kenc_{p,A}\{M,\ ctr\},\ ID_A,\ ID_p,\ a,\ init\_time,\ exp\_time,\ ctr, \\ &MAC_{Kauth_{p,A}}(M,\ ID_A,\ ID_p,\ a,\ init\_time,\ exp\_time,\ ctr)] \qquad (2) \end{aligned}$$

Any nearby sensor providing services from group p (let us name it S_p) can now reply to A after deriving the appropriate pair of keys from the received information and MS_p. The counter is explicitly stated on plain text so synchronization is not lost due to an arbitrary sequence of messages if more than one sensor is involved in the conversation.

$$S_p \rightarrow A : [Kenc_{p,A}\{M',\ ctr+1\},\ ctr+1,\ MAC_{Kauth_{p,A}}(M',\ ctr+1)] \qquad (3)$$



(a) Approach 1 (b) Approach 2

Fig. 2. Examples of the two approaches with three groups.

5.2 Approach 2: hierarchical privilege groups

In this case, services are arranged in hierarchical groups: users assigned to privilege group p should be granted access to all services in groups $[1,\ p]$. Here every sensor in the system receives the lowest group's level master secret MS_1 from the BS. The rest are obtained by hashing the immediately lower master secret, i.e. $MS_p = H(MS_{p-1})$. This requires lower permanent storage requirements at the cost of a slightly higher computational demand and more security risks as every sensor can obtain the master secret for any privilege level. Figure 2(b) shows an example with three users and three sensors.

Thanks to this modification, user devices need to store only one pair of keys, that of the highest privilege level they are granted. For example, a user A in group 3 will only receive $(Kenc_{3,A},\ Kauth_{3,A})$ from the Base Station. However the use of this key pair is enough for being granted access to any service in groups 1 to 3.

The verification of user credentials at the sensor side goes as follows. After receiving a message encrypted and authenticated with $(Kenc_{p,A},\ Kauth_{p,A})$ (see Eq. (2)) the sensor derives $MS_p = H(...H(MS_1))$. From MS_p and user-bound parameters the sensor obtains $(Kenc_{p,A},\ Kauth_{p,A})$ as in Eq. (1). Communications can now be established as in Eq. (3).

5.3 Combining hierarchical authentication with individual privacy

The basic protocol provides one-to-one authentication and encryption between a user and a sensor. On the other hand, approaches 1 and 2 allow to perform one-to-many authentication and encryption: all sensors holding the affected master secret will be able to authenticate the user and decrypt the conversation. Next, we consider the possibility of having services that demand one-to-one private communications and group-based authorization at the same time. For achieving this we base on Approach 1, however the extension to Approach 2 is straightforward.

In this case sensor S is assigned by the Base Station an individual master secret MS_S (as in the basic protocol) and one master secret MS_p for each privilege group p the sensor provides services from (in Approach 2 the sensor would be assigned MS_1 and would derive the rest by hashing).

User A is assigned a pair of keys for individual communication with S, i.e. $(Kenc_{S,A},\ Kauth_{S,A})$, and a pair of keys $(Kenc_{p,A},\ Kauth_{p,A})$ for the privilege group she is entitled to, say p. Like before, these keys are generated for A by the Base Station by feeding MS_p and user-related parameters $ID_A,\ a,\ init\_time,\ exp\_time$ to a Key Derivation Function.

Now, when A wants to communicate only with S while proving her authorization level, she encrypts her messages with $Kenc_{S,A}$ and computes the corresponding MAC with $Kauth_{p,A}$ as in Eq. (4). S replies using the same pair of keys and incrementing the counter, which needs not to be included on plain text given that the message exchange takes place between two players only:

$$\begin{aligned} A \rightarrow S : [&Kenc_{S,A}\{M,\ ctr\},\ ID_A,\ ID_p,\ a,\ init\_time,\ exp\_time,\ ctr, \\ &MAC_{Kauth_{p,A}}(M,\ ID_A,\ ID_p,\ a,\ init\_time,\ exp\_time,\ ctr)] \quad (4) \\ S \rightarrow A : [&Kenc_{S,A}\{M,\ ctr+1\}, MAC_{Kauth_{p,A}}(M,\ ctr+1)] \quad (5) \end{aligned}$$

6 Experimental environment

To evaluate our solution we measure two different aspects: how it behaves in a embedded platform and its security strengths and weaknesses.

In order to measure its performance we have implemented and tested step 3c of the protocol in Section 4, which is the core of our proposal and its most resource-demanding stage. The tests were run in the Arduino platform [10]. The code is publicly available at `http://github.com/dventura3/Nist`.

We run the experiments in two Arduino boards (Uno and Mega) to analyse whether they behave differently. Table 2 shows their technical specifications.

Table 2. Technical characteristics of the assessed Arduino boards

Arduino Board	Microcontroller	Flash Memory	SRAM	EEPROM	Reference
Uno	16 MHz ATmega328	32 kB	2 kB	1 kB	[11]
Mega2560	16 MHz ATmega2560	128kB	8 kB	4kB	[12]

Furthermore, for the derivation function we used keys of different lengths: 128 bits, 256 bits and 512 bits. To derive the keys, we used the NIST key derivation function [13] together with the HMAC-SHA-1 and HMAC-SHA-256 authentication functions [14]. SHA-1 is used as a baseline, although its use is currently not advised anymore. SHA-512 implementation is not currently implemented in the Cryptosuite library used [5]. Therefore, to obtain the key 512 bits key we call to SHA-256 twice with two different subkeys.

7 Evaluation

7.1 Performance evaluation

To know the impact of our solution in Arduino, we derived keys of 128, 256 and 512 bits 100 times in each of the Arduino models tested. The average time needed to derive each key is 28.67, 99.84 and 288.15 ms respectively with standard deviations of less than 1 ms. The measures showed no difference between models.

The sensor, i.e., the Arduino board, will need to generate two keys: one for authentication and another for encryption. Considering, the 288 ms needed to generate a 512 bits key, it will require 576 ms in total. If we also take into account the additional tasks to be performed afterwards (decryption, encryption and MAC), the sensor might take too much time to answer a request. To mitigate this effect, the sensor can cache the pair of keys for each user/group at the cost of using more memory.

To analyse how our solution would affect the memory consumed by an Arduino board, we first checked how deriving keys affects their free memory. Each Arduino board allocated 38, 54 and 106 bytes during the generation of each key of 128, 256 and 512 bits respectively.

Since the most power-demanding operation in a sensor is airing messages through its antenna [6,7], we only send the data needed to create a key once (see Figure 1). This reduces the message length of the subsequent requests, but forces the sensor to store ID_p, a, *init_time*, *exp_time* and the updated *ctr* to regenerate each key. Arduino needs 16 bytes to store each of these set of fields. Figure 7.1 represents this case with the blue bars. Considering the most limited board (i.e., Uno), it is able to manage the data of 26, 24, 20 users or groups for each key length in SRAM with the current program. As the program is loaded also in SRAM, a more complex program will reduce this number. However, if we ignore all the MS_S stored and other additional data stored by the program, the EEPROM could store enough information for other 64 additional users or groups.

The available EEPROM will be additionally reduced in approaches 1 and 2 because they require the sensor to store permanently a pair of keys for the group. Approach 1 requires the sensor to store a master secret for every privilege group it might be assigned to. In Approach 2, the sensor can decide (a) to permanently store a single master secret or (b) to store all master secrets once derived. *Case a* saves computations at the cost of the space, while *Case b* the saves memory increasing the computation.

If the sensor caches the keys as suggested before, it will keep in memory $Kenc_{S,A}$, $Kauth_{S,A}$, ID_a *exp_time*, and the updated *ctr*. This will require it

[5] `https://github.com/dventura3/Cryptosuite`

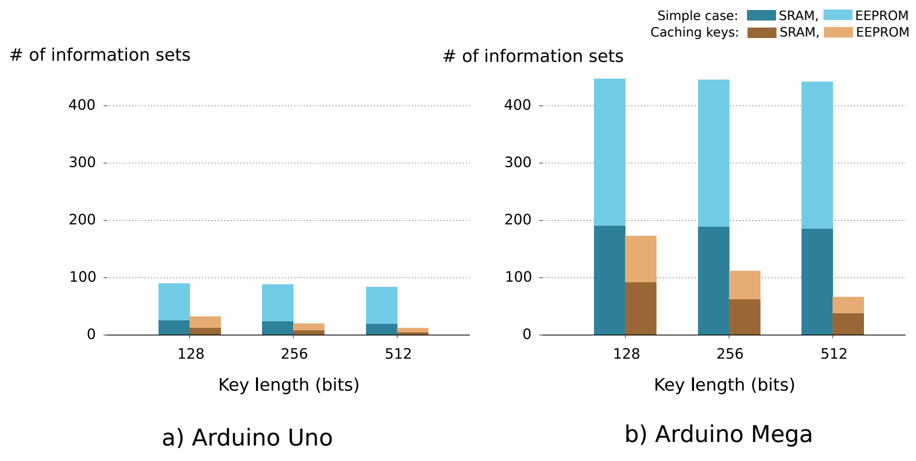


Fig. 3. A sensor needs to keep a set of fields about each user/group it communicates with. The chart shows how many sets each board can manage. In the most simple case it derives the keys needed after each request. In the other, it caches the keys.

to store 50, 82 and 146 bytes for 128, 256 and 512 bits. Figure 7.1 represents this case with the brown bars. Considering the memory already consumed by the program, Arduino Uno will be able to keep the data for 13, 8 and 5 users in SRAM for each key length. Using the EEPROM, it could manage the information of 20, 12 and 7 additional users.

To summarize, normal operation of the device generating two keys at each request is slow for the worst case (512 bits), but acceptable for the 128 and 256 bits cases. The storage of the keys in cache reduce the response time but it also reduces the maximum number of users that can use the sensor due to its memory limitations. The minimum of the maximum number of concurrent users is five, which is determined taking into account the most limited board and the most demanding memory instances of our experiments, i.e. memory needed by the program, the 512 key lengths and the use of cache. The settings of the systems will depend on the characteristics of the final devices and the number of concurrent users the application demands.

7.2 Security evaluation with the AVISPA tool

The well known AVISPA tool (Automated Validation of Internet Security Protocols and Applications) [4] performs model checking for security protocols with a high level of reliability. Model-checking techniques allow to automatically verify finite-state-concurrent systems such as communication protocols in a fast and, more importantly, deterministic way. The latter implies that every possible state of the protocol is visited. We have modeled our protocol in the HLPSL

(High Level Protocol Specification Language) [15] language specification and used AVISPA 1.1 to search for possible attacks.

Two different flavors of our protocol were chosen for the analysis: (1) the basic version (Section 4) and (2) the individual-groups combination (Section 5.3). The plain groups versions (Sections 5.1 and 5.2) can be considered similar to the basic one since the only difference relies on which master secret is used for key derivation. We have made both HLPSL scripts publicly available at `http://www.hpca.ual.es/~jalvaro/stuff/AVISPA/`.

The three players have been modeled in each of them. As for the initial (A,BS) and (BS,S) secure channels, the former has been modeled as a shared symmetric key between both players, while in the latter case we assume the master secret(s) are already known by both players.

Regarding security goals, the following ones were imposed:

1. Secrecy on the data sent from A to S (i.e. Steps 5 and 7 in Fig. 1).
2. Authentication on all keys used between A and S (i.e. $Kenc_{S,A}, Kauth_{S,A}$ as well as $Kenc_{p,A}, Kauth_{p,A}$) so any message encrypted and MAC-ed with those keys must come either from A or S.

Communication channels assume the presence of a Dolev-Yao Intruder [16]. This implies three main assumptions: the adversary can *(i)* capture any message in the network, *(ii)* impersonate any legitimate user and *(iii)* run many concurrent instances of the protocol. Other assumptions made by the model are *(iv)* one-way functions are perfectly secure and *(v)* there exists a Public Key Infrastructure in which all public keys are known by all players and every private key is only known by its owner (this one is not relevant to our protocol since we only use symmetric key cryptography). The analysis covered concurrent instances of both protocols with the intruder impersonating every role.

We tested the protocols against three of the model checkers provided with AVISPA: OFMC, CL-ATSE and SATMC. These checkers verify the fulfillment of the imposed security requirements at any possible execution state of both protocol versions. All outputs returned a "SAFE" result against Dolev-Yao intruders under the specified security requirements; no attacks were reported.

8 Conclusions

Here we study the security and performance of a group-based extension for an access control protocol oriented to wireless network services. This extension addresses infrastructures populated by constrained devices (such as wireless sensors) that are arranged in different groups of services: users are granted access to these groups depending on their privileges. We consider two different scenarios, depending on whether privilege groups are hierarchical (entitlement to a privilege group implies access to all services down to the lowest group) or not (users can only access services contained in the very privilege group they are entitled to). Furthermore, we show a way of combining individual encryption with group-based authorization. Regardless of the approach chosen, the authentication and authorization processes are performed efficiently and with no additional messages between the user and the addressed sensor.

In order to prove the security and applicability of our proposal we perform two different tests. First, the security mechanisms in which this project is based have been adapted to the popular Arduino platform. Using them, we have measured the performance of our solution in the real world. The key derivation functions which are in the core of this paper show a good performance in these really limited platforms. However, it is advisable to cache the keys to avoid their re-generation affecting future requests of the same user or group. These experiments complement earlier results obtained from a Raspberri Pi unit.

Second, we model the protocol in HLPSL language and we perform a security analysis by means of the AVISPA tool with the following requirements: privacy of the data and authentication of the cryptographic keys used in the protocol against Dolev-Yao intruders. The tool did not find any attack.

As of future work, we plan to develop an open-source solution for real deployments.

Acknowledgements

This work was funded by the Spanish Ministry of Economy and Competitiveness (TIN2008-01117, TIN2012-37483, and IPT-2011-1558-430000), Junta de Andalucía (P11-TIC-7176), and partially by the European Regional Development Fund (ERDF). Part of this work was done while J.A.M. Naranjo was with University of Almería.

References

1. Aitor Gómez-Goiri, Pablo Orduña, Javier Diego, and Diego López-de-Ipiña. Otsopack: Lightweight Semantic Framework for Interoperable Ambient Intelligence Applications. Computers in Human Behavior, Volume 30, Pages 460-467, January 2014, ISSN 0747-5632, 10.1016/j.chb.2013.06.022.
2. J.A.M. Naranjo, Pablo Orduña, Aitor Gómez-Goiri, Diego López-de-Ipiña and L.G. Casado. Enabling user access control in energy-constrained wireless smart environments. Journal of Universal Computer Science, Volume 19, number 17, Pages 2490-2505, November 2013.
3. J.A.M. Naranjo, Aitor Gómez-Goiri, Pablo Orduña, Diego López-de-Ipiña and L.G. Casado. "Extending a User Access Control Proposal for Wireless Network Services with Hierarchical User Credentials". International Conference CISIS13. Advances in Intelligent Systems and Computing (2014) 239 pp: 601-610.
4. The AVISPA Project. `http://www.avispa-project.org`
5. Lightsec Project. `http://github.com/lightsec`
6. Perrig, A., Szewczyk, R., Tygar, J. D., Wen, V. and Culler, D. E. "SPINS: security protocols for sensor networks". Wireless Networks (2002) 8:5, pp: 521-534.
7. Zhu, S., Setia, S. and Jajodia, S. "LEAP+: Efficient security mechanisms for large-scale distributed sensor networks". ACM Transactions on Sensor Networks, (2006) 2:4, pp: 500-528.

8. Ngo, H.H., Xianping W., Phu D.L. and Srinivasan, B. “An Individual and Group Authentication Model for Wireless Network Services”. JCIT (2010) 5:1, pp: 82-94.
9. Le, X.H., Khalid, M., Sankar, R. and Lee S. “An Efficient Mutual Authentication and Access Control Scheme for Wireless Sensor Networks in Healthcare”. Journal of Networks (2011) 6:3 pp: 355-364.
10. Banzi, M. “Getting Started with Arduino”. O’Reilly Media, Inc, (2009).
11. Arduino Uno. `http://arduino.cc/en/Main/arduinoBoardUno`
12. Arduino Mega. `http://arduino.cc/en/Main/arduinoBoardMega`
13. Chen, L. “Recommendation for Key Derivation Using Pseudorandom Functions”. NIST Special Publication 800-108 (2008).
14. Bellare, M., and Canetti, R, and Krawczyk, H. “Keying Hash Functions for Message Authentication”. Proceedings of CRYPTO’96, pp: 1–15. Springer, 1996.
15. Chevalier, Y., Compagna, L. et al. A high level protocol specification language for industrial security-sensitive protocols. Proceedings of Workshop on Specification and Automated Processing of Security Requirements (SAPS), Linz, Austria, 2004.
16. Dolev, D., Yao, A. “On the security of public key protocols”. IEEE Transactions on Information Theory, (1983) 29:2, pp:198208.