An Energy-Efficient Symmetric Cryptography Based Authentication Scheme for Wireless Sensor Networks

Oscar Delgado-Mohatar[1], José M. Sierra[2], Ljiljana Brankovic[3], and Amparo Fúster-Sabater[1]

[1] Instituto de Física Aplicada, C.S.I.C, Madrid (SPAIN) *
oscar.delgado@iec.csic.es
[2] Universidad Carlos III de Madrid, Departamento de Informática, Leganés, Madrid (SPAIN)
[3] School of El.Eng. & Comp.Sc, Faculty of Engineering & Built Environment Callaghan, The University of Newcastle (AUSTRALIA)
Ljiljana.Brankovic@newcastle.edu.au

Abstract. Sensor networks are ad-hoc mobile networks that include sensor nodes with limited computational and communication capabilities. They have become an economically viable monitoring solution for a wide variety of applications. Obviously, it is important to ensure security and, taking into account limited resources available in wireless sensor networks, the use of symmetric cryptography is strongly recommended. In this paper we present a light-weight authentication model for wireless sensor networks composed of a key management and an authentication protocol. It is based on simple symmetric cryptographic primitives with very low computational requirements, and it achieves better results than other similar proposals in the literature. Compared to SPINS and BROSK protocols, our system can reduce energy consumption by up to 98% and 67% respectively. It also scales well with the size of the network, due to it only requiring one interchanged message, regardless of the total number of nodes in the network.

1 Introduction

Due to the unique nature of Wireless Sensor Networks (WSN), ensuring their security is problematical in many ways and six common challenges are usually defined as follows [7]: (i) wireless nature of communication, (ii) resource limitation on sensor nodes, (iii) very large and dense WSN, (iv) lack of fixed infrastructure, (v) unknown network topology prior to deployment, (vi) high risk of physical attacks on unattended sensors. In addition to these security challenges, there

* This work was supported in part by CDTI (Spain) and the companies INDRA, Unión Fenosa, Tecnobit, Visual Tool, Brainstorm, SAC and Technosafe under Project Cenit-HESPERIA as well as supported by Ministry of Science and Innovation and European FEDER Fund under Project TIN2008-02236/TSI.

is a number of unavoidable constraints that have to be taken into account, e.g. power consumption and key management constraints.

In this paper we present a novel light-weight authentication model, specifically tailored towards sensor networks with low computational resources. The protocol is based on symmetric cryptographic primitives and it makes use of a network-wide key for deriving pair-wise keys. Once the pair-wise keys are derived, the master key is no longer needed and can be erased from the memory.

Our approach has the benefits of pair-wise key schemes, i.e. node capture resilience, without requiring pre-distribution and storage of a large number of keys in each node. As a consequence, our scheme scales well and could manage networks with an arbitrarily large number of nodes. In addition, the protocol includes session-key transport capabilities, which allow us to achieve both objectives, authentication and key establishment, with a reduced number of interchanged messages.

The rest of the paper is organized as follows. In the next section we first introduce the notation used throughout the paper. We then present and analyse our proposal and we compare it to two well-known security schemes.2. In Section 3 we evaluate the performance and efficiency of our proposal, both in terms of energy consumption and scalability . Finally, we give some concluding remarks in Section 4.

1.1 Notation

For reasons of clarity, the symbols used in this paper are listed below:

k_M	Network-wide master key.
$\{M\}_k$	Encryption of message M with key k.
$[M]$	Hash of message M.
$[M]_k$	Hash of message M with key k (HMAC).
$[M]^i$	Message M is hashed i times without key.
$[M]^i_k$	Message M is hashed i times with key k.
k^A_{enc}	Encryption key of the node A.
∇^i_j	jth tuple of the ith cycle authenticator.
k^j_{auth}	Authentication key of the jth authentication cycle, that is, $k^j_{auth} = [k_M]^j$

2 Proposed Authentication and Key-Establishment Scheme

We next present the basic features and properties of our scheme. It is composed of two protocols: a *key establishment scheme*, carried out during the network deployment, and an *authentication protocol*, which is used when a new node joins the network once the previous phase is over.

The proposal has been designed to be very light-weight: it only makes use of hash functions and symmetric encryption and does not require expensive public-key operations. Consequently, the proposed scheme is efficient and by orders of magnitude faster than public-key schemes. In addition, it is quite undemanding in the sense that it does not impose any specific requirement on the network, such as on routing and network topology or encryption algorithms. The solution proposed in this paper involves three phases, to be carried out in the following order:

- *Key pre-distribution phase*: carried out before the deployment of the network, more precisely during the node's manufacturing time.
- *Network initialization phase*: comprises the very first steps required in order to setup the network's security, and it is performed during the network deployment.
- *Authentication protocol*: it is carried out every time a new node requests to join the network, once the previous phase is over.

Next, we analyse each of the three phases in detail.

2.1 Key Pre-Distribution Phase

During this phase the network manufacturer generates and securely stores a network-wide symmetric master key, k_M. During manufacturing time, each node is preloaded with an *initial authenticator*. An *ith cycle authenticator* ∇^i is an operator that can be used by a node to authenticate to another. The superscript indicates the cycle in which the authenticator is. It is composed of n tuples of random numbers and the result of applying a keyed-hash function with the current authentication key over them.

During the first cycle of authentication, just after the node manufacturing and before their deployment, the authentication key is equal to the master key, $k^0_{auth} = k_M$. In general, the authentication key in the jth cycle will be $k^j_{auth} = [k_M]^j$ and the authenticators set:

$$\nabla^j = \{(r_i, [r_i]_{k^j_{auth}})\},\ i = 0, ..., n-1.$$

The authenticator transits from one cycle to the next when the current set of tuples is about to be exhausted. This process is analyzed in detail in section 2.3.

2.2 Network Initialization Phase

This phase takes place during network deployment in the operational environment where every node discovers its neighbors in communication range. The steps to be performed by each node are:

1. Each node i generates its unique symmetric key, k_{enc}^{i}, called the *node encryption key*. This key is calculated by generating a random number, r_i, and performing $k_{enc}^{i} = [k_M, r_i]$. For example, the encryption key for node A would be calculated as: $k_{enc}^{A} = [k_M, r_A]$.
2. Each node broadcasts its random value, r_i, for a short period of time, that can be as short as a few seconds [1]. A typical value would be about a minute. In this way, an attacker who is listening to the broadcast traffic just obtains random values.
3. Each node receives the random values from its neighbor nodes and calculates their encryption keys by using the common master key. At this point, each node stores a list of pair-wise keys of its neighbors' nodes.
4. Each node hashes the common master key and saves it in the form of the first authentication key, $k_{auth}^{1} = [k_M]$. This is done because storing the master key in the node's memory is a great potential risk if the node is captured. This is the main reason of the existence of the *authenticator*, which provides a way to authenticate other nodes and to verify their knowledge of the common master key, without storing the master key itself.
5. As a result of this phase, each node stores its own encryption key, k_{enc}^{i}, the set of encryption keys of its neighbor nodes, the key for the next authentication cycle, k_{auth}^{1}, that is, the hash of the master key, and the current cycle authenticator, ∇^0, which is composed of a set of n tuples. The structure of an authenticator will be discussed in detail in section 2.3.
6. At this point, nodes can start to communicate with others using the pairwise encryption keys.

Memory Requirements As a result of the Network Initialization phase, each node stores a set of encryption keys and an authenticator operator. Taking into account the typical number of neighboring nodes in this kind of networks [1] and the key length we are considering (128 bits), the memory storage requirements of the proposed scheme appear affordable.

For example, for a high node density of 5 neighbors, key length of 128 bits, 160-bits output hash function, and a 10 tuples authenticator, the required memory is about 360 bytes only.

2.3 Authentication Operator

The *authentication operator* is used by the network nodes to mutually authenticate each other. The operator objective is to provide nodes with capacity to authenticate new nodes in the network, once the deployment phase is over. Recall that retaining the master key in the node memory could lead to an easy total compromise of the whole network. To avoid this, the idea behind the authentication operator is simple: the necessary authentication material, that is, the challenge/response tuples are pre-calculated with an authentication key that is later erased. Therefore, we are still able to authenticate new nodes, verifying their knowledge of the authentication key, without having it stored in memory.

The authentication operator makes use of two known cryptographic primitives, as the *challenge/response scheme* [4] and the *key chains* [3]. Its operation is somewhat similar to other existent schemes, as μTESLA [6], but it provides several advantages:

- It does not depend on a time-based *key disclosure mechanism*, so it does not need a base station nor a time synchronization mechanism.
- It uses a hash chain for authentication of new nodes, which we believe to be simpler than the use of a trusted third party S and the interchange of four messages, as in SPINS [5]. In comparison, the proposed scheme achieves the same goal by using a common master key and only one message.

Authenticator Generation The authenticator of an arbitrary jth cycle is constructed with the key material of the previous cycle, $j-1$. In this way, the node is able to prove the knowledge of the master key, because the authentication key of the cycle $j-1$ can only be derived from it, without storing the master key itself. If a node is compromised in this situation, the attacker only obtains the authenticator of the current cycle and is therefore not able to compromise the authentication and exchange of keys performed using previous cycles of the same authenticator.

As a node runs out of authenticators instances, it simply generates a new set, that is, it starts a *new cycle* of the authenticators set. The process to increase the jth cycle of the authenticators set is composed of the following steps:

1. Calculate a new authenticators set with n tuples of random numbers and apply the current authentication key, k^j_{auth}, to each of them, to obtain:
$$\nabla^{j+1} = \left\{ (r_i, [r_i]_{k^j_{auth}}) \right\}, \; i = 0, ..., n-1$$
2. Update the current authentication key, by hashing it and obtaining:
$$k^{j+1}_{auth} = [k^j_{auth}]$$
3. So the new key material is:
$$\left[k^{j+1}_{auth}, \nabla^{j+1} = \left\{ (r_i, [r_i]_{k^j_{auth}}) \right\} \right], \; i = 0, ..., n-1$$

Implementation issues Each one of these tuples has a *state label* associated with it, which describes the current state of the tuple in the authentication process. The possible values for this state are:

- *UNUSED* - the tuple has not been yet used.
- *ASSIGNED* - the tuple has been temporarily assigned to a node in an in-progress authentication process. The details can be found in the following section. If the process fails, then the state changes to UNUSED and the tuple is available for use again.

– *USED* - the tuple has been used in a successful authentication attempt and is no longer available to other processes. In this way, replay attacks can be avoided.

In addition to these labels, the authenticator structure has another field, called *current tuple index*, δ, which stores the first UNUSED tuple and is incremented by one every time a tuple changes its state from UNUSED to ASSIGNED.

2.4 Authentication Protocol

Once the initialization phase of the network is over, the nodes start communication by using the pairwise encryption keys. When a new node A wants to join the network after the deployment phase, a mutual authentication protocol based on a challenge/response scheme begins.

Node A is a *fresh* one, in the sense that it is the first time it joins the network. Therefore, its authentication operator is in the first cycle, ∇^1. Let B be the authenticator node, which can be in any arbitrary jth cycle. The protocol between A and B is then carried out as follows:

1. A produces a challenge for B by generating a random number, r_A. A then sends to B a message in the following format: $M_1 = r_A$.
2. B receives M_1 from A and performs the following operations:
   (a) It opens the first unused tuple, marked by the current tuple index, δ, of its authenticator and it extracts the corresponding random number, r_B, and its pair $[r_B]_{k_{auth}^{j-1}}$. Next, it changes the state label of the δ tuple from *UNUSED* to *ASSIGNED*.
   (b) It generates the response to the challenge issued by A by using the defined keyed-function with k_{auth}^{j-1} over the challenge r_A, obtaining $[r_A]_{k_{auth}^{j-1}}$.
   (c) It recovers its own encryption key, k_{enc}^B, and encrypts it with the current authentication key, obtaining $\{k_{enc}^B\}_{k_{auth}^j}$.
   (d) It includes the current cycle of its authenticator, j, in order to A to be able to synchronize later. Then it sends a message to A in the following format:
   $$M_2 = r_B, [r_A]_{k_{auth}^{j-1}}, \{k_{enc}^B\}_{k_{auth}^j}, j$$
3. A receives M_2 and performs the following operations:
   (a) It retrieves the current cycle j of B and calculates the difference with its own cycle. As A is a fresh node, so its current cycle will be 1. Therefore, A needs to perform $j-1$ hashes over k_M to obtain $k_{auth}^j = [k_M]^{j-1}$ and thus synchronize with B. More information on this step can be found in the Remarks section below.
   (b) It checks that the response from B is correct by comparing the result of its own computation with the received value. At this point, B has demonstrated knowledge of the original master key and has been successfully authenticated.

(c) It computes the image of the challenge r_B, obtaining $[r_B]_{k^{j-1}_{auth}}$.

(d) It generates its own encryption key, k^A_{enc}, and encrypts it with the current authentication key, obtaining $\{k^A_{enc}\}_{k^j_{auth}}$.

(e) Finally, it sends a message containing both values to B, in the following format:

$$M_3 = [r_B]_{k^j_{auth}}, \{k^A_{enc}\}_{k^j_{auth}}$$

4. Finally, B receives M_3 from A and executes the following actions:
   (a) It compares the response with the corresponding pair of the authenticator in use, ∇^j_k:
      i. If they are equal, it changes the state label of ∇^j_k from ASSIGNED to USED, the new node is successfully authenticated and access to the network is granted.
      ii. If they are not equal, the state label is changed to UNUSED again. At this point, the new node is unsuccessfully authenticated and no access to the network is granted.

The whole process can be summarized as follows:

$$\begin{aligned}&A \rightarrow B : r_A\\&A \leftarrow B : r_B, [r_A]_{k^{j-1}_{auth}}, \{k^B_{enc}\}_{k^j_{auth}}\\&A \rightarrow B : [r_B]_{k^j_{auth}}, \{k^A_{enc}\}_{k^j_{auth}}\end{aligned}$$

As it is can be observed in messages 2 and 3, this protocol also provides a key establishment procedure, by effectively transporting the appropriate encryption key, k^a_{enc} or k^b_{enc}, to the corresponding party.

3 Energy Consumption Evaluation

We now consider the energy consumption of our proposal by comparing it to the other protocols in a worst-case scenario. We use a N by N grid topology for Comparison and assume that each node can hear the data transmitted by the nodes immediately around it, which implies that most nodes have eight neighbors.

In this scenario the average transmissions for nodes in SPINS is about 8, while BROSK and our proposal only need one. However, each one of these transmissions has different lengths, which can significantly affect the final result.

In order to evaluate this effect, the total lengths of exchanged messages for each protocol were calculated, assuming nonce values of 64 bits, ID values of 14 bits, symmetric keys of 128 bits and HMAC values of 160 bits. As a result we obtain the message length of 632 bits for SPINS (which needs 8 of these messages, requiring a total of $8 \times 632 = 5056$ transmitted bits), 238 bits for BROSK and 78 bits for our proposal.

Therefore, it can be concluded that our proposal can save up to 98% and 67% of the required energy for message transmission by SPINS and BROSK, respectively.

4 Conclusions

In this paper, a light-weight authentication and key management protocol for sensor networks has been proposed. It uses symmetric cryptography with an encryption of only a few bytes to be performed by the claimant node once per authentication attempt.

This scheme meets all the requirements usually defined for sensor networks. Specifically, it provides a *perfect resilience against node capture* as, according to the metric defined in [2], due to the compromise of a node reveals no information about links that is not directly involved in. It also provides *node-to-node identity authentication*, as the nodes are able to verify the identities of the nodes they are communicating with and an adversary is unable to impersonate the identity of a node unless this has already been captured.

Additionally, also in the deployment phase our approach provides an improvement over other similar proposals in the literature, as the well-known key infection paradigm [1], which broadcasts encryption keys in the clear for a short period of time. Our scheme is secure against *an attacker which is present even before network deployment*, regardless the number of present attackers or their physical location.

Regarding the energy consumption and scalability, the proposal presents some improvements over the other proposals in the literature. Due to the smaller number and length of the exchanged messages, it can save up to 98% and 67% of the required energy by SPINS and BROSK protocols, respectively.

On the other hand, the scheme is specifically design to scale regardless the number of nodes in the network, requiring only one message to be interchanged. Finally, the memory requirements are small and affordable as well. They only depend on the number of neighbor nodes, and not on the total number of nodes in the network.

References

1. Ross Anderson, Haowen Chan, and Adrian Perrig. Key infection: Smart trust for smart dust. Proc. of ICNP'04, October 2004.
2. Haowen Chan, A. Perrig, and D. Song. Random key predistribution schemes for sensor networks. *Proceedings of Symposium on Security and Privacy, 2003*, pages 197–213, 2003.
3. Leslie Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772, November 1981.
4. Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography.* CRC Press, Inc., Boca Raton, FL, USA, 1996.
5. Adrian Perrig, Robert Szewczyk, J. D. Tygar, Victor Wen, and David E. Culler. Spins: security protocols for sensor networks. *Wirel. Netw.*, 8(5):521–534, September 2002.
6. Adrian Perrig, Robert Szewczyk, Victor Wen, David Culler, and J. D. Tygar. Spins: Security protocols for sensor networks. In *Wireless Networks*, pages 189–199, 2001.
7. Frank Stajano. *Security for Ubiquitous Computing.* John Wiley and Sons, February 2002.