Matrix-based key management scheme for IoT networks

Mohammed Nafi, Samia Bouzefrane, Mawloud Omar

Matrix-Based Key Management Scheme for IoT networks

Mohammed NAFI[a], Samia BOUZEFRANE[b], Mawloud Omar[c]

[a] *Laboratoire d'Informatique Médicale, Faculté des Sciences Exactes, Université de Bejaia,06000 Bejaia, Algérie*
[b] *CEDRIC Lab, Conservatoire National des Arts et Métiers -CNAM, Paris, France*
[c] *LAMOS, Faculté des Sciences Exactes, Université de Bejaia,06000 Bejaia, Algérie*

Abstract

The key management is the central element of network security. In fact, key distribution is necessary for securing applications in the context of Internet of Things (IoT). However, existing key management protocols are not directly applicable on IoT due, among other things, to severe and high resource constraints of some devices that make up the IoT network. Therefore, it is necessary that the proposed key management protocols takes in charge these constraints.

In this paper, we propose a new lightweight Matrix based key management protocol for Iot network. The formal verification tool AVISPA has been used in order to check these security properties like authentication, integrity and secrecy.

Security and performance analysis show that the proposed scheme protects user's sensitive data from several types of attacks by achieving secure end-to-end communication, and is suitable for resource-limited networks.

Keywords: Key management, Security, Internet of Things, Dynamic networks

1. Introduction

The concept of Internet of Things (IoT) was initially introduced in 1999 [1]. It is a system that interconnects variety of heterogeneous devices. The devices, like RFID tags, sensors, actuators for instance, have the ability to sense their environment by acquiring measurements, to comunicate with each others by exchanging data over the network, and to process the gathered data, etc. The interaction between devices is performed without human intervention [2]

To make the communication between these IoT devices more secure, a key management service is needed. However, key management in the context of IoT is more challenging than in traditional networks. This is because of many reasons, such as the use of wireless links that are vulnerable to eavesdropping attacks; lack of a central authority; mobility of some IoT devices and heterogeneity of devices in terms of resources such as memory, computing and energy capacity and bandwith availability; high resource constraints of some devices, like sensor that have limited battery, computation and memory capacity, etc. For all these reasons, integrating security for IoT is a real challenge and new key management solutions must take in charge these inherent features. In this paper, we propose a new key management scheme for Iot that is based on Matrix.

The main contributions of this paper are the following:

- The number of keys to be stored in a node's memory is very small since a node stores at most only the keys of its direct neighbors;
- The approach offers a negligeable communication overhead. In other words, no much communication is requiered during key establishment phase because the keys are generated in a distributed manner.
- Low computation cost because no complex operation is requiered to estabilish secret keys so the computation of symmetric keys betwen two neighbors is efficient.

- The security goals such as secrecy, integrity and authentication are guaranteed.
- The proposed scheme is resistant to several types of attacks such as eavesdropping, compromising, sybil, forward and backward and replay attacks.
- The secret keys are estabilished in a distributed manner. therefore, it avoids a central entity that could be a point of failure or weakness.
- The security analysis and performance evaluation show that the proposed scheme can protect user's data privacy and is suitable for the resource-limited network.

The organization of the rest of this paper is as follows. Related work on IoT security and key management is summarized in Section 2. In Section 3, we present the proposed matrix based key management scheme, which consists of a set of protocols for initialization phase, key establishment phase, node addition phase, key revocation phase and periodic key renewal phase. In Section 4, we describe the security, analytical and performance analysis of our protocol. Finally, Section 5 concludes the paper and gives future directions.

2. Related work

According to [3], the key management protocols proposed in the literature for IoT networks can be classified into two main categories: (1) preshared approaches, which are based on the predistribution of shared context used to generate a secret shared key between the two communicating entities; and (2) public key approaches, which are based on asymmetric encryption to generate a commun secret between two entities having no previous preshared context.

Among the schemes belonging to the first category, we find a matrix-based scheme for establishing pairwise keys that was initially proposed by Blom [4]. This scheme allows any pair of nodes to be able to compute a commun pairwise key. It is based on the use of a set of matrices : a $(\lambda + 1) \cdot N$ public matrix called G, known to everyone, a $(\lambda+1)\cdot(\lambda+1)$ random symmetric matrix called D, and a $N \cdot (\lambda+1)$ secret matrix called $A = (D \cdot G)^T$. Since A is a symmetric matrix, the key matrix $K = A \cdot G$ is also symmetric. Therefore, the element (i, j) of K is equal to the element (j, i) that correspond to the pairwise key shared betwen the nodes i and j. Every node i maintains in its memory the ith row and ith column of the private and public matrix respectively as key material. When two nodes i and j want to communicate, they first exchange their public column vector of G then each node calculates separately the same commun key $K_{ij} = K_{ji}$, which is obtained by the product of the private row vector of A of one node and the public column of the other node. In this scheme, since nodes never exchange their private rows, no adversary can compute any pairwise key by simply listening to the communications. In addition, it has been shown in [4] that the above scheme is λ-secure when all the columns of the matrix G are linearly independent. However, if the number of compromised nodes exceeds the threshold λ, the whole secret matrix K could be calculated, thus the entire network becomes unsecure.

Based on the above Blom's scheme, Du et al. [5] proposed a new key pre-distribution scheme that uses multiple-space key in order to acheive better resilience of the network to node capture attack. In fact, to be able to break at least one key space, an adversary must capture a significant number of nodes equals to $\lambda + 1$ that all share the same key space's information. In this scheme, every node needs to store a row vector of $\lambda+1$ elements of its private information for each selected key space and a single element (a seed) of its public information in its memory. To be able to compute a commun pairwise key, two nodes must share at least one key space. Compared to Blom's scheme, this scheme is more resilient to node capture attack and more efficient in terms of communication, but less in terms of computation and storage. In fact, nodes do not need to exchange their public column, which reduces the communication overhead but it is regenerated by the node itself, which increases computation overhead (2 modular multiplications). Moreover in blom's scheme, a node needs only to store $2(\lambda + 1)$ elements but here it stores a seed and

$\lambda + 1$ elements for each key space, so $(\lambda + 1) \cdot t$, where t is the number of key spaces. Furthermore, this scheme does not address node addition, revocation and key refresh.

Yu and Guan in [6]. proposed a key management scheme using deployment knowledge based on Blom's scheme. In their scheme, a deployment area is partionned into grids (hexagons, squares or triangles) and sensor nodes are arranged into groups where the number is the same as that of grids. After that, each group of sensors is deployed into a single grid. The authors show that hexagonal grids are best in terms of security and memory requirement compared with the others. During key predistribution phase, all groups share the same global matrix G and each one is assigned a unique secret matrix A and a set of B matrices. During discovery phase, each pair of neighbors that belong to the same group, generate the pairwise key from the common matrix A and G as in Blom's scheme. The nodes that are not from the same group but share at least one common B matrix, can also compute a common key from one choosen matrix B and the matrix G using Blom's scheme too. The other nodes that can not compute pairwise keys between them may use another key discovery mechanism to establish pairwise keys.
In this scheme, each node stores one column of matrix G, one row of matrix A, and, at most, w rows of B matrices, where each row has $\lambda+1$ elements. Moreover, if more than λ nodes of a group are compromised, the matrix A and some B matrices will be broken. But their scheme achieves a higher connectivity with a much lower memory requirement and a shorter transmission range. It is more resilient against node capture attacks.

Another scheme based on Blom's scheme have been proposed by Rahman et al. [7]. The authors have improved the original Blom's scheme so that it becomes suitable for use in resource-constrained environment like wireless sensor networks. In this scheme, two nodes are able to generate a common key without any exchange of messages between them but only by knowing the identifier of the other. In fact, a node does not need to store or exchange its public column of the matrix G, but generates it by using the node's identifier, which significantly reduces the storage and the communication overhead. In addition, the authors proposed mechanisms for updating keys, adding new nodes after deployment, and revoking compromised nodes. They also presented a dynamic mechanism for establishing secure group keys that uses pairwise keys. This makes their model more flexible. However, the public matrix G is generated by the node itself, which increases the computation overhead on the node side, and affects the resilience of the scheme to node capture attack. Indeed, when an attacker compromises a single node, he is not only able to compute all its keys shared with the other nodes, but also obtains more information about the secret matrix.

The authors Y. Zhang et al in [8]. have presented a matrix-based cross-layer key establishment protocol for smart homes with no prior secret sharing. This is motivated by the fact that domestic devices are heterogeneous and are not necessarily produced by the same factory. The protocol is based on [9] and uses the multiple key-spaces idea which is proposed in [5]. Two kinds of keys are used in this protocol: the secret master key ki and the secret session key Kij. The first key is extracted by using the physical layer key extraction algorithm [10] when the device joins the network, and shared between a device and the home gateway. The second one is established at the higher levels and shared between two appliances Pi and Pj . Thus, the proposed protocol allows any pair of devices to be able to compute a common secret session key with light energy consumption by delegating the heavy operations to the home gateway that is powerful. It also achieves key refresh and network scalability. In addition to this, the storage cost is very low since home appliances have no need to pre-load any secrets and store only the key seed sent by the home gateway. However, each appliance needs to compute $2\lambda + 1$ multiplications to establish a session key and the communication cost of the proposed protocol is relatively higher compared with the previous ones.

Messai and Seba in [11]. presented a new key management scheme for hierarchical wireless sensor networks called EAHKM+. This scheme is composed of two phases: the key pre-distribution phase and the cluster formation and key establishment phase. In the first phase, each sensor node is pre-assigned with three keys before its deployment in the sensing environment: the network key known to all sensor nodes in the network, which is deleted after the second phase, and two pairwise keys used to secure the

communication channel that separates sensor node from the base station. In the second phase, clusters are securely formed and two new keys are established: the cluster key shared between all cluster members and a pairwise key shared between the sensor node and its cluster head. This scheme provides a secure cluster creation and it is flexible since it allows adding and removing sensor nodes after deployment. In addition to this, it is scalable and can support a large number of nodes. However, the rekeying process is costly in terms of communication and computation since it requies the re-run of clustering algorithm.

To overcome the limitations of current key distribution and management schemes, we propose an efficient and a new matrix-based key management scheme.

3. Overview of the proposed scheme

In this section, we describe our matrix-based key management system for Iot networks that aims to minimize the computation, communications and storage overhead. First of all, we begin by presenting the network model that we are considering, then we give some assumptions. After that, we summarize the notations used in this paper. Finally, we describe in detail the differents phases of our scheme.

3.1. Network model

We adopt the network architecture that consists of three main components: constrained nodes, gateway nodes and remote server node (command node), as shown in Figure 1.

3.1.1. Constrained nodes

In this category, we can find all the nodes that are highly constrained in terms of resources (energy, memory and computation), such as sensors, RFID Tags, wearable devices (like watch) that can be carried by a human, etc. The role of these devices consists of monitoring or sensing the environment so that they collect and transmit the collected data to the Gateway nodes via bluetooth, ZigBee or wifi technologies. For example, in healthcare application, sensors can be planted in or on a human's body in order to collect health related data (e.g. blood pressure, blood glucose level, temperature level, etc.). Another examples of constrained nodes are MICAz and TelosB that are really constraining. Both platforms run TinyOS. The MICAz is based on the low-power 8-bit microcontroller ATmega128L running at 7.37 MHz and the TelosB is based on the 16-bit microcontroller running at 4 MHz [12].

3.1.2. Gateway nodes

The gateway nodes have significantly higher energy resources and are equipped with high performance processors and more memory compared to nodes belonging to the previous category (constrained nodes), but they have less resources when compared to remote server node. The gateway nodes fuse the received data collected by the different constrained nodes, process and send or forward it to the remote server (command node). The communication between the gateway node and the constrained nodes may be via Bluetooth or ZigBee, and the one between the gateway node and the remote node may be done through cellular (3G, 4G) or WiFi networks.

3.1.3. Remote server node (command node)

It can be assumed that the server node has no limitations in terms of power, computation and storage compared to the two previous ones. For instance, the medical specialists or doctors at remote server node side can continuously follow patient's health status based on the data received in order to intervene in time.

3.2. Assumptions

- We have not made any trust assumptions about the constrained nodes. They can therefore be captured and compromised. This means that the adversary can read all the information from the node's memory, including the keys.

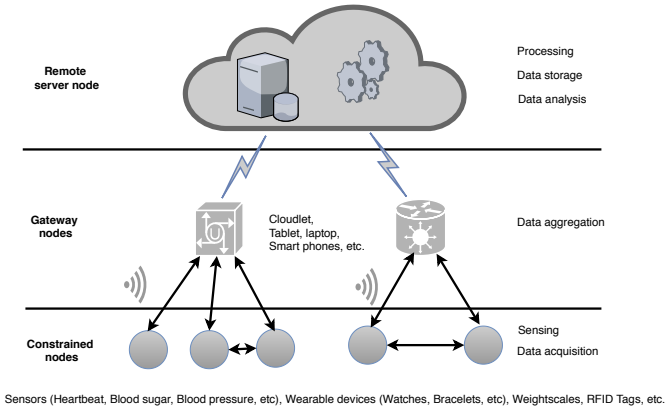

Figure 1: Network Model

- Constrained nodes are only able to perform symmetric encryption to avoid high computations since they are limited in resources.
- We assume that the gateway nodes can also be captured by the attacker and are supposed to be able to perform symmetric and asymmetric encryption. So, we propose to include TPM (Trusted Platform Module)[13, 14] to secure the keys stored in their memory.
- The remote server node is powerful enough to support symmetric and asymmetric cryptography.

The notations used in this paper are listed in Table 1.

Notation	**Description**
Idi	Identification number of the node Ni.
M	Square matrix of order n, where n refers to the size of the network during the deployment
M_i	Matrix of the node Ni
vi	Neighbors vector of the node Ni
$a \parallel b$	Information a is concatenated with b's one
$Hash(Msg, k)$	One way keyed-hash function
$\|x\|$	Absolute value of the number x
$det(M)$	Determinant of the matrix M
S_{ij}	Positive secret value computed by two neighbors Ni et Nj
K_{ij}	Pairwise secret key shared betwen two neighbors Ni etNj
K_i/K_i^{-1}	Public/Private key of the gateway node Ni
Si	Random positive secret value generated by the gateway GNi
$\{M\}\,k$	Message M is encrypted with the key k
Ti	Timer of the node Ni
$Nonce_i$	Nonce (random value) generated by the node Ni
$List_i$	List kept by the node Ni

Table 1: Notations

3.3. Phases of our scheme

The proposed scheme is composed of fives phases: initialization, key establishment, adding new node, key revocation and key refresh phase. These phases are described in detail in the following section.

3.3.1. Initialization Phase

During this phase, all of the following information is preloaded into the node's memory just before they are being deployed:

1. An identification number Idi that is unique in the network.
2. A square matrix M of order n, where the elements are generated randomly and are strictly positives, and n represents the number of nodes in the network during the deployment phase: $M[i,j] > 0$, with $i,j = 1..n$. This matrix can be stored in the flash RAM and hence will be erased from the node's memory later.
3. A one way keyed-hash function $Hash(msg,k)$ that takes as input a message of arbitrary length msg and a secret key k and compresses the message into a short fixed length hash value, so it will take up less space.

With the Matrix M and the hash function $Hash()$, every node Ni is able to compute the symmetric key K_{ij} that will be shared with its direct neighbor Nj. Each gateway node has a timer initialized to a value which decrements with time. When it reaches zero, the key renewal process will be launched.

3.3.2. Key establishment phase

The key generation phase contains the two following steps:

a) Neighbor discovery

As soon as the deployment is done, every node Ni identifies its direct neighbors (1-hop) by sending them a Helloi message along with its identifier, a nonce and a hash value (digest), which is computed by the preshared one way function that uses the diagonal element of the sender as secret value.

$$Ni \rightarrow GNj : Helloi, Idi \parallel Nonce_i, Hash(Idi \parallel Nonce_i, M[i,i]) \tag{1}$$

When a node that is within the radio range of the sender, receives this message, it checks its authenticity and integrity by calculating the hash value H of the first part of the Helloi message by using the diagonal element $M[i,i]$ of the sender as secret value as follows: $H = Hash(Idi \parallel Nonce_i, M[i,i])$. The obtained result will then be compared to the second part ie $Hash(Idi \parallel Nonce_i, M[i,i])$, which corresponds to the hash value computed by the sender node Ni. If both digests are different, then this message is ignored and rejected, oherwise it is authentic, so it has not been altered by an adversary during its transmission. As a result, this message will be accepted by the recipient node GNj and the latter updates its neighbors vector vj by adding the identifier of $Ni(Idi)$ and arranges the verctor in ascending order. After that, the gateway node replies to the sender node Ni with the Helloj message that contains its identifier, the succesor of the received nonce and a digest.

$$GNj \rightarrow Ni : Helloj, Idj \parallel Nonce_i + 1, Hash(Idj \parallel Nonce_i + 1, M[j,j]) \tag{2}$$

At the end of this step, each node has its neighbors vector completely updated that contains all the identifiers of its 1-hop neighbors arranged in ascending order.

b) Computation of the secret values and pairwise keys

During this step, each node computes its square matrix M_i, which is obtained from the initial one M by keeping only the elements $M[k,l]$, where $\forall \quad k,l \in v_i$.

Moreover, it computes a positive secret value S_{ij}, which is equal to the absolute value of the determinant of the 2 by 2 matrix Mij, where the elements are:

$$Mij = \begin{bmatrix} M_i[i,i] & M_i[i,j] \\ M_i[j,i] & M_i[j,j] \end{bmatrix} = \begin{bmatrix} M_j[i,i] & M_j[i,j] \\ M_j[j,i] & M_j[j,j] \end{bmatrix}, \quad with \quad i < j$$

$$S_{ij} = |det(Mij)| \tag{3}$$

Finally, it obtains a shared secret key K_{ij} by applying the hash function that uses S_{ij} as secret value as follows:

$$K_{ij} = Hash(Idi \parallel Idj \parallel S_{ij}, S_{ij}), \quad with \quad i < j \tag{4}$$

This pairwise key will be used by the two nodes in order to secure their communications by encrypting and decrypting the messages exchanged between them. Note that keys can or not be stored in node's memory according to the capacity of the node. In fact, a node with low storage and high processing capacity may store few or no keys. The other keys will be calculated during communication, which reduces storage costs. On the other hand, a node with high storage and low processing capacity can store all or a large number of pairwise keys. This allows the node to avoid recalculating them during the next communication session with the same node, which reduces the calculation cost. Thus, a node can store a set of keys whose number is between zero and d where d is the number of its neighbors.

In the other hand, each gateway node GNi computes its private/public keys K_i^{-1}/K_i that will be used to secure the communication channel separated him to the server node. The private key is computed as follows:

$$K_i^{-1} = Hash(S_{ij} \parallel S_{ik} \parallel ... \parallel S_{il}, M[i,i]) \tag{5}$$

where $j, k, .., l \in v_i$ and $j < k < .. < l$.
The public key is obtained from the last private key as follows:

$$K_i = Hash(K_i^{-1}, M[i,i]) \tag{6}$$

The key K_i is public and known by everyone. It can be specially used by the server node when he wants to send messages securely to that gateway node. However, the key K_i^{-1} is private and must be kept secret and known only by the gateway node itself. It can be used to sign/decrypt messages sent/received to/from the server node. The server node sends its public key encrypted with the public key of each gateway node. This latter uses its private key K_i^{-1} to decrypt the messages encrypted with its public key K_i.

The complete process of establishing of the symmetric key between the gateway node j and its direct neighbor i is described in the Figure 2 below.

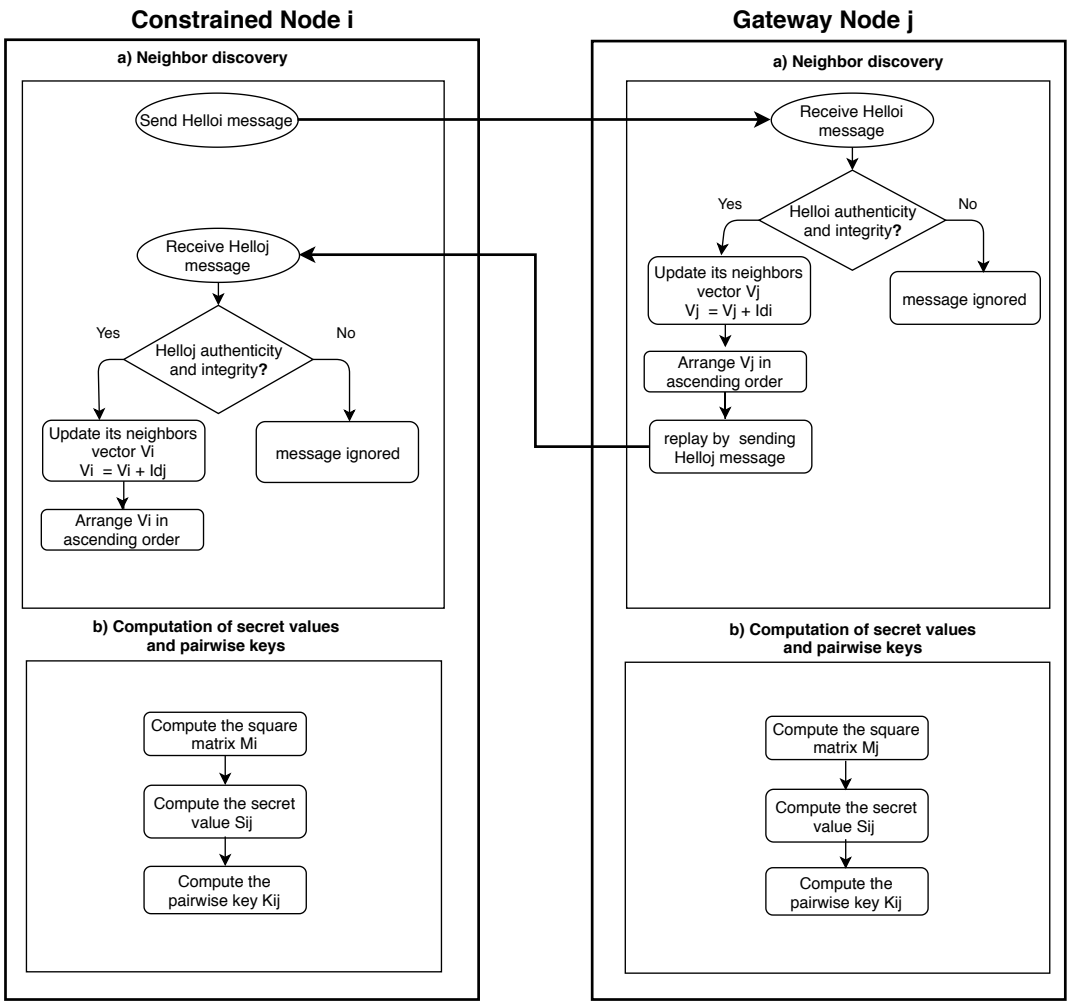

Figure 2: Process of establishing of the symmetric key between two nodes

Illustrative example

The network depicted in the Figure 3 is composed of four nodes: three sensors and one gateway node. During the initialization phase, all nodes are assigned a unique identifier(number) and pre-loaded with the hash function $Hash()$ and the square matrix M of order 4 (a 4 by 4 matrix).

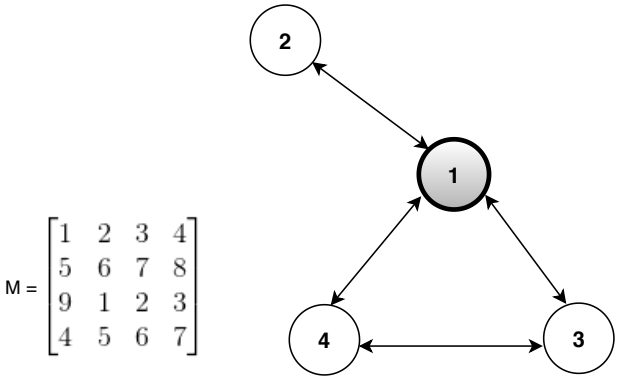

Figure 3: Network with four nodes

During the neighbor discovery stage, the nodes of the network exchange hello messages betwen them so that they update their neighbors vector and compute their square matrix as follows:
The gateway node 1 has three neighbors (nodes 2, 3 and 4). Therefore, its neighbors vector $v1$ has four identifiers arranged in ascending order (its identifier and the identifier of each neighbor). So the vector $v1 = [1, 2, 3, 4]$ and its square matrix $M1$ is equal to the initial matrix M because all the nodes are neighbors with the node 1.
The node 2 has one neighbor (node 1). Hence, its neighbors vector $v2$ has two identifiers (the identifier of the node 1 and its identifier in this order) arranged in ascending order $v2 = [1, 2]$ and its square matrix $M2$ is obtained as follows:

$$M2 = \begin{bmatrix} M[1,1] & M[1,2] \\ M[2,1] & M[2,2] \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 5 & 6 \end{bmatrix}$$

The nodes 3 and 4 have the same neighbors (nodes 1, 3, 4), thus they have the same neighbors vector $v3 = v4 = [1, 3, 4]$ and they share the same 3 by 3 matrix ($M3 = M4$) obtained from the matrix M as follows :

$$M3 = M4 = \begin{bmatrix} M[1,1] & M[1,3] & M[1,4] \\ M[3,1] & M[3,3] & M[3,4] \\ M[4,1] & M[4,3] & M[4,4] \end{bmatrix} = \begin{bmatrix} 1 & 3 & 4 \\ 9 & 2 & 3 \\ 4 & 6 & 7 \end{bmatrix}$$

In the next step, each pair of neighbors compute the secret value in the following way:
The nodes 1 and 2 compute the secret value:

$$S12 = |det(M12)| = \left| det \begin{pmatrix} M[1,1] & M[1,2] \\ M[2,1] & M[2,2] \end{pmatrix} \right| = \left| det \begin{pmatrix} 1 & 2 \\ 5 & 6 \end{pmatrix} \right| = |6 - 10| = 4$$

The nodes 1 and 3 compute the secret value:

$$S13 = |det(M13)| = \left| det \begin{pmatrix} M[1,1] & M[1,3] \\ M[3,1] & M[3,3] \end{pmatrix} \right| = \left| det \begin{pmatrix} 1 & 3 \\ 9 & 2 \end{pmatrix} \right| = |2 - 27| = 25$$

The nodes 1 and 4 compute the secret value:

$$S14 = |det(M14)| = \left| det \begin{pmatrix} M[1,1] & M[1,4] \\ M[4,1] & M[4,4] \end{pmatrix} \right| = \left| det \begin{pmatrix} 1 & 4 \\ 1 & 7 \end{pmatrix} \right| = |7 - 4| = 3$$

The nodes 3 and 4 compute the secret value:

$$S34 = |det(M34)| = \left| det \begin{pmatrix} M[3,3] & M[3,4] \\ M[4,3] & M[4,4] \end{pmatrix} \right| = \left| det \begin{pmatrix} 2 & 3 \\ 6 & 7 \end{pmatrix} \right| = |14 - 18| = 4$$

Finally, in the last stage, every two neighboring nodes are able to generate the common symmetric secret key as follows:
The nodes 1 and 2 compute the secret shared symmetric key:
$K12 = Hash(1 \parallel 2 \parallel 4, 4) = Hash(124, 4)$
The nodes 1 and 3 compute the secret shared symmetric key:
$K13 = Hash(1 \parallel 3 \parallel 25, 25) = Hash(1325, 25)$
The nodes 1 and 4 compute the secret shared symmetric key:
$K14 = Hash(1 \parallel 4 \parallel 3, 3) = Hash(143, 3)$
The nodes 3 and 4 compute the secret shared symmetric key:
$K34 = Hash(3 \parallel 4 \parallel 4, 4) = Hash(344, 4)$
The gateway nodes 1 also computes its private/public keys as follows:
$K_1^{-1} = Hash(S_{12} \parallel S_{13} \parallel S_{14}, M[1,1]) = Hash(4253, 1)$, $K_1 = Hash(K_1^{-1}, M[1,1])$.

3.3.3. Adding new node Phase

To add a new node Nn to the network, the gateway node Ni that is close to this new node, randomly generates a new positive secret value Si and sends it to each of its neighboring nodes encrypted with the corresponding pairwise key. Each node that receives this message, decrypts it with the appropriate symmetric key, extracts and saves the secret value, and encrypts a message in which it inserts the new secret value Si and transmits it to its direct neighbors except to the one who sent him the last message. This process is repeated until all the nodes obtain this new value. This value and the hash function $Hash()$ are also stored in the memory of the new node Nn before deploying it in the network so that they will be used to calculate the secret keys that the new node will share with its neighbors.

$$Ni \rightarrow Nj : \{Si, Nonce_i\}\, K_{ij} \tag{7}$$

where K_{ij} is the pairwise key shared between the gateway node Ni and its neighbor node Nj.
After deployment, the new node sends a join message along with its identifier, a random number (nonce) and a digest.

$$Nn \rightarrow * : Join, Idn \parallel Nonce_n, Hash(Idn \parallel Nonce_n, Si) \tag{8}$$

The adjacent devices that are within the radio range of the new node, receive it then check its authenticity and integrity. If this verification fails, the join message will be ignored, otherwise all the neighboring nodes update their neighbors vectors by adding the identifier Idn of the new node and their matrices are also updated by creating, at the end, a new row and a new column, where the elements are equals to zero except the main diagonal element that equals to $M_j\,[k+1, k+1] = Nonce_n$, with k represents the order of the square matrix M_j just before adding the new node. After that, each neighboring node Nj will respond with an acknowledgement message that contains its diagonal element $M\,[j, j]$ of the matrix M.

$$Nj \rightarrow Nn : Ack, Idj \parallel Nonce_j \parallel M\,[j,j] \parallel Nonce_n + 1, Hash(Idj \parallel Nonce_j \parallel M\,[j,j] \parallel Nonce_n + 1, Si) \tag{9}$$

Upon the reception of the last message, the new node checks it. If this verification is successful, the new node will update its neighbors vector by adding the identifier of node Nj. When this neighbor vector is completly updated, the new node updates its square matrix in the following way: $Mn[i,j] = M[vn[i], vn[i]]$, with $i = j$, $Mn[i,j] = 0$, otherwise.

When the matrix Mn is fully updated, the new node can start to generate all the symmetric keys Kni that will be shared with each of its neighbor Ni by executing the step b) of the key establishment phase. After that, it replies to the node Nj by sending a message containing the succesor of the received nonce $Noncej$ and encrypted with the obtained pairwise key Knj.

$$Nn \rightarrow Nj : \{Nonce_j + 1\}\, Knj \tag{10}$$

At the end, the node Nj responds to the gateway node Ni by sending the message that contains the succesor of the nonce of Ni encrypted with the pairwise key K_{ij}.

$$Nj \rightarrow Ni : \{Nonce_i + 1\}\, K_{ij} \tag{11}$$

The process of adding a new node to the network is described in the Figure 4 below.

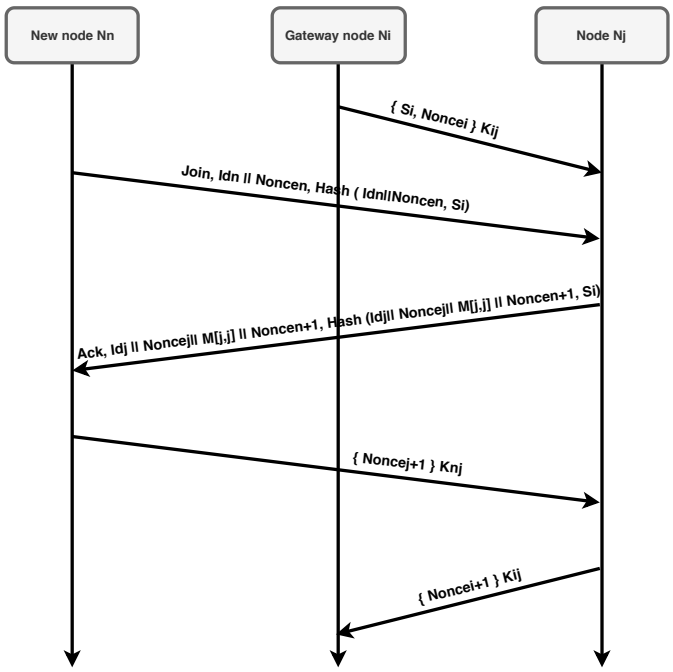

Figure 4: Process of adding a new node to the network

Example

The Figure 5 shows an example of adding a new node to the network. The network is initially composed of six nodes. The gateway node 1 and the node 2 are neighbors to each others and have the same neighbor (node 3), therefore, they share the same neighbors vectors $vi = [1, 2, 3]$, with $i = 1, 2$, and the same square Matrix $M_i = [1 \quad 2 \quad 3; 4 \quad 5 \quad 6; 7 \quad 8 \quad 9]$, with $i = 1, 2$. The node 4 wants to join the network. So, the gateway node 1 close to this new node, generates a random secret value $S1$ and sends it to its neighboring nodes 2 and 3 encrypted with the corresponding pairwise keys $K12$ and $K13$ respectively. The new node 4 is preloaded with this secret value $S1$ and a hash function $Hash()$ before its deployment then it generates a random number($Nonce4$ equals to 55 for example) and sends a same join message to its 1 hop neighbors(node 1 and 2). The nodes 1 and 2 check the authenticity of the join message by computing the hash value of the first part of the message and compares it with the second part. If the verification succeed, then both nodes add the identifier 4 to their neighbors vectors $vi = [1, 2, 3, 4]$, with $i = 1, 2$. After that the nodes 1 and 2 add a new forth row where $M_i(4, j) = 0$, with $i = 1, 2$ and $j = 1, 2, 3$ and $M_i(4, j) = Nonce4 = 55$ else (with $i = 1, 2$ and $j = 4$) and a new forth column where $M_i(j, 4) = 0$, with $i = 1, 2$ and $j = 1, 2, 3$ and $M_i(j, 4) = nonce4 = 55$ else (with $i = 1, 2$ and $j = 4$) . The nodes 1 and 2 respond with the acknowledgement messages containing the diagonal element $M(1, 1) = 1$ and $M(2, 2) = 5$, respectively. Upon the reception of the ACK messages, the new node 4 ckecks if they are authentic. if yes then it adds the identifiers 1 and 2 to its neighbors vector $v4 = [1, 2, 4]$ ordered and updates its square matrix $M4 = (1 \quad 0 \quad 0; 0 \quad 5 \quad 0; 0 \quad 0 \quad 55)$. The new node 4 has its 3 by 3 matrix fully updated, so it is able to generate a symmetric keys $K14$ and $K24$ by executing the step b) of the key estabilishment phase as follows:

The node 1 and 4 compute the secret value:

$S14 = |det(M14)| = |det(1 \quad 0; 0 \quad 55)| = |55| = 55$, and then generate the common key: $K14 = Hash(1 \parallel 4 \parallel 55, 55) = Hash(1455, 55)$.

The node 2 and 4 compute the secret value:

$S24 = |det(M24)| = |det(5 \quad 0; 0 \quad 55)| = |275| = 275$, and then compute the common key: $K24 = Hash(2 \parallel 4 \parallel 275, 275) = Hash(24275, 275)$

The new node 4 sends to node 1 and 2 the message containing the succesor of the nonce of node 1 and 2 encrypted with the corresponding pairwise key $K14$ and $K24$ respectively.

Finally, the node 2 replies to the gateway node 1 with the message containning the successor of the received nonce encrypted with the symmetric key $K12$.

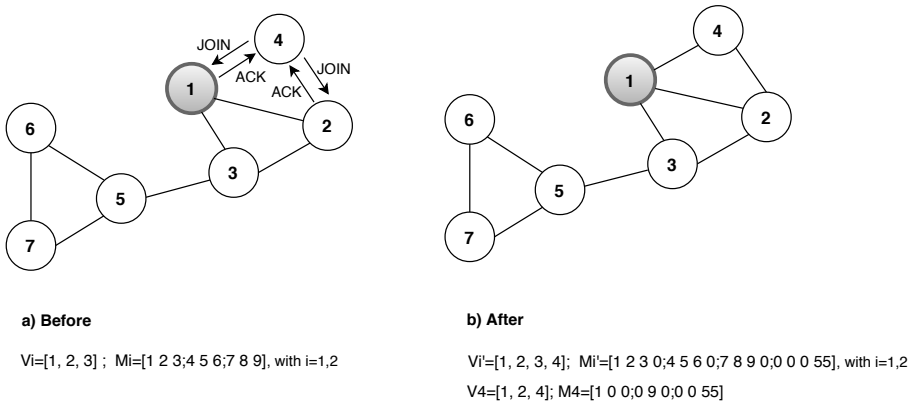

Figure 5: Example of adding the node 4 to the network

3.3.4. Key revocation phase

Two cases may arise: either the node leaves after it has been detected as compromised, or it leaves with its proper willingness.

Case 1 :

A node Ni that detects a bad or suspicious behavior (does not forward correctly messages for example) from its neighboring node Nj does the followings tasks:

a) informs its neighboring nodes Nk by sending a message encrypted with the pairwise keys that it shares with them. The message contains the identifier of the compromised node and a nonce.

$$Ni \rightarrow Nk : \{Idj \parallel Nonce_i\} K_{ik} \tag{12}$$

b) deletes from its memory the pairwise key that it shares with the compromised node.

c) updates its square matrix by deleting the row and the column that correspond to the compromised node.

d) updates its neighbors vector by removing the identifier of the compromised node.

When the neighboring node receives this message that comes from its neighbor, it does the following actions:

1. decrypts it with the appropriate pairwise key.
2. compares the identifier received with its identifier and with the identifier of the sender. If it is different from both identifiers then it checks if the compromised node belongs to its neighbors. In other words, it verifies whether the identifier of the compromised node appears in its neighbors vector. If it is not the case, the message will be ignored by the recipient, otherwise the node does the operations *b*), *c*) and *d*) described above.
3. replies to the sender by an acknowledgment message *Ack* encrypted with the appropriate symmetric key that contains the successor of the received nonce.

$$Nk \rightarrow Ni : \{Ack, Nonce_i + 1\} K_{ik} \tag{13}$$

Case 2 :
A node Ni that leaves the network with its own will, sends a Leave message to inform its 1-hop neighbors Nj.

$$Ni \rightarrow Nj : \{Leave, Idi \parallel Nonce_i\} K_{ij} \tag{14}$$

A neighbor who receives that message decrypts it with the appropriate pairwise key, responds with an acknowledgment message *Ack* and carries out the operations *b*), *c*) and *d*) described above.

$$Nj \rightarrow Ni : \{Ack, Idj \parallel Nonce_i + 1\} K_{ji} \tag{15}$$

All the neighboring nodes delete the pairwise keys that they share with the outgoing or compromised node. As a result, the latter can no longer communicate with its neighbors afterwards. In other words, the neighbors of the leaving or compromised node will avoid to transmit messages to him since they dont hold appropriate symmetric keys.

Example
We consider the network depicted in the Figure 6 below. The node 2 leaves the network with its own will, so it informs its neighboring nodes (node 1 and 4) by sending a *Leave* message encrypted with the appropriate pairwise key. In fact, the message $\{2 \parallel Nonce_2\} K12$ is sent to the node 1 and the message $\{2 \parallel Nonce_2\} K24$ is sent to the node 4.

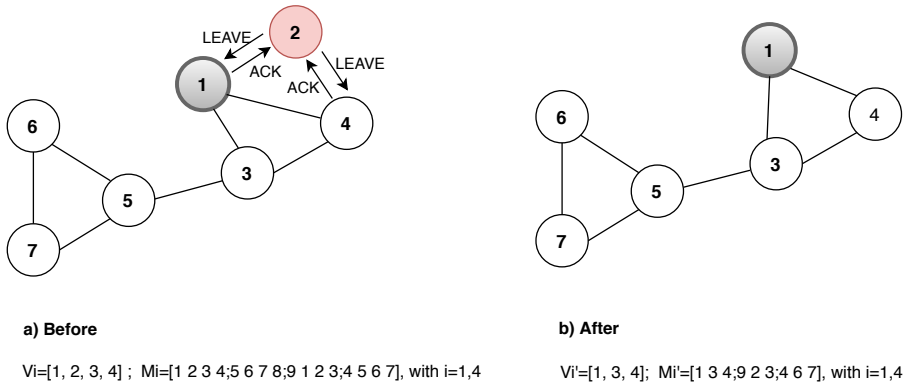

Figure 6: Example of Node 2 leaving the network

Before the departure of the node 2, we assume that the neighbors vectors and the square matrices of the node 1 and 4 are equals to:

$$v1 = v4 = \begin{bmatrix} 1, & 2, & 3, & 4 \end{bmatrix}; M1 = M4 = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{bmatrix}$$

When receiving the *Leave* message, the nodes 1 and 4 do the following operations:
- decrypt the message with the appropriate keys $K12$ and $K24$ respectively.
- reply to the leaving node by an *Ack* messages: $\{Ack, 1 \parallel Nonce_2 + 1\} K12$ and $\{Ack, 4 \parallel Nonce_2 + 1\} K24$ respectively.
- delete from their memories the pairwise keys $K12$ and $K24$ respectively.
- update their square matrices $M1$ and $M4$ by deleting the second row and the second column that correspond to the node 2.
- update their neighbors vectors $v1$ and $v4$ by deleting the identifier 2 of the node 2.

After the deletion of the node 2, the neighbors vectors and the square matrices of the nodes 1 and 4 become:

$$v'1 = v'4 = \left[\ 1, \quad 3, \quad 4\ \right]; M'1 = M'4 = \begin{bmatrix} 1 & 3 & 4 \\ 9 & 2 & 3 \\ 4 & 6 & 7 \end{bmatrix}$$

3.3.5. Key refresh phase

This phase aims to improve security because the new keys will be completly different from the old ones, which add difficulties to cryptanalytic attack. In fact, if the same keys are kept for a long time, the attacker may be able to obtain these keys by traffic analysis. Therefore, all the keys should be updated. The refresh process is initiated periodically by the gateway nodes after the expiration time is reached. This process can be devided into two subphases that are detailled as follows:

a) The goal of this first subphase is that the neighboring gateway nodes agree about a small secret value. During this subphase, each gateway node GNi generates a random positive number Si as secret and it sends the message below, encrypted with the appropriate pairwise key K_{ij}, to its 1-hop neighboring gateway node GNj.

$$GNi \rightarrow GNj : \{Refresh, Idi \parallel Si\}\, K_{ij} \tag{16}$$

Upon the reception of the above message, the recipient GNj uses the appropriate pairwise key K_{ij} to decrypt it. After that, it extracts and saves the secret value Si in its list $List_j := List_j + Si$.

At the end of this subphase, each gateway node GNi determines the smalest secret value Min_i among all the values belonging to its list.
$Min_i = min(S_k), with \quad S_k \in List_i$.

b) In this second subphase, each gateway node GNi perfoms the following operations:

1. generates a nonce $Nonce_i$ and sends the following refresh message along with the smallest secret value Min_i to its one hop constrained nodes Nj encrypted with the appropriate old secret keys.

$$GNi \rightarrow Nj : \{Refresh, Idi \parallel Nonce_i \parallel Min_i\}\, K_{ij} \tag{17}$$

2. updates its square matrix by multiplying the main diagonal elements times the smalest positive value Min_i. So $M_j\,[k,k] = Min_i * M_j\,[k,k] \quad with \quad k = 1, .., n$, and n represents the order of the square matrix M_j.
3. executes the step *b*) of the key establishment phase in order to generate new symmetric keys, which can be used for further communication.
4. deletes the old or prior keys from their memories and replaces it with the new ones.

Each constrained node who receives the first above refresh message, it uses the appropriate symmetric key K_{ij} to decrypt it, then it extracts the secret value and executes the steps 2) 3) and 4) above. After that, it replies to the sender by the following acknowledgment message, which contains the successor of the received nonce, encrypted with the new symmetric key. Note that constrained nodes that have more than one gateway node nearby, accept the first refresh message received and disregard the others.

$$Nj \rightarrow GNi : \{Ack, Idj \parallel Nonce_i + 1\}\, K'ij \tag{18}$$

4. Analysis

4.1. Security analysis

In this section, we first give an informal analysis of our scheme by showing its resistance against several types of attacks. After that, we present a formal analysis by using AVISPA (Automated Validation of Internet Security Protocols and Applications) tool [15].

4.1.1. Informal analysis

The proposed approach is resilient to the following types of attacks.

- Eavesdropping attack
  IoT devices generally use wireless links to communicate with each other. An adversary can easily listen to messages when they are clearly transmitted over wireless links.
  In our scheme, during the key generation phase between two neighbors, an adversary can have access to the first part of the *Hello* message because it is not encrypted. However, he is not able to compute the symmetric pairwise key since he knows neither the initial square matrix M nor the one way hash function Hash() that are used. After the key estalishment phase, all communications betwen nodes are secured with the established pairwise keys. As a result, an attacker could not read the content of the exchanged messages. Thus, our system is resilient to this kind of attack.

- Compromising (Node capture) attack
  An adversary who captures and compromises an IoT device can have access to all the information stored in its memory (secret keys, square matrix and the hash function). Since a node stores at most a number of keys equals to the number of its neighbors. Therefore, the attacker can read, alter and/or delete messages transmitted over the communication channels shared between the compromised node and its neighbors. The other channels will not be affected and will remain safe. We assume that this type of attack does not occur before the deletion of the initial square matrix M, otherwise an adversary will be able to compute all the secrets keys. In other words, if the initial matrix M is compromised before the expiration of the timer then all the secret keys will be revealed. But if an adversary captures and compromises a node after the expiration of the timer then only the node's symmetric keys will be discovered because the initial matrix M has been deleted from its memory. On the other hand, an adversary that captures a gateway node dont have access to the information stored in its memory since it is protected by the use of TPM [13, 14]. Thus, the communication links betwen the gateway nodes and the server could not be compromised.

- Sybil attack
  In a Sybil attack, a malicious node provides several identities to other nodes in the network. To protect against such attack, the identity of each node needs to be checked. This can be performed directly by a neighboring node or indirectly by another trusted entity.
  In our system, this verification is performed directly by a neighboring node since a node shares a unique symmetric key with each of its neighbors. So every node checks the identity of its neighbor by using the appropriate symmetric key when they want to establish a link betwen them. Thus, a malicious node cannot claim to be another node without prior knowledge of all the secret keys stored in the memory of the latter. On the other hand, in neighbor discovery and join phases for example, a node inserts its identity then generates a MAC (Message Authentication Code), which is added to the message before sending it. The recipient then uses this MAC to verify the identity and authenticate the sender. Therefore, the proposed system offers a certain robustness against this attack.

- Forward and backward secrecy
  Forward secrecy ensures that a node is unable to decipher new messages by using an old key. As for backward secrecy, it is the opposite. That means, a node is unable to decrypt previous messages encrypted with old keys by using the new key. Both forward and backward secrecy are used to defeat node capture attacks.
  In the proposed scheme, when a node is compromised or leaves the network, all its neighbors remove from their memories the symmetric key that they share with it and update their neighbor vectors and also their square matrices. Therefore, the leaving node cannot use the old symmetric keys to establish a new communication with its neighbors because these latter have already removed this

old shared key from their memories. Furthermore, these neighbors cannot establish any more new communications with this leaving node since they cannot generate a new secret key for this latter because of the deletion of the row and the column corresponding to this node from their square matrices. A node that has left the network is unable to decrypt the new messages by using the old keys, because all its neighborings nodes have removed these keys from their memories. Similarly, when a node joins the network again it can not decipher the previous messages, because the new keys differ from the old ones. Thus, our scheme ensures forward and backward secrecy.

- Replay attack
  In a Replay attack, the attacker first does passive eavesdropping by capturing the data exchanged over the network, and later uses this obsolete data to violate integrity or authentication.
  Freshness is introduced into our scheme by using a random and unpredictibles values such as secret values and nonces in order to defeat this kind of attacks. In addition, an attacker is not able to reuse the messages of an old session in another new session since keys are periodically refreshed. Of course, in the key refresh phase for instance, if the attacker does passive eavesdropping and captures the refresh message sent by a gateway node i to the node j, The attacker is not able to decrypt the message since he does not know the pairwise key K_{ij}. When the attacker replays such rekeying message captured in the previous session, the recipient node j remarks that the new pairwise key K'_{ij} does not match with the old one K_{ij} with which the message is encrypted. In fact, the node j has already received a fresh message comming from the gateway node i or from another one and thus it has already updated its pairwise keys and deleted the old ones from its memory. As a result, the recipient node j detects such replay attack.

- Man in the middle attack
  In a Man-in-the-Middle attack, the attacker is placed between the two communicating entities in order to intercept and/or modify the exchanged messages. The sender and the recipient are unaware of the existance of the middleman.
  In our scheme, all the messages are exchanged directly between any pair of IoT neighbors without any intermediate entity. Both IoT nodes detect the presence of the adversary since they receive the messages sent by this unauthorised user. Furthermore, only $Hello$ and $Join$ messages are not encrypted but they contain MAC part so that any modification of the messages during their transmission will be detected by the recipient. All of the other messages exchanged in our scheme are encrypted with the appropriate keys, so the attacker is not able neither to read nor to alter their content. In others words, the attacker can not launch any attack on confidentiality and integrity. Thus our scheme is robust against Man in the middle attack.

Furthermore, our scheme ensures the following properties that must be taken into consideration to assess key management protocols:

- **extensibility** since it allows to new nodes to be added to the network after the deployment phase by going through the add phase. Thus, our scheme is flexible and allows dynamism.

- **scalability** because each node stores at most only the keys of its neighbors, thus sensors are assigned with a small number of keys to save their memories. As a result, the number of keys stored in a constrained node's memory does not increase linearly or exponnentially with the add of new nodes.

- **resilience** since the corruption of a given node has limited consequences on the whole of the system. Indeed, an attacker who compomises a node will only have access to the information (keys) stored in its memory and as the node stores few keys (at most only these of its direct neighbors) by consequence such an attack will have little influence on the whole of the system. In other words, if a given node is compromised, less number of keys would be revealed to the adversary.

- **authentication** through the use of initial shared secret such as hash function, initial matrix and nonces in the key management.
- **distribution** since the initial context used in the key estabilishment process is distributed in an offline mode before the deployment.

On the other hand, the secret keys are established in a distributed manner, thus avoiding a central entity that could be a point of failure or weakness.

4.1.2. Formal analysis

In this subsection, we give a formal analysis of our scheme by using AVISPA tool [15]. AVISPA (Automated Validation of Internet Security Protocols and Applications) is a modeling tool for building and analyzing formal models of the security protocols. It includes four back-ends: OFMC (On-the-fly Model-Checker)[16], CL-AtSe (Constraint-Logic-based Attack Searcher)[17], SATMC (SAT-based Model-Checker) [18] and TA4SP (Tree Automata based on Automatic Approximations for the Analysis of Security Protocols).
Our protocols are first written in Alice-Bob notation then specified in High Level Protocol Specification Language HLPSL [19]. After that their security properties like secrecy, integrity and authentication are analyzed by the four back-ends of the AVISPA tool.
As an example, the Alice-Bob notation for the $JOIN$ protocol is given below.

1. $S->A : \{Ss, Ns\}\, Ksa$
2. $B->A : B, Nb, Hash(B, Nb, Ss)$
3. $A->B : A, Na, Sa, Succ(Nb), Hash(A, Na, Sa, Succ(Nb), Ss)$
4. $B->A : \{Succ(Na)\}Kab$
5. $A->S : \{Succ(Ns)\}Ksa$

where S is a gateway node, A a node and B a new node.

The obtained results show that all protocols are SAFE under the OFMC and CL-AtSe, INCONCLUSIVE under SATMC and SAFE or INCONCLUSIVE under TA4SP back-ends. That means that the security goals stated in the environment roles (secrecy, integrity and authentication) are satisfied and could not be violated. In other words, the back-ends found no attacks as shown in Table 2.

Phase/Back-end	**OFMC**	**CL-AtSe**	**SATMC**	**TA4SP**
KEP	Version of 2006/02/13 SUMMARY SAFE	 SUMMARY SAFE	 SUMMARY INCONCLUSIVE	 SUMMARY INCONCLUSIVE
JOIN	SUMMARY SAFE	SUMMARY SAFE	SUMMARY INCONCLUSIVE	SUMMARY INCONCLUSIVE
LEAVE Case1	SUMMARY SAFE	SUMMARY SAFE	SUMMARY INCONCLUSIVE	SUMMARY INCONCLUSIVE
LEAVE Case2	SUMMARY SAFE	SUMMARY SAFE	SUMMARY INCONCLUSIVE	SUMMARY SAFE
REFRESH	SUMMARY SAFE	SUMMARY SAFE	SUMMARY INCONCLUSIVE	SUMMARY SAFE

Table 2: Results of protocols specifications verified in AVISPA

Therefore, we can affirm that an attacker is not able to launch an attack against the authentication, the integrity and the secrecy on our protocols. Of course, the mutual authentication betwen a node and its neighbors is satisfied by exchanging their nonces. The secrecy of the sensitive data like shared key is also guaranteed. The integrity is guaranteed by using hash function. In fact, for instance when an intruder intercepts the message M, $Hash(M, k)$ and substitutes the message M by $\acute{M}$, he will not be able to compute H($\acute{M}$,k) since he does not know the secret key K. Whenever he sends $\acute{M}$, H(M,k), the

recipient immediately notices that the message has been modified since the hash value of the received message H(Ḿ,k) is different from the hash value sent $H(M, k)$.

4.2. Analytical analysis

In this subsection, we evaluate the resilience of our scheme against node capture attack. The resilience is computed as the fraction of links compromised Fx when x nodes are captured. In our scheme, when an adversary captures a constrained node, he could read all the keys stored in its memory. Since each node stores, at most, only the symmetric keys of its neighbors, the number of links that will be compromised is equal to twice the number of its neighbors (the degree d of the compromised node) because when an attacker gets the symmetric key K_{ij} for example, he is able to decrypt all messages exchanged between nodes i and j in both directions, i.e from i to j and vice versa. However, if a gateway node is captured, the number of links that will be compromised is zero because the adversary gets no key since the memory of this type of node is protected by a TPM [13, 14]. Figure 7 shows the obtained results in terms of resilience to node capture attack of the compared schemes. We note that the proportion of compromised links increases with increasing the number of compromised nodes. We assume that the adversary compromises one to twenty constrained nodes randomly. The network considered here is composed of 1000 nodes (constrained and gatways nodes). Thus, each node has about 14 neighbors according to [20]

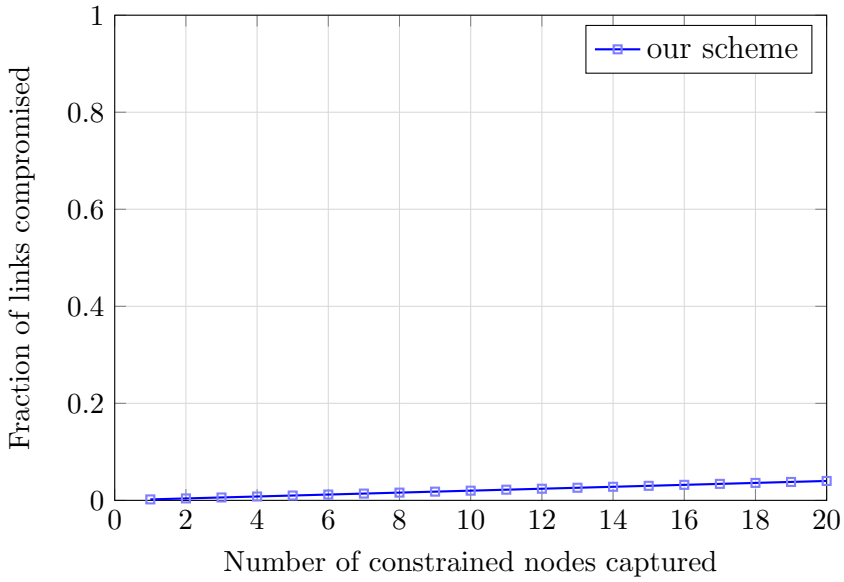

Figure 7: Resilience to node capture attack

4.3. Performance analysis (experimental study)

In this subsection, we present a performance evaluation of our scheme by focusing on the communication, storage and computation overhead during the several phases. We especially estimate the energy consumption of constrained node as Mica2dot sensor node, since the energy of this sensor is very limited. We assume that the rate of gateway nodes is 10% of the network size.

The Table 3 below shows a comparative study of our scheme with the key establishment protocols seen in related work section according to the memory (storage), computation and communication requirements during key establishment phase.

In our scheme, each node needs to store at most only the pairwise keys of its direct neighbors. The gateway nodes also store their public/private keys and the public key of the server node. In addition, a node also stores a square matrix of order d (d is the number of neighbors) and a neighbors vector of d elements. Thus, one node stores $d(d+1)$ elements. If the size of one element is four bytes, then the total space required to store the key material is $4 * d(d+1)$. In the network of 1000 nodes for instance, one

Schemes	Memory	Computation	Communication
Blom [4]	$2(\lambda+1)$	$(\lambda+1)$ multiplications λ additions	$Col_j(G)$ $(\lambda+1)$
Du et al [5]	$1+t\cdot(\lambda+1)$ t key space	$2\lambda+1$ multiplications λ additions	the seed of G and indices of t
Yu et al [6]	$(\lambda+1)(2+w)$ w rows of B	$\lambda+1$ multiplications λ additions	$Col_j(G)$
Rahman et al. [7]	$(\lambda+2)$	$2\lambda+1$ multiplications λ additions	
Y. Zhang et al [8]	key seed si	$2\lambda+1$ multiplications λ additions	request and $row_i(A_c)$
Messai et al [11]	4 + number of cluster members	MAC	$(d+1)$ $Hello$ messages
Our scheme	$d(d+1)$ d neighbors	2 multiplications 1 substraction (addition)+ MAC	2 $Hello$ messages

Table 3: Comparative study of the schemes

node needs to store 840 bytes that is negligible compared with the capacity of the very constrained node like Mica2dot which is about 512 kB.

In the other hand, our scheme needs less computation than the others. In fact, to compute one pairwise key, a node carries out only two multiplications and one substraction (addition), unlike the other schemes which require at least $\lambda+1$ multiplications and λ additions and, thus, it depends on the security parameter λ.

Furthermore, to establish one pairwise key betwen a constrained node and a gateway node, they need to exchange only two $Hello$ messages, whereas in Messai et al scheme, to compute one secret key between a sensor and its cluster head, the sensor node has to send one and receive d $Hello$ messages from its neighbors.

A. **Communication Overhead**

Since the communication cost is closely related on the number and the size of each message exchanged betwen two communicating neighboring nodes, we first evaluate the length of each message sent or received by a constrained node during the differents phases of our scheme, then we give the estimation of energy consumption by sending or receiving such messages according to the energy model given in [21]. The cost of receiving one byte is about $(28.6\mu J)$ and sending a byte is about $(59.2\mu J)$ for the Mica2dot node [21]. We assume that the size of the parameters within messages are 4 bytes for identity of the node, 4 bytes for nonce, E bytes for encryption, H bytes for hash value. The energy consumption on communications of our scheme is given in Table 4.

Phase	Message sent	Message received	Length (byte)	Energy consumption (μJ)
Init	-	-	-	-
KEP	(1)		(4+4+H)=(8+H)	(8+H)*59.2
		(2)	d*(4+4+H)=d*(8+H)	d*(8+H)*28.6
JOIN		(7)	E	E*28.6
		(8)	4+4+H	(8+H)*28.6
	(9)		4+4+4+4+H	(16+H)*59.2
		(10)	E	E*28.6
	(11)		E	E*59.2
LEAVE		(12) or (14)	E	E*28.6
	(13) or (15)		E	E*59.2
REFRESH		(16)	E	E*28.6
	(17)		E	E*59.2

Table 4: Length of messages sent or received by a constrained node

Figure 8 illustrates the communication energy (in joules) consumed by all constrained nodes during key establishment phase in function of hash algorithm. We note that the communication overhead increases for both compared schemes when the size of hash value increases. As the energy consumption is closely related to the amount of data exchanged, the quantity generated during this phase depends on the hash algorithm used and therefore on the size of hash value. We notice that this quantity is more important in EAHKM+ than in our scheme. As a result, the performances of our scheme are higher.

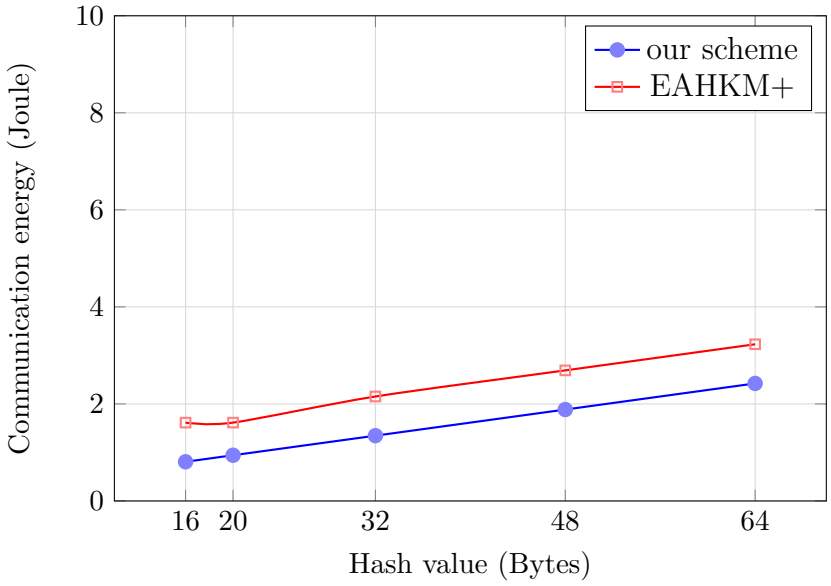

Figure 8: Communication energy consumed by the constrained nodes during key establishment

Figure 9 presents the energy (in joule) dissipated in communication by all constrained nodes during key establishment phase according to the network size. As shown in this Figure, the communication cost of our scheme is better than EAHKM+. Although the number of messages exchanged during this phase in both schemes is the same, but their size in EAHKM+ is greater than ours. As we know, the energy consumption depends directly on the amount of data sent and/or received.

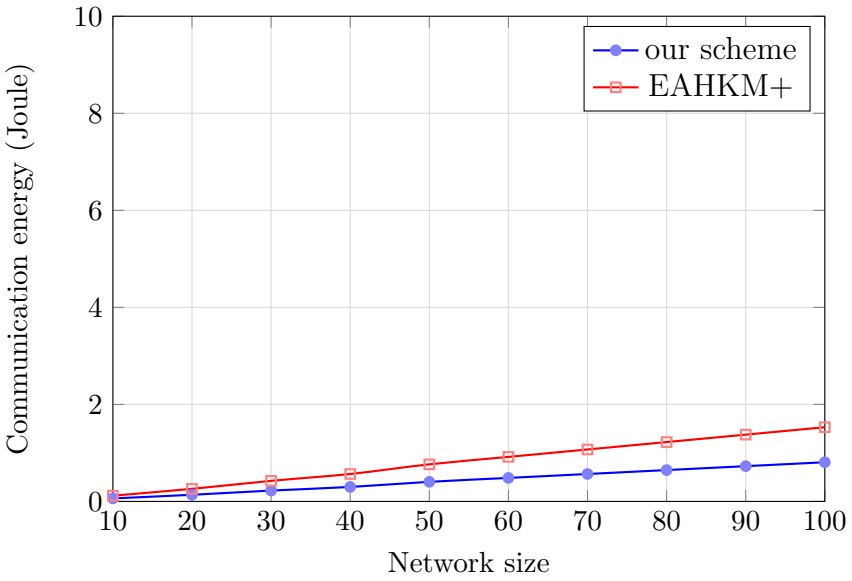

Figure 9: Communication energy consumed by the constrained nodes during key establishment

Figure 10 plots the communication energy (in joules) consumed by all constrained nodes during key refresh phase in function of the network size. We assume that the encryption algorithm used here is AES-128 and the size of hash value is 16 bytes. This Figure indicates clearly that our scheme saves more energy than EAHKM+. This is because in our system, during the rekeying process, each constrained node only needs to receive one encrypted message while in EAHKM+, each sensor receives $(d+1)$, which significantly increases the communication overhead. Moreover, we notice that the consumed energy increases more slowly in our scheme than in EAHKM+. This can be explained by the fact that the quantity of messages received by sensors, in EAHKM+, depends mainly on the number of neighbors which also depends on the network size.

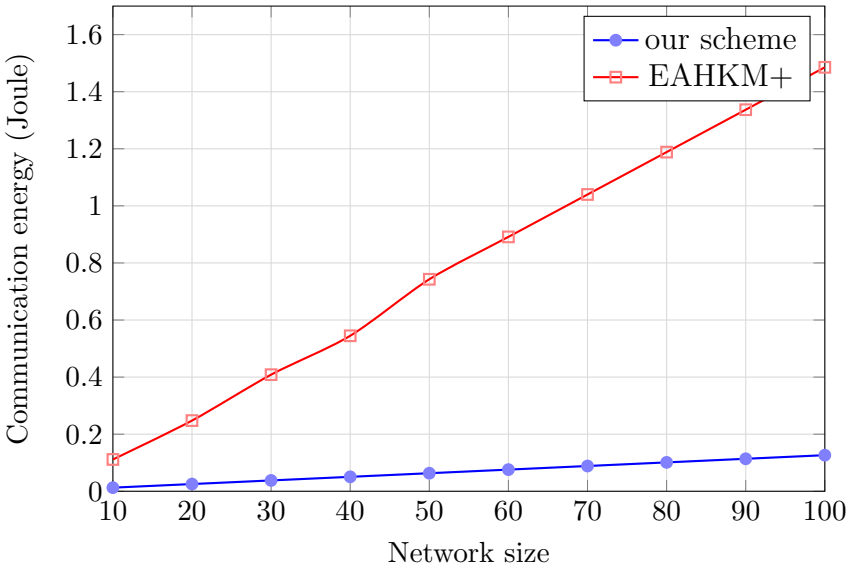

Figure 10: Communication energy consumed by the constrained nodes during key refresh

B. **Computation Overhead**

The computation overhead mainly depends on encryption, decryption and authentication cost. To evalu-

ate this cost, we consider the total number of encryption, decryption and hash operations performed by a constrained node in each phase of our scheme. After that, we estimate the energy consumption. According to authors in [12], the energy cost of one AES-128 encryption (or decryption) and hash operation in TelosB platform is $9\mu J$ and $40mJ$ respectively. The obtained results are summerized in Table 5.

Phase	**Number of AES-128 encryption (decryption)**	**Number of Hash operations**	**Energy consumption (μJ)**
Init	0	0	0
KEP	0	1+2*d	$(1 + 2 * d) * 40 * 10^3$
JOIN	1 + (1+1)	2	$03 * 9 + 2 * 40 * 10^3$
LEAVE	01 + (01)	0	02*9=18
REFRESH	01 + (01)	0	02*9=18

Table 5: Number of encryption, decryption and hash operations performed by a constrained node

Figure 11 plots the energy consumption (in joules) during the computation of the pairwise keys shared betwen constrained nodes with the gateway nodes in our scheme and sensors with their cluster heads in EAHKM+. We notice that our results are slightly better than those of EAHKM+. In fact, in our scheme, constrained nodes only perform hash operations, but in EAHKM+, sensors do encryption and decryption in addition to hash oparations which increases the computation overhead.

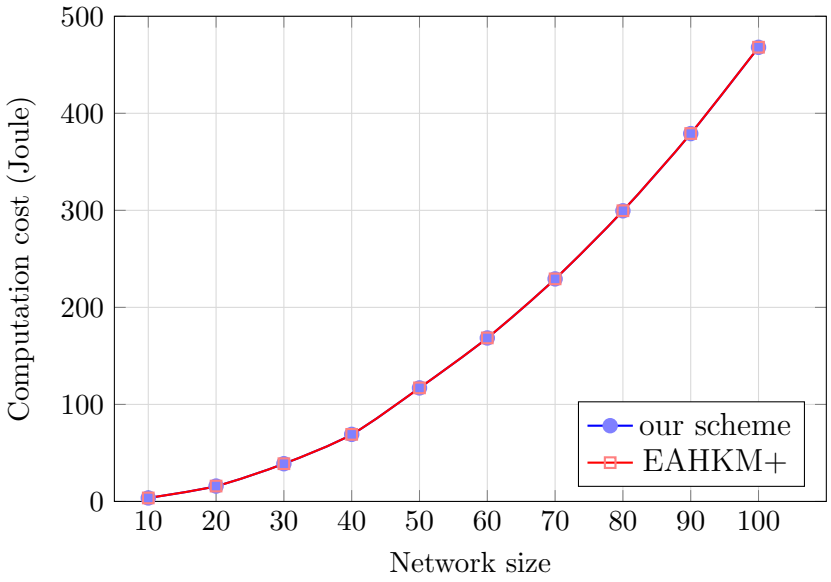

Figure 11: Computation cost in the constrained node side during key establishment

Figure 12 compares the computation cost in terms of energy consumed by all constrained nodes during key refresh phase in function of the network size. The results show that our scheme consumes less energy than EAHKM+. Indeed, in our scheme, constrained nodes just perform encryption and decryption in order to update the old keys, but in EAHKM+, sensors carry out encryption, decryption and hash operations. Furthermore, in our scheme to update keys, constrained nodes only reexecute the second step of key establishment phase, but in EAHKM+, sensors rerun the whole cluster formation and key establishment algorithm.

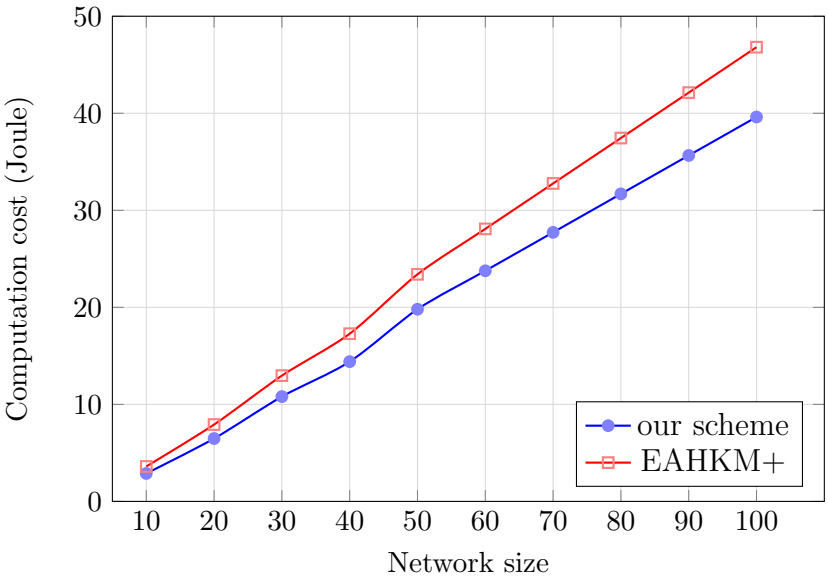

Figure 12: Computation cost in the constrained node side during key refresh

5. Conclusion

In this paper, we have proposed a lightweight matrix based key management scheme for securing communications betwen IoT devices. Security analysis shows that the proposed scheme ensures the security goals like secrecy, integrity and authentication and can protect the sensitive data from various types of attacks. In addition, the proposed system allows extensibility, scalability, resilience, authentication and distribution.
In our future work, we propose to explore the use of blockchain technology for securing the internet of things networks.

References

[1] S. Li, L. Da Xu, S. Zhao, The internet of things: a survey, Information Systems Frontiers 17 (2015) 243–259.

[2] Z. Bi, L. Da Xu, C. Wang, Internet of things for enterprise systems of modern manufacturing, IEEE Transactions on industrial informatics 10 (2014) 1537–1546.

[3] R. Roman, C. Alcaraz, J. Lopez, N. Sklavos, Key management systems for sensor networks in the context of the internet of things, Computers & Electrical Engineering 37 (2011) 147–159.

[4] R. Blom, An optimal class of symmetric key generation systems, in: Workshop on the Theory and Application of of Cryptographic Techniques, Springer, pp. 335–338.

[5] W. Du, J. Deng, Y. S. Han, P. K. Varshney, J. Katz, A. Khalili, A pairwise key predistribution scheme for wireless sensor networks, ACM Transactions on Information and System Security (TISSEC) 8 (2005) 228–258.

[6] Z. Yu, Y. Guan, A key management scheme using deployment knowledge for wireless sensor networks, IEEE Transactions on Parallel and Distributed Systems 19 (2008) 1411–1425.

[7] M. Rahman, S. Sampalli, An efficient pairwise and group key management protocol for wireless sensor network, Wireless Personal Communications 84 (2015) 2035–2053.

[8] Y. Zhang, Y. Xiang, X. Huang, X. Chen, A. Alelaiwi, A matrix-based cross-layer key establishment protocol for smart homes, Information Sciences 429 (2018) 390–405.

[9] L. Xu, Y. Zhang, Matrix-based pairwise key establishment for wireless mesh networks, Future Generation Computer Systems 30 (2014) 140–145.

[10] Q. Wang, H. Su, K. Ren, K. Kim, Fast and scalable secret key generation exploiting channel phase randomness in wireless networks, in: 2011 Proceedings IEEE INFOCOM, IEEE, pp. 1422–1430.

[11] M.-L. Messai, H. Seba, Eahkm+: energy-aware secure clustering scheme in wireless sensor networks, International Journal of High Performance Computing and Networking 11 (2018) 145–155.

[12] G. De Meulenaer, F. Gosset, F.-X. Standaert, O. Pereira, On the energy cost of communication and cryptography in wireless sensor networks, in: Networking and Communications, 2008. WIMOB'08. IEEE International Conference on Wireless and Mobile Computing,, IEEE, pp. 580–585.

[13] W. Arthur, D. Challener, A practical guide to TPM 2.0: using the Trusted Platform Module in the new age of security, Apress, 2015.

[14] S. L. Kinney, Trusted platform module basics: using TPM in embedded systems, Elsevier, 2006.

[15] A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuéllar, P. H. Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, et al., The avispa tool for the automated validation of internet security protocols and applications, in: International conference on computer aided verification, Springer, pp. 281–285.

[16] D. Basin, S. Mödersheim, L. Vigano, An on-the-fly model-checker for security protocol analysis, in: European Symposium on Research in Computer Security, Springer, pp. 253–270.

[17] M. Turuani, The cl-atse protocol analyser, in: International Conference on Rewriting Techniques and Applications, Springer, pp. 277–286.

[18] A. Armando, L. Compagna, Satmc: a sat-based model checker for security protocols, in: European workshop on logics in artificial intelligence, Springer, pp. 730–733.

[19] Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, J. Mantovani, S. Mödersheim, L. Vigneron, A high level protocol specification language for industrial security-sensitive protocols, in: Workshop on Specification and Automated Processing of Security Requirements-SAPS'2004, Austrian Computer Society, pp. 13–p.

[20] H. Chan, A. Perrig, D. Song, Random key predistribution schemes for sensor networks, in: Security and Privacy, 2003. Proceedings. 2003 Symposium on, IEEE, pp. 197–213.

[21] A. S. Wander, N. Gura, H. Eberle, V. Gupta, S. C. Shantz, Energy analysis of public-key cryptography for wireless sensor networks, in: Pervasive Computing and Communications, 2005. PerCom 2005. Third IEEE International Conference on, IEEE, pp. 324–328.