A versatile Key Management protocol for secure Group and Device-to-Device Communication in the Internet of Things

Mohamed Ali Kandi, Hicham Lakhlef, Abdelmadjid Bouabdallah, Yacine Challal

A Versatile Key Management Protocol for Secure Group and Device-to-Device Communication in the Internet of Things

Mohamed Ali Kandi[1], Hicham Lakhlef[1], Abdelmadjid Bouabdallah[1] and Yacine Challal[1,2]
[1]Sorbonne Universités, Université de Technologie de Compiègne, CNRS, UMR7253 Heudiasyc-CS 60319-60203 Compiègne Cedex, France
[2]Laboratoire de Méthodes de Conception de Systèmes, École nationale Supérieure d'Informatique, Algiers, Algeria
[2]Centre de Recherche sur l'Information Scientifique et Technique, Algiers, Algeria
Email: $\{mohamed-ali.kandi, hicham.lakhlef, madjid.bouabdallah, yacine.challal\}$ @$hds.utc.fr$

***Abstract*—The Internet of Things (IoT) is a network made up of a large number of devices that collaborate to provide various service for the benefit of society. Two communication modes are required to enable a smooth collaboration. A device can send the same message to several other ones participating in the same service. It may also address a specific device in a Peer-to-Peer manner. The first mode of communication is called Group Communication, while we refer to the second as Device-to-Device Communication. One of the main challenges facing the IoT is how to secure these two modes of communication. Among all the security issues, the Key Management is one of the most challenging. This is mainly due to the fact that most of the IoT devices have limited resources in terms of storage, calculation, communication and energy. Although different approaches have been proposed to deal with this problem, each of them presents its own limitations and weaknesses. Moreover, they usually consider either the Group or the Device-to-Device Communication. In this paper, we propose a novel versatile Key Management protocol for the Internet of Things. To the best of our knowledge, it is the first protocol that secures both modes of communication at the same time. We then analyze the security and performance of our solution and compare it to the existing schemes. For Group Communication, we show that our solution ensures the forward and backward secrecy and, unlike most of the existing Group Key Management protocols, guarantees the secure coexistence of several services in the network. With regard to Device-to-Device Communication, we prove that our solution is flexible and provides a good level of resilience and network connectivity compared to the existing Peer-to-Peer Key Management schemes. We finally demonstrate that, by balancing the loads between the heterogeneous devices according to their capabilities, our solution is both efficient and scalable.**

***Index Terms*—Internet of things, Group Communication, Device-to-Device Communication, Security, Key Management.**

I. Introduction

The number of devices connected to Internet is constantly increasing since its appearance. Now that this number far exceeds that of people in the world, we are no longer talking about Internet but about Internet of Things (IoT). This emerging technology gives rise to revolutionary applications such as health care, environment monitoring, smart homes, smart cities, autonomous vehicles...etc. To achieve this, the IoT devices are able to automatically communicate to each other in two different ways: Group Communication and Device-to-Device Communication.

In Group Communication, a device communicates with several other ones at the same time. These devices usually participate in the same service and thereby have a common interest. This same device can address a specific other one in a Peer-to-Peer manner. This is called Device-to-Device Communication [36, 37]. An example of Group Communication is the Vehicle-to-Everything Communication. It is a technology that consists of allowing a vehicle to communicate with all the nearby devices (cars, bicycles, public lighting...etc.). The aim is to make the vehicle sense its environment and therefore take the right decision. A Vehicle-to-Vehicle Communication, on the other hand, allows two specific vehicles to exchange information, in a Peer-to-Peer manner, about their speed and position. Thus, they can avoid crashes, ease traffic congestion and improve the environment [11].

One of the main challenges facing the IoT is how to secure communication between its devices. This is considered as a difficult issue mainly because the IoT devices have the particularity of being heterogeneous [28]. They have then different capabilities in terms of storage, calculation, communication and energy. More importantly, some of them are constrained by their small physical size and so have limited capabilities [1]. Among all the security issues encountered by the IoT, the Key Management (KM) is one of the most difficult. The KM is the core of secure communication. Its role is to establish secure links between the network members. To achieve this, it provides them with secret cryptographic keys that are used to encrypt and decrypt the exchanged data [56]. According to the mode of communication, the KM schemes can be classified into two categories: Group and Peer-to-Peer Key Management.

Group Key Management (GKM) protocols entail using the same key to encrypt and decrypt the Group Communication (Figure 1a). This key, usually called group key, must be known only by current members [15]. The GKM must then ensure that, when a member leaves the group, it is no longer able to decipher the future messages (forward secrecy). It must also guarantee that a joining node cannot decipher the previous ones (backward secrecy). Backward and forward secrecy are usually guaranteed by rekeying. Thus, when a node joins or leaves the group, the keys are revoked and new ones are distributed to the remaining members.

Peer-to-Peer Key Management (PKM) schemes can be used to secure the Device-to-Device Communication. It consists of using several keys instead of one. The less communications a compromised device can decipher, the more the PKM is considered as resilient against node capture [53]. A perfect level of resilience can be reached (at the expense of scalability) if a distinct key is used for each pair of devices. This key, usually called Pairwise Key, should be known by the two devices only (Figure 1b). Hence, when a network member is compromised, the communications of others are not jeopardized.

As the IoT requires the two modes of communication, none of these schemes is suitable for it. Indeed, if the same key is used for all Device-to-Device Communication, every network member will be able to decipher them. The GKM schemes are therefore not resilient against node capture. On the other hand, if several keys are used in Group Communication, the same message will be encrypted and sent several times. This will require additional calculation and communication and thereby more energy consumption. Finally, most of the existing schemes suffer from considerable KM overheads and are not suitable for the IoT constrained devices. Implementing two different protocols will then be too heavy for them to handle.

In this work, our aim is to secure the two communication modes of the IoT without loss of efficiency and scalability. To achieve this, we balance the protocol overheads between the heterogeneous devices according to their capabilities. The contributions of our work are as follows:

- We present a state of the art of the different existing KM schemes. We classify them according to the mode of communication into two categories: GKM and PKM.
- We propose a versatile KM protocol for the IoT. To the best of our knowledge, it is the first protocol that secures both modes of IoT communication at the same time.
- We analyze the security and performance of our solution and compare it to the existing ones. For Group Communication, we show that it ensures the forward and backward secrecy and, unlike most of the existing GKM schemes, guarantees the secure coexistence of several services in the network. For Device-to-Device Communication, we prove that our solution is resilient, flexible and has a good connectivity compared to the existing PKM schemes. We finally show that, by balancing the loads between the heterogeneous devices according to their capabilities, our solution is efficient and scalable.

The remainder of this paper is organized as follows: related works are discussed in Section II. We detail then our solution in Section III. Section IV presents the security analysis of our protocol. In Section V, we evaluate the performance of our solution. Finally, we conclude our work in Section VI.

II. Related Works

The Key Management is the core of secure communication. Although different approaches have been proposed, each of them presents its own limitations and weaknesses. More importantly, they are usually intended for one of the modes of the IoT communication. Different classifications and taxonomies of these methods have been proposed in the literature. In [13] and [39], the authors classify the KM schemes according to the network topology: centralized, decentralized and distributed. In the first class, a single entity controls the whole network, while in the second one, the network management is distributed on several entities. In the third category, however, the network members handle the Key Management themselves.

On the other hand, the authors of [56] classify the KM schemes according to the encryption technique used: symmetric, asymmetric and hybrid. Symmetric approaches [5, 7–10, 17–23, 25–27, 30–32, 35, 45–48, 51–55] involve the use of the same key for encryption and decryption. On the contrary, asymmetric protocols [2, 3, 12, 33, 34, 38, 40, 42, 43, 49, 50] use two different keys: a public key which may be disseminated widely and a private key which is known only to the owner. Finally, a hybrid approach [4] consists of combining a symmetric scheme with an asymmetric one. Generally, symmetric schemes require less computation time than the asymmetric ones and are then more suitable for the IoT constrained devices. For this reason, we focus, in this work, on the symmetric approaches.

According to the mode of communication, we classify the existing symmetric Key Management schemes into two categories: the GKM (Figure 1a), for Group Communication, and the PKM (Figure 1b), for Device-to-Device Communication. In the context of the IoT, the GKM protocols must, above all, ensure the backward and forward secrecy. They should also consider the heterogeneous nature of the IoT and the possibility that several services can coexist in the same network. The PKM schemes, on the other hand, must be resilient, flexible and provide a good network connectivity. Finally, both categories must be efficient and scalable.

(a) Group Key Management.

(b) Peer-to-Peer Key Management.

Fig. 1: Key Management.

A. Group Key Management schemes

The Group Key Management schemes' requirements for the IoT that we consider in this paper are: backward and forward secrecy, collusion resistance, heterogeneity, efficiency and scalability. The GKM must, above all, ensure that the group key is known only to the current members. Thus, when a node joins or leaves the network, the key is revoked and new one is distributed to the remaining members. This rekeying ensures that a joining node will not have access to the old keys (backward secrecy) and a leaving member will no longer know the future ones (forward secrecy). The GKM must also prevent multiple evicted nodes to cooperate to regain access to the group key. Such an attack is referred to as collusion attack [45]. Heterogeneity includes the fact that the IoT devices have different capabilities in terms of storage, calculation, communication and energy but also the possibility that they participate in different services. Efficiency implies minimal use of node resources. Finally, if increasing the network size does not significantly degrade its performance, the protocol is scalable. The GKM schemes are usually based on: tree structures, combinatorial optimisation, batch rekeying,...etc.

1) Tree based schemes:

The Logical Key Hierarchy (LKH) [48, 51] consists of using a tree structure to reduce the communication cost during the process of rekeying. The root of the tree corresponds to the group key, its leaves to the members' secret keys and the other nodes to intermediate keys. Each member stores the keys forming its branch. When a device joins or leaves the group, the server replaces only the keys it knows. In the case of a binary tree, nodes' storage cost will be proportional to $O(log_2(n))$ and the size of the rekey message to $O(2log_2(n))$. The OFT protocol [35] uses a One-way Function to reduce the size of the rekey message to $O(log_2(n))$. Both CASMA and GROUPIT protocols aim to deal with the dynamicity of IoT environments. While the former divides the network into multiple zones each implementing LKH [24], the latter combines LKH with the Chinese Remainder Theorem [29].

The Tree based schemes are usually secure as they guarantee the backward and forward secrecy and are resistant to collusion attacks. They are also reasonably efficient and provides a good scalability. Nonetheless, these schemes rarely consider the heterogeneous nature of the IoT (Figure 2a).

2) Combinatorial optimisation based schemes:

The Exclusion Basic System (EBS) scheme is based on combinatorial optimization. It aims to make it possible to choose a compromise between the number of keys stored on nodes and that of messages exchanged during the rekeying process. The idea was first introduced in [20]. Other protocols were then proposed to improve the efficiency and the collusion resistance. The protocols GKIP [22] and SHELL [52], for example, are based on the nodes deployment knowledge to achieve this, while LOCK [21] uses two layers of EBS.

The EBS based schemes ensure the backward and forward secrecy. They are efficient and scalable. Nevertheless, they are generally vulnerable to collusion attacks and do not consider the heterogeneous nature of the IoT (Figure 2b).

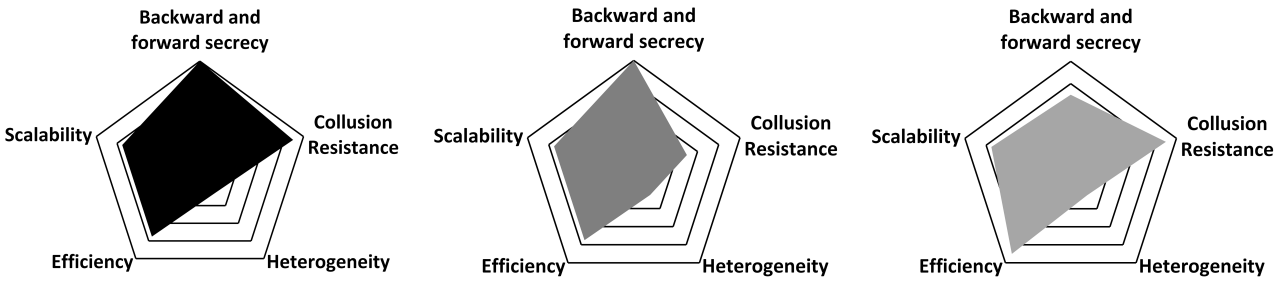
3) Batch rekeying based schemes:

Most of the exiting GKM schemes are based on individual rekeying, i.e. they rekey the group after each join or leave request. For more efficiency, the batch rekeying based schemes [30, 31, 47] were proposed. The main idea is to periodically rekey the group in order to reduce the rekeying overheads.

It is obvious that these schemes are more efficient than those based on individual rekeying. However, a new node has to wait until the next rekeying operation to actually joining the network. Even worse, as long as the group key has not been replaced yet, a leaving or an evicted member can still decipher the communications. Forward secrecy is then not totally guaranteed (Figure 2c).

4) Discussion:

Our literature review shows that the main problem of the existing solutions is that they do not consider the heterogeneous nature of the IoT (Figure 2). They do not balance the loads between devices and impose the same costs on a powerful computer or a weak sensor. Thus, while a negligible part of the former's resources is used, those of the latter may not even be enough. Also, they usually use the same parameters to secure all communications. As the IoT provides various services, communications within a service will be accessible to all nodes even if they did not subscribe to it. Moreover, the capture of a member will jeopardize all services. In previous works [26, 27], we proposed algorithms that provide various mechanisms to resolve some of these problems. However, these works do not consider Device-to-Device Communication.



(a) Tree based schemes. (b) Combinatorial optimisation schemes. (c) Batch rekeying based schemes.

Fig. 2: Group Key Management approaches.

B. Peer-to-Peer Key Management schemes

The PKM schemes requirements for the IoT are: resilience, flexibility, connectivity, efficiency and scalability. A protocol is resilient if the capture of a node does not jeopardize the communications of the other ones. On the other hand, a flexible protocol operates well regardless of the position of nodes and supports their dynamic deployment, i.e. nodes can join and leave the network at any time [13]. When network connectivity is low, some neighboring communicators may not share a common key and then relay on intermediate nodes to establish a secure path. Finally, efficiency and sclablilty were presented in the previous section. On this basis, the Peer-to-Peer Key Management schemes can be classified into three subcategories: deterministic, pure probabilistic and deployment knowledge based schemes.

1) Deterministic schemes:

The basic deterministic scheme, also called Pairwise Key protocol, consists of using a distinct pairwise key for each pair of nodes. This technique [17], nevertheless, requires a lot of storage as nodes has to store a number of keys proportional to the network size. Other approaches were then proposed. Polynomial-based protocols [5, 9] use bivariate polynomials, $f(x,y) = \sum_{(i,j)}^{t} a_{ij} x^i y^j$, for which $f(x,y) = f(y,x)$. In each node i is then stored $f(i,y)$. Thus, a pair of nodes (i,j) can calculate the shared key $f(i,j)$. Matrix-based schemes [8, 19, 46], on the other hand, use symmetrical matrices. The main idea is that each node i stores the i^{th} row vector and the i^{th} column vector. Two nodes wishing to communicate exchange their columns and multiply them by their own row to get the shared key. Note that the rows must be kept secret.

Deterministic schemes have the advantage of being resilient since the capture of a node will not jeopardize the other ones. Moreover, they are usually efficient and have a good connectivity. However, they suffer from poor scalability (Figure 3a). Indeed, Pairwise Key schemes require that a node stores as many keys as there are members in the network. On the other hand, the larger is the network, the more vulnerable the Polynomial and Matrix-based approaches are to compromise because captured nodes can collaborate to recover the polynomial or the Matrix. Furthermore, most of these schemes lack flexibility as they are based on key pre-distribution, i.e. the keys are stored in the nodes' memory before their deployment.

2) Pure probabilistic schemes:

The first pure probabilistic scheme were introduced in [23]. It consists of using a large pool of keys and to distribute some of them (a key ring) to each network member. Thus, two nodes can only communicate if they have a common key. Otherwise, they relay on intermediate nodes to establish a secure path. Other approaches were then proposed to enhance resilience. Using the Q-composite [10] scheme, nodes can only communicate if they share Q keys. Polynomial pool based schemes [41, 55] use a pool of polynomials instead of keys.

These schemes are as resilient as the deterministic ones and even more scalable. Nevertheless, they suffer from poor flexibility, efficiency and connectivity. Indeed, they are usually based on key pre-distribution. Moreover, intermediate nodes may be necessary to establish secure paths between two neighbouring nodes (Figure 3b). This requires additional calculation and communication and thereby more energy consumption. Some works tried to enhance the connectivity using the unital design theory [7], system of equations [54]...etc. However, as long as they are probabilistic, the connectivity is never total.

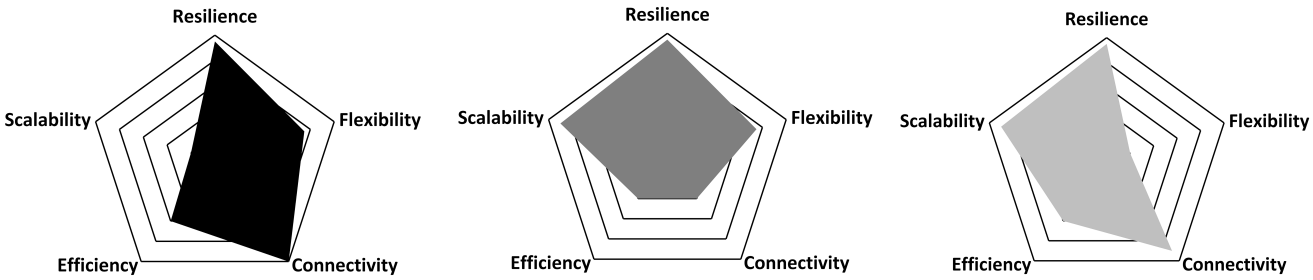
3) Deployment knowledge based schemes:

These schemes are neither deterministic nor purely probabilistic. They are based on deployment knowledge to maximize the network connectivity. Thus, to increase the probability of sharing common keys, nodes are distributed in regional zones. Key pools are then assigned to them so that neighboring nodes share a maximum of keys. Like the other two approaches, the deployment knowledge based schemes can use pairwise keys [16, 18], polynomials [32] or matrices [25, 53].

These approaches are as resilient as the deterministic schemes. They also have a better network connectivity than the pure probabilistic ones and are then more efficient. However, these methods are not flexible and lose all their interest if the deployment knowledge is not possible (Figure 3c).

4) Discussion:

Our literature review shows that none of the existing solutions meets all the IoT requirements: resilience, efficiency, scalability, connectivity and flexibility (Figure 3). Most of deterministic protocols suffer from poor scalability and flexibility. Although the solution we propose is deterministic, we show that, being scalable and flexible, it provides the best compromise between the IoT requirements.



(a) Deterministic schemes. (b) Pure probabilistic schemes. (c) Deployment knowledge schemes.

Fig. 3: Peer-to-Peer Key Management approaches.

III. Our solution

Our solutions combines the benefits of both Group Key and Peer-to-Peer Key Management schemes without loss of efficiency or network scalability. Thus, it allows to multiply the functionalities of the IoT devices and secures the two modes of communication that they may use: Group and Device-to-Device Communication.

For Group Communication, we prove that our solution guarantees the backward and forward secrecy and resists to collusion attacks. Furthermore, we show that, unlike most of the existing GKM schemes, our solution considers the heterogeneous nature of the IoT. It provides then the best compromise between the GKM requirements for the IoT (Figure 4). To achieve this, the network members are distributed into several groups according to the services to which they subscribe. These groups have independent security parameters so that the compromise of a service has no effect on the others. Each group is in turn divided into logical subgroups having different sizes. The aim is to balance the load between its heterogeneous members according to their capabilities.

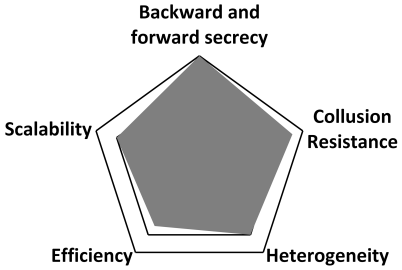


Fig. 4: Group Key Management.

For Device-to-Device Communication, we prove that our solution provides a level of resilience, against node capture, comparable to that offered by the deterministic schemes. Furthermore, we show that, unlike most of the existing probabilistic approaches, our solution is flexible and has a good network connectivity. Thus, it provides the best compromise between the PKM requirements for the IoT (Figure 5). To achieve this, we take advantage of the grouping and subgrouping previously presented. A device shares then a distinct pairwise key with each member of its subgroup, a unique pairwise subgroup key with the members of each of the subgroups of its group and a single pairwise service key with the other members of each service in which it participates. This will significantly reduce storage costs and make our solution more scalable.

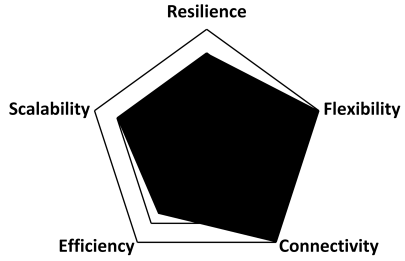


Fig. 5: Peer-to-Peer Key Management.

A. Overview

Our solution is organized into three layers (Figure 6). The upper layer divides the network N into multiple groups. Nodes are then assigned to them according to the services to which they subscribe. Current members can be moved from a group to an other when they subscribe to new services or unsubscribe from others. By doing this, the security parameters of services will be independent and the compromise of one of them will have no effect on the others. Each group, G_i, is associated with an ID, gid_i, which is unique within N.

On the other hand, the middle layer distributes the nodes of each group into subgroups. Note that these subgrouping is logical and transparent to the application layer. The aim is to enhance the protocol efficiency and scalability. Each subgroup requires an overhead proportional to the capability of its members. When a node joins a group, the protocol assigns it to the subgroup that matches its capability. Subgroups are created and can be splitted when it is necessary. Also, when a node leaves a group, the protocol tries to reduce the number of subgroups. It removes then those that become empty and merges the others to the possible extent. Each subgroup, S_j^i, of the group G_i is also associated with a unique ID, sid_j^i.

Finally, the lower layer manages the nodes and the cryptographic keys they hold. Just like groups and subgroups, each node $u_k^{i,j}$ of the subgroup S_j^i is associated with a unique ID, $nid_k^{i,j}$. The keys can be classified into two types: Data Encryption Keys ($DEKs$), which are used to encrypt the data exchanged between nodes, and Key Encryption Keys ($KEKs$), which are used to protect the $DEKs$.

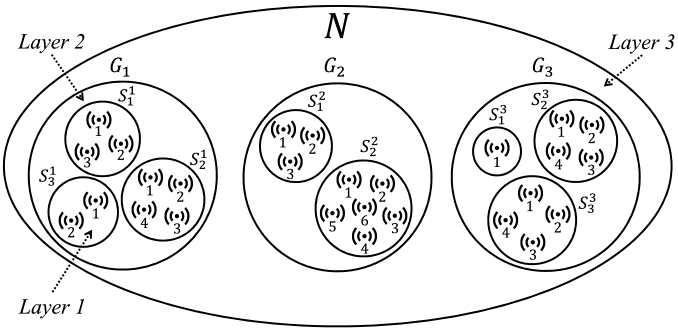


Fig. 6: Example of a network partitioning.

B. Layer 1: Node and key management

The first layer manages the nodes and the keys they store. Each node is first assigned to a subgroup S_j^i of the group G_i. The group is chosen according to the combination of services in which the joining node participates (Section III-D). Also, the choice of the subgroup is made according to the node's capability (Section III-C). The node $u_k^{i,j}$ is then associated with an ID, $nid_k^{i,j}$, which is unique within S_j^i and reflects its members' total order. Given two nodes $u_k^{i,j}$ and $u_{k_2}^{i,j}$, we have $nid_k^{i,j} < nid_{k_2}^{i,j}$ if and only if $u_k^{i,j}$ has joined S_j^i before $u_{k_2}^{i,j}$. Thus, $u_k^{i,j}$ is considered as an elder cognate of $u_{k_2}^{i,j}$ whereas $u_{k_2}^{i,j}$ is seen as a junior cognate of $u_k^{i,j}$.

Notation	Definition
G_i/G	The i^{th} group of the network / The group of the network if there is only one
S_j^i/S_j	The j^{th} subgroup of the group G_i / The j^{th} subgroup of its group
$u_k^{i,j}/u_k$	The k^{th} node of the subgroup S_j^i / The k^{th} node of its subgroup
A_i	The combination of services associated to the group G_i
n	The number of nodes in the network (or the group if there is only one)
p_i/p	The number of subgroups of the group G_i / The number of subgroups of the group G
m_j	The number of nodes in the subgroup S_j
mc_j	The minimum capacity of the subgroup S_j
c_k	The capability of the node u_k
$K_k^{i,j}/K_k$	The node key of the node $u_k^{i,j}/u_k$
$K_{k,k_2}^{i,j}/K_{k,k_2}$	The pairwise node key shared by the nodes $u_k^{i,j}/u_k$ and $u_{k_2}^{i,j}/u_{k_2}$
K_j^i/K_j	The subgroup key of the subgroup S_j^i/S_j
$K_{j,j_2}^i/K_{j,j_2}$	The pairwise subgroup key shared by the subgroups S_j^i/S_j and $S_{j_2}^i/S_{j_2}$
K^e	The service key of the e^{th} service
K_{i,i_2}	The pairwise service keys shared between the groups G_i and G_{i_2}
$\overleftarrow{K}/\overrightarrow{K}$	The left/right part of the key K
K_R	A refresh key
KDF	A key derivation function
H	A hash function
lsg	The list of subgroups
pcm	The percentage of merging

TABLE I: Summary of notations.

1) Classification of Cryptographic Keys:

According to their usage, the keys managed by our solution can be classified into two types: Data Encryption Keys ($DEKs$) and Key Encryption Keys ($KEKs$). The $DEKs$ are used by nodes to encrypt the data exchanged between them. These keys may be either group keys (used to secure Group Communication) or pairwise keys (used to secure Device-to-Device Communication). The KEK, on the other hand, are used to secure the communication between the KM and the nodes in order to protect the $DEKs$. These keys allow then to ensure the backward and forward secrecy (for Group Communication) and the dynamic deployment of nodes (for Device-to-Device Communication). Thus, nodes can securely join and leave the network at any time. Let us consider a node $u_k^{i,j}$ which belongs to the subgroup S_j^i of the group G_i. The node holds the following keys:

- A node key, $K_k^{i,j}$, which is a KEK known only to $u_k^{i,j}$. It allows the KM to communicate with the node safely.
- A pairwise node key, $K_{k,k_2}^{i,j}$, for each of its cognate $u_{k_2}^{i,j}$. This is a DEK used to secure the Device-to-Device Communication between the node and its cognate $u_{k_2}^{i,j}$.
- A subgroup key, K_j^i, which is a KEK known by the members of S_j^i only. It is used to secure the communication between the KM and the subgroup members.
- A pairwise subgroup key, K_{j,j_2}^i, for each subgroup $S_{j_2}^i$. It is a DEK used to secure the Device-to-Device Communication between the node and the members of $S_{j_2}^i$.
- A service key, K^e, for each service e in which the node participates. This is a DEK used to secure the Group Communication between the service members.
- A pairwise service key, K_{i,i_2}, for each group G_{i_2} containing members that participate in the same service. It is a DEK used to secure the Device-to-Device Communication between the node and the members of G_{i_2}.

Service and subgroup keys are used, instead of the node keys, in order to enhance the protocol scalability and efficiency. Table II shows the keys held by the members of the network, presented in Figure 6, in which two services coexist. The groups G_1 and G_2 contain the nodes participating in the first and second service, respectively, while those of G_3 participate in both services. Note that the keys $K_{k,k_2}^{i,j}$ and $K_{k_2,k}^{i,j}$ are the same and can be used interchangeably. The same goes for the keys K_{j,j_2}^i and $K_{j_2,j}^i$ as well as K_{i,i_2} and $K_{i_2,i}$.

2) Hash Functions:

To reduce the storage cost, nodes can share some pairwise keys. They are then able to decipher some Device-to-Device messages that are not intended for them. To enhance the resilience of our solution without loss of scalabilty, we propose the use of a Hash Function (H) or two (One-level and Two-level approach, respectively). The aim is to create different keys from a single pairwise key. Thus, a node will store a single pairwise key without sharing it with the others.

One-level approach: In this approach, instead of storing the pairwise subgroup key, K_{j,j_2}^i, and the pairwise service key, K_{i,i_2}, the node $u_k^{i,j}$ stores the $(k-1)^{th}$ hash of the former, $K_k^{i,(j,j_2)}$, and the $(j-1)^{th}$ hash of the latter, $K_j^{(i,i_2)}$ (Formulas 1 and 2, respectively).

$$K_k^{i,(j,j_2)} = H^{(k-1)}(K_{j,j_2}^i) \quad (1)$$

$$K_j^{(i,i_2)} = H^{(j-1)}(K_{i,i_2}) \quad (2)$$

Although nodes will have different pairwise keys, one of them can calculate the key known by the other. This is due to the fact that the keys they hold are calculated from the same key and using the same Hash Function. To have a common communication key, the nodes relay on their IDs. Thus, knowing the IDs of each other, the node with the smallest one can use H to calculate the key of the other.

Node	Node key	Pairwise node keys	Subgroup key	Pairwise subgroup keys	Service keys	Pairwise service keys
$u_1^{1,1}$	$K_1^{1,1}$	$K_{1,2}^{1,1}, K_{1,3}^{1,1}$	K_1^1	$K_{1,2}^1, K_{1,3}^1$	K^1	$K_{1,3}$
$u_2^{1,1}$	$K_2^{1,1}$	$K_{2,1}^{1,1}, K_{2,3}^{1,1}$				
$u_3^{1,1}$	$K_3^{1,1}$	$K_{3,1}^{1,1}, K_{3,2}^{1,1}$				
$u_1^{1,2}$	$K_1^{1,2}$	$K_{1,2}^{1,2}, K_{1,3}^{1,2}, K_{1,4}^{1,2}$	K_2^1	$K_{2,1}^1, K_{2,3}^1$		
$u_2^{1,2}$	$K_2^{1,2}$	$K_{2,1}^{1,2}, K_{2,3}^{1,2}, K_{2,4}^{1,2}$				
$u_3^{1,2}$	$K_3^{1,2}$	$K_{3,1}^{1,2}, K_{3,2}^{1,2}, K_{3,4}^{1,2}$				
$u_4^{1,2}$	$k_4^{1,2}$	$K_{4,1}^{1,2}, K_{4,2}^{1,2}, K_{4,3}^{1,2}$				
$u_1^{1,3}$	$K_1^{1,3}$	$K_{1,2}^{1,3}$	K_3^1	$K_{3,1}^1, K_{3,2}^1$		
$u_2^{1,3}$	$K_2^{1,3}$	$K_{2,1}^{1,3}$				
$u_1^{2,1}$	$K_1^{2,1}$	$K_{1,2}^{2,1}, K_{1,3}^{2,1}$	K_1^2	$K_{1,2}^2$	K^2	$K_{2,3}$
$u_2^{2,1}$	$K_2^{2,1}$	$K_{2,1}^{2,1}, K_{2,3}^{2,1}$				
$u_3^{2,1}$	$K_3^{2,1}$	$K_{3,1}^{2,1}, K_{3,2}^{2,1}$				
$u_1^{2,2}$	$K_1^{2,2}$	$K_{1,2}^{2,2}, K_{1,3}^{2,2}, K_{1,4}^{2,2}, K_{1,5}^{2,2}, K_{1,6}^{2,2}$	K_2^2	$K_{2,1}^2$		
$u_2^{2,2}$	$K_2^{2,2}$	$K_{2,1}^{2,2}, K_{2,3}^{2,2}, K_{2,4}^{2,2}, K_{2,5}^{2,2}, K_{2,6}^{2,2}$				
$u_3^{2,2}$	$K_3^{2,2}$	$K_{3,1}^{2,2}, K_{3,2}^{2,2}, K_{3,4}^{2,2}, K_{3,5}^{2,2}, K_{3,6}^{2,2}$				
$u_4^{2,2}$	$K_4^{2,2}$	$K_{4,1}^{2,2}, K_{4,2}^{2,2}, K_{4,3}^{2,2}, K_{4,5}^{2,2}, K_{4,6}^{2,2}$				
$u_5^{2,2}$	$K_5^{2,2}$	$K_{5,1}^{2,2}, K_{5,2}^{2,2}, K_{5,3}^{2,2}, K_{5,4}^{2,2}, K_{5,6}^{2,2}$				
$u_6^{2,2}$	$k_6^{2,2}$	$K_{6,1}^{2,2}, K_{6,2}^{2,2}, K_{6,3}^{2,2}, K_{6,4}^{2,2}, K_{6,5}^{2,2}$				
$u_1^{3,1}$	$K_1^{3,1}$	-	K_1^3	$K_{1,2}^3, K_{1,3}^3$	K^1, K^2	$K_{3,1}, K_{3,2}$
$u_1^{3,2}$	$K_1^{3,2}$	$K_{1,2}^{3,2}, K_{1,3}^{3,2}, K_{1,4}^{3,2}$	K_2^3	$K_{2,1}^3, K_{2,3}^3$		
$u_2^{3,2}$	$K_2^{3,2}$	$K_{2,1}^{3,2}, K_{2,3}^{3,2}, K_{2,4}^{3,2}$				
$u_3^{3,2}$	$K_3^{3,2}$	$K_{3,1}^{3,2}, K_{3,2}^{3,2}, K_{3,4}^{3,2}$				
$u_4^{3,2}$	$K_4^{3,2}$	$K_{4,1}^{3,2}, K_{4,2}^{3,2}, K_{4,3}^{3,2}$				
$u_1^{3,3}$	$K_1^{3,3}$	$K_{1,2}^{3,3}, K_{1,3}^{3,3}, K_{1,4}^{3,3}$	K_3^3	$K_{3,1}^3, K_{3,2}^3$		
$u_2^{3,3}$	$K_2^{3,3}$	$K_{2,1}^{3,3}, K_{2,3}^{3,3}, K_{2,4}^{3,3}$				
$u_3^{3,3}$	$K_3^{3,3}$	$K_{3,1}^{3,3}, K_{3,2}^{3,3}, K_{3,4}^{3,3}$				
$u_4^{3,3}$	$k_4^{3,3}$	$K_{4,1}^{3,3}, K_{4,2}^{3,3}, K_{4,3}^{3,3}$				

TABLE II: Example of keys held by a network members.

For pairwise subgroup keys, let us consider that the nodes $u_k^{i,j}$ and $u_{k_2}^{i,j_2}$ wish to communicate. The first one stores $K_k^{i,(j,j_2)} = H^{(k-1)}(K_{j,j_2}^i)$ and the second knows $K_{k_2}^{i,(j,j_2)} = H^{(k_2-1)}(K_{j,j_2}^i)$. If we assume that $k < k_2$, $u_k^{i,j}$ can calculate the $(k_2 - k)^{th}$ hash of its key and both nodes will have the same key (Figure 7). Indeed, we have:

$$\begin{aligned} K_{k_2}^{i,(j,j_2)} &= H^{(k_2-1)}(K_{j,j_2}^i) \\ &= H^{((k_2-1)-(k-1))}(H^{(k-1)}(K_{j,j_2}^i)) \\ &= H^{(k_2-k)}(K_k^{i,(j,j_2)}) \end{aligned} \tag{3}$$

With regard to the pairwise service keys, let us assume that the node $u_k^{i,j}$ wishes to communicate with $u_{k_2}^{i_2,j_2}$. In this case, the two nodes do not belong to the same group. However, we assume that they participate in the same service and therefore can communicate. The first node stores $K_j^{(i,i_2)} = H^{(j-1)}(K_{i,i_2})$ and the second $K_{j_2}^{(i,i_2)} = H^{(j_2-1)}(K_{i,i_2})$. If $j < j_2$, $u_k^{i_2,j}$ can calculate the $(j_2 - j)^{th}$ hash of its key and both nodes will have the same pairwise service key. Indeed, we have:

$$\begin{aligned} K_{j_2}^{(i,i_2)} &= H^{(j_2-1)}(K_{i,i_2}) \\ &= H^{((j_2-1)-(j-1))}(H^{(j-1)}(K_{i,i_2})) \\ &= H^{(j_2-j)}(K_j^{(i,i_2)}) \end{aligned} \tag{4}$$

Since Hash Functions are irreversible, the One-level approach ensures that nodes cannot decipher the Device-to-Device messages exchanged between the other nodes that have smaller IDs. Although the resilience is improved, nodes are still able to decrypt the communications of those having bigger IDs. This has led us to propose the Two-level approach.

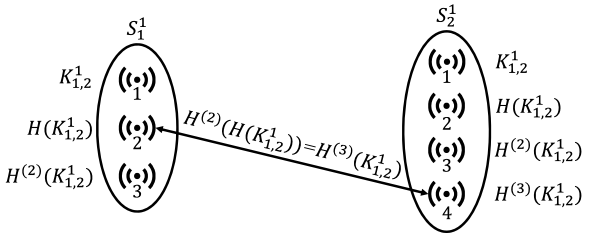


Fig. 7: Example of using One-level approach.

Two-level approach: In this second approach, two different Hash Functions (H and H_2) are used. The pairwise subgroup keys are therefore split into two parts ($\overleftarrow{K}$ and $\overrightarrow{K}$), each hashed separately with one of the two functions. The node $u_k^{i,j}$ stores then the $(k-1)^{th}$ hash (calculated using H) of the left part of the key and the $(m-k)^{th}$ hash (calculated using H_2) of the right part (Formula 5). Note that $||$ is a concatenation operation and m is the maximum number of the subgroup members.

$$K_k^{i,(j,j_2)} = H^{(k-1)}(\overleftarrow{K}^i_{j,j_2})||H_2^{(m-k)}(\overrightarrow{K}^i_{j,j_2}) \tag{5}$$

Like the One-level approach, the Hash Functions and the IDs can be used by the nodes to calculate a comman key. Knowing the IDs of each other, the node with the smallest one can apply H on the left part of its key and the other may apply H_2 on the right part of its key. They will then have the same communication key. Let us consider that $u_k^{i,j}$ and $u_{k_2}^{i,j_2}$ wish to communicate. If we assume that $k < k_2$, $u_k^{i,j}$ can use H to calculate the $(k_2-k)^{th}$ hash of the left part of the key it knows and $u_{k_2}^{i,j_2}$ may use H_2 to calculate the $(k_2-k)^{th}$ hash of the right part of its key (Figure 8). Indeed, we have:

$$\begin{aligned}\overleftarrow{K}^{i,(j,j_2)}_{k_2} &= H^{(k_2-1)}(\overleftarrow{K}^i_{j,j_2}) \\ &= H^{((k_2-1)-(k-1))}(H^{(k-1)}(\overleftarrow{K}^i_{j,j_2})) \\ &= H^{(k_2-k)}(\overleftarrow{K}^{i,(j,j_2)}_{k})\end{aligned} \tag{6}$$

and:

$$\begin{aligned}\overrightarrow{K}^{i,(j,j_2)}_{k} &= H_2^{(m-k)}(\overrightarrow{K}^i_{j,j_2}) \\ &= H_2^{((m-k)-(m-k_2))}(H_2^{(m-k_2)}(\overrightarrow{K}^i_{j,j_2})) \\ &= H_2^{(k_2-k)}(\overrightarrow{K}^{i,(j,j_2)}_{k_2})\end{aligned} \tag{7}$$

This second approach is more resilient than the first one since it ensures that nodes cannot decipher, in addition to the messages exchanged between the members with smaller IDs, those exchanged by the nodes with larger IDs. This is because two different Hash Functions are used, one in ascending order of IDs and the other in descending order. Note that, in the same way, we can use the Two-level approach with the pairwise service keys.

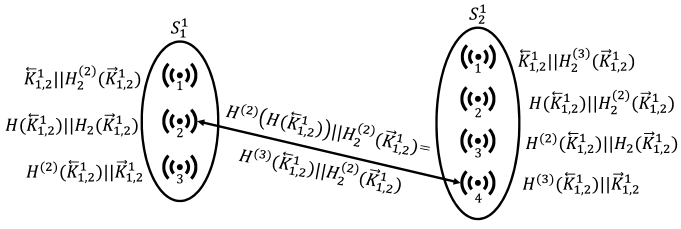


Fig. 8: Example of using Two-level approach.

In the following, we can afford not to put exponents on a notation if there is no risk of ambiguity (For example, we can use u_k instead of $u_k^{i,j}$ if we now that the node is in S_j^i). We also assume that a One-level approach is used.

3) Rekeying upon joining:

We consider a node u_k joining the subgroup S_j of G_i. We assume that G_i contains the nodes participating in the set of services A_i. The KM starts by determining the node ID. Then, it generates some new keys and updates some of the existing ones to ensure the backward secrecy. Next, the KM provides some nodes with the new keys and sends to others the elements allowing them to update some of their keys. The process of rekeying upon joining consists of the following steps.

Key generation: The KM starts by determining the secret key of the joining node. Next, it generates a pairwise key for each existing member of the subgroup S_j. Finally, the KM uses the Hash Function H to generate the $(k-1)$ hash of all pairwise subgroup keys associated to S_j (Formula 1).

Key update: The KM starts by randomly generating a refresh key, K_R. Then, using it and a pseudo-random key derivation function, KDF, the KM updates the subgroup and all the service and pairwise service keys associated to A_i (Formulas 8, 9 and 10 respectively).

$$K_j^+ = KDF(K_j||K_R) \tag{8}$$

$$K^{e+} = KDF(K^e||K_R), (\forall e \in A_i) \tag{9}$$

$$K^+_{i,i_2} = KDF(K_{i,i_2}||K_R), (\forall G_{i_2} \in N, A_{i_2} \cap A_i \neq \emptyset) \tag{10}$$

Key distribution: After the keys generation and update are completed, the KM distributes these new keys to the appropriate nodes. Thus, it sends to each current member, u_{k_2}, of S_j the unicast message $JM1$ encrypted by means of the node secret key. The message contains the ID of the joining node, the pairwise key associated to it, the refresh key and the list of the updated pairwise service keys u_{k_2} knows, L_{k_2}. The KM also broadcasts the message $JM2$ for each other subgroup S_{j_2} (may be in G_i or not) that contains members participating in at least one of the services of A_i. The message is encrypted using the current subgroup key and contains the list of services A_i, the refresh key and the list of the updated pairwise service keys its members know, L_{k_3}. Finally, the KM provides the joining node, via a pre-existing secure channel, with all the keys associated to it and discards K_R.

$$JM1 : KM \rightarrow u_{k_2} :< \{nid_k, K_{k,k_2}, K_R, L_{k_2}\}_{K_{k_2}} >$$

$$JM2 : KM \rightarrow S_{j_2} :< \{A_i, K_R, L_{k_3}\}_{K_{j_2}} >$$

Key installation: When a node u_{k_2} ($u_{k_2} \in S_j$) receives $JM1$, it first decrypts the messages, using its secret key, and installs K_{k,k_2} as the pairwise key to use for encrypting the communication with the joining node. It also uses the KDF and K_R to update the subgroup and all the service keys it knows (Formulas 8 and 9, respectively). The node also replaces the compromised pairwise service keys it knows by those contained in the list L_{k_2}. After that, u_{k_2} discards K_R. On the other hand, when a node u_{k_3} ($u_{k_3} \in S_{j_2}$) receives $JM2$, it first decrypts the message, using the current subgroup key. Then, it uses K_R and the KDF to update, among the service keys it knows, those that are related to A_i. The node also replaces the compromised pairwise service keys it knows by those contained in the list L_{k_3}. Finally, u_{k_3} discards K_R.

4) Rekeying upon leaving:

A node u_k can leave a subgroup S_j, of the group G_i, or be evicted when it gets compromised. In both cases, the keys it knows must be revoked. We assume that G_i contains the nodes participating in the set of services A_i. The KM removes then some of these keys and updates some others. The aim is to ensure the forward secrecy. Indeed, if these keys are not updated, the leaving node will be able to decipher the future messages. Next, the KM provides the remaining nodes with the elements allowing them to remove the keys that must be removed and to update those that must be updated. The process of rekeying upon leaving consists of the following steps.

Key removal: The first step in the rekeying process consists of removing the ID and the secret key of the leaving node as well as all the pairwise keys associated to it.

Key update: The KM starts by randomly generating the refresh key. Then, using the pseudo-random key derivation function, it updates the subgroup and service keys as well as the pairwise service keys related to A_i and the pairwise subgroup keys associated to S_j (Formulas 8, 9, 10 and 11, respectively).

$$K^+_{j,j_2} = KDF(K_{j,j_2}||K_R), (\forall S_{j_2} \in G_i, S_{j_2} \neq S_j) \qquad (11)$$

Key distribution: The KM sends to each node u_{k_2} of G_i ($u_{k_2} \neq u_k$) the unicast message $LM1$ encrypted by means of the node key. The message contains the ID of the leaving node, that of its subgroup, the list of services A_i, the refresh key and the list of the updated pairwise subgroup and service keys u_{k_2} knows, L_{k_2}. The KM also broadcasts the message $LM2$ for each subgroup S_{j_3} ($S_{j_3} \notin G_i$) that contains members participating in at least one of the services of A_i. The message is encrypted using the subgroup key and contains the list of services A_i, the refresh key and the list of all the updated pairwise service keys its members know, L_{k_3}. Finally, the KM discards K_R.

$$LM1 : KM \rightarrow u_{k_2} :< \{nid_k, sid_j, A_i, K_R, L_{k_2}\}_{K_{k_2}} >$$
$$LM2 : KM \rightarrow S_{j_3} :< \{A_i, K_R, L_{k_3}\}_{K_{j_3}} >$$

Key installation: When a node u_{k_2} receives $LM1$, it first decrypts the message using its secret key ans looks at the leaving member's subgroup ID. If it is its cognate, the node starts by removing the pairwise key which was used for encrypting the messages exchanged with it. Next, the node uses K_R and the KDF to update the subgroup key (Formula 8). Furthermore, whether it is the cognate of the leaving member or not, the node u_{k_2} updates all the service keys related to A_i (Formula 9). The node also replaces the compromised pairwise subgroup and service keys it knows by those contained in the list L_{k_2}. Then, it discards K_R. On the other hand, when a node u_{k_3} ($u_{k_3} \in S_{j_3}$) receives $LM2$, it first decrypts the message using the subgroup key. Then, using the KDF, it updates all the service keys related to A_i (Formula 9). The node also replaces the compromised pairwise service keys it knows by those contained in L_{k_3} and discards K_R.

C. Layer 2: Subgroup management

In order to reduce the nodes' storage overhead, each group G_i is partitioned into a set of logical subgroups. It is important to note that these subgrouping is logical and transparent to the application layer. Since the number of services is usually negligible compared to that of nodes and as groups are managed independently of each other, only one group (i.e. one service) is considered in this section. Let us denote by G the group in question, by p the number of its subgroups and by m_j the number of nodes in each of them, S_j. In this case, a node of the subgroup S_j will store one secret key, $m_j - 1$ pairwise node keys, one subgroup key, $p - 1$ pairwise subgroup keys and one service key. The storage is therefore proportional to the sum $p + m_j$.

Two points come out of this. First, regardless of the subgroup to which a node belongs, the value of p is the same. Thus, if it is minimized, storage overheads are reduced on any node of the group. Moreover, the number of keys held by a node depends on the size of its subgroup. Hence, to balance the loads between the nodes of a heterogeneous network, the most constrained ones must be assigned to the smallest subgroups, and conversely. Indeed, for a node to store fewer keys than a more powerful one, the former must be assigned to a subgroup smaller than the one to which the latter belongs.

We focus in this section on the management of heterogeneous subgroups, i.e. subgroups of different sizes, while minimizing their number, p. Note that this does not mean that we do not allow two subgroups to have the same size. To achieve this, we rely on the fact that the nodes of S_j must be able to handle at least $p + m_j$ keys. The size of S_j is then chosen so that $p + m_j$ does not exceed the storage capability of its members or, to put it more simply, the capability mc_j of its weakest node. Indeed, as mc_j is the minimum capability that a member of S_j can have, if its value is greater than $p + m_j$ then all the nodes of S_j will be able to handle the costs. The problem is, therefore, to choose the minimum capabilities of subgroups and to assign them nodes so as to always satisfy:

$$min\ p \qquad (12)$$

$$\text{under duress:} \quad \forall S_j, \quad mc_j \geq p + m_j \qquad (13)$$

1) Storage Capability Evaluation Function:

Before we introduce the heterogeneous subgroup management, we present the Storage Capability Evaluation Function (SCEF) used to evaluate the number of keys a node can store. The SCEF takes as input the storage capability of a node u_k (sc_k), the percentage of memory the protocol can use (pm), and the size of a key (ks). The SCEF takes into account a percentage of the node resources only to balance the cost associated to the KM against other node requirements. This percentage is chosen according to the network and application demands. The SCEF calculates then c_k, the number of keys that can be held by u_k (Formula 14).

$$c_k = pm.\frac{sc_k}{ks} \qquad (14)$$

2) Heterogeneous subgrouping:

The heterogeneous subgrouping management consists of manipulating subgroups of different sizes while minimizing their number and ensuring that the Constraint 13 is always satisfied. To achieve this, a minimum capability mc_j is attributed to each subgroup S_j when created. To satisfy the Constraint 13, $mc_j - p$ nodes are assigned to S_j at most ($m_j \leq mc_j - p$). Note that mc_j must always be greater than p for m_j to be greater than 0. Also, the size of a subgroup varies according to its minimum capability and the value of p. Thus, the greater the capabilities of its members, the larger its size.

A node u_k, that can handle c_k keys, is assigned to S_j only if mc_j is the nearest value less than c_k ($mc_j \leq c_k < mc_j^+$, while mc_j^+ is the value that follows mc_j). Thus, u_k will store $p + m_j$ keys, in the worst case. Since the Constraint 13 is satisfied for S_j and $c_k \geq mc_j$ then $c_k \geq p + m_j$. In other words, u_k can always support the storage overheads. Moreover, thanks to this, the loads are well balanced between the nodes according to their capabilities.

After the assignment, depending on whether S_j is an existing subgroup or a new one, the value of p or m_j increases. It can happen that for a subgroup S_{j_2} (S_{j_2} may be S_j or not) the sum $p + m_{j_2}$ exceeds mc_{j_2} and then some of its members may not be able to handle all the keys anymore. In this case, S_{j_2} is splitted into two subgroups having the same minimum capability mc_{j_2}. The size of the resulting subgroups is equal to the half of m_{j_2} and the Constraint 13 is true again for them. However, S_{j_2} cannot be splitted if it contains only one node. It is then removed and its member is revoked.

Considering the Constraint 13 and the fact that S_j cannot be empty, any node u_k should be able to store at least $p + 1$ keys. On the other hand, if u_k can manage only $p + 1$ keys then it is the only node of S_j and must be revoked when a new subgroup is created. Indeed, if the value of p increases, u_k cannot handle all the keys anymore. For simplicity, we assume that u_k is authorized to join the group only if it can store at least p keys (i.e. $c_k \geq p$ instead of $p + 1$). Therefore, smaller is p, the more likely it is that more constrained nodes can join the group. This is one of the reasons why p should be minimized. For this purpose, depending on the state of the group, subgroups may be merged to reduce their number.

Finally comes the choice of the minimum capabilities of subgroups. The difficulty lies in the fact that subgroups are created and removed as and when required and that the abilities of nodes are not known a priori. We tried then different increasing sequences and found out that the loads are well balanced and p is minimized when the sequence grows exponentially. Indeed, if the minimum capabilities are close to each other, the subgroups will be well balanced but their number will be too large. However, the aim of the subgrouping is precisely to minimize the number of subgroups and thereby reduce the nodes' storage overhead. We then selected two sequences in particular: powers of two and Fibonacci sequence. Note that other sequences can be used as long as they grow exponentially.

If powers of two are used, the group is partitioned so that a minimum capability is the double of the preceding one (Formula 15).

$$mc(l) = \begin{cases} 2.mc(l-1), & \text{if } l > 0. \\ 1, & \text{otherwise.} \end{cases} \quad (15)$$

On the other hand, if a Fibonacci sequence is used, a minimum capability is the sum of the two preceding ones (Formula 16). Note that c_1 and c_2 are arbitrary constants.

$$mc(l) = \begin{cases} mc(l-1) + mc(l-2), & \text{if } l > 1. \\ c_2, & \text{if } l = 1. \\ c_1, & \text{otherwise.} \end{cases} \quad (16)$$

The heterogeneous subgrouping is based on two algorithms. The Assignment Algorithm is run when nodes join the group and assigns them to the right subgroups. It creates new ones when it is necessary and may split others so that the Constraint 13 remains always satisfied. On the other hand, the Reorder Algorithm is executed after a node leaving to reduce the number of subgroups. It then removes those that become empty and merges others to the possible extent. Figure 9 shows an example of a group partitioned using powers of two. Note that the Constraint 13 is satisfied for all the subgroups and the value of p is minimal.

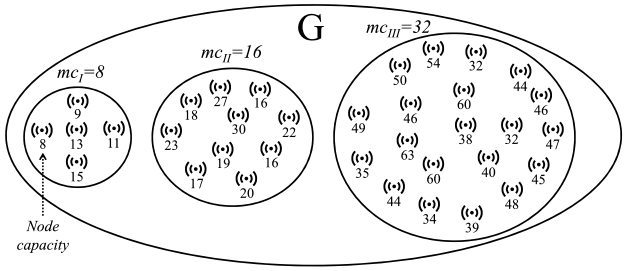


Fig. 9: Example of a group partitioned into three subgroups.

3) Assignment Algorithm:

The Assignment Algorithm (Algorithm 1) is run by the KM when a node u_k is authorized to join the group. It takes as input c_k, the number of keys that can be stored by u_k, and assigns it to a subgroup according to the input value. To achieve this, the algorithm manipulates a list of subgroups, lsg, of size p. Each of its items contains the ID of a subgroup S_j, sid_j, its minimum capability, mc_j, and its size, m_j.

Algorithm 1: Assignment Algorithm

Input : c_k = capability of the node u_k

Round down c_k to the nearest minimum capability mc_k;
Find in lsg a subgroup S_j so that $mc_j = mc_k$;
if *no subgroup is found* **then** Create a new one S_j ;
Assign u_k to S_j;
Update lsg;
while $\exists$ S_{j_2} *for which* $mc_{j_2} < p + m_{j_2}$ **do**
  Split S_{j_2};
end

When u_k is authorized to join the group, the Assignment Algorithm starts by determining the minimum capability mc_k that matches it. It then rounds down c_k to the nearest power of two or term of a Fibonacci sequence. Next, it searches in lsg a subgroup S_j such as $mc_j = mc_k$. If no subgroup is found (this includes the case where the group is empty), a new one is created. Next, the algorithm assigns u_k to S_j, updates lsg and renews the group security material following the steps described in section III-B3.

Also, the algorithm checks if the Constraint 13 is still satisfied for all subgroups. It browses then the list lsg and as long as there is a subgroup S_{j_2} for which $mc_{j_2} < p + m_{j_2}$, it is splitted. The size of the resulting subgroups will then be equal to the half of m_{j_2} and the inequality 13 will be true again for them.

Subgroup creation: Creating a new subgroup S_j consists of determining its ID, sid_j, its key, K_j, and a pairwise subgroup key for every subgroup S_{j_2} of the group. Each of these pairwise subgroup keys, K_{j,j_2}, is encrypted using the key of the subgroup associated to it and sent to its members (message CM).

$$CM : GKM \rightarrow S_{j_2} :< \{sid_j, K_{j,j_2}\}_{K_{j_2}} >$$

Subgroup splitting: Splitting S_j consists first of creating a new subgroup S_{j_2} ($mc_{j_2} = mc_j$). The $\frac{m_j}{2}$ last nodes that have joined S_j are then moved to S_{j_2}. We denote by S_j^+ the subgroup S_j after being splitted and by u_f the first node of S_j to join S_{j_2}, i.e. $\forall u_k \in S_j^+, nid_k < nid_f$ and $\forall u_{k_2} \in S_{j_2}, nid_{k_2} \geq nid_f$.

The algorithm determines first sid_{j_2}. Next, to ensure the forward secrecy, it randomly generates two refresh keys, K_{R_1} and K_{R_2}. Then, using the KDF, it computes K_j^+ and K_{j_2} (Formulas 17 and 18). After that, all the pairwise keys associated to two nodes which no longer belong to the same subgroup are removed. Also, for each subgroup S_{j_3} (including S_j), a pairwise subgroup key K_{j_2,j_3} is created.

$$K_j^+ = KDF(K_j || K_{R_1}) \tag{17}$$

$$K_{j_2} = KDF(K_j || K_{R_2}) \tag{18}$$

Furthermore, the algorithm sends the unicast message $SM1$ to each node $u_k \in S_j$ ($nid_k < nid_f$). The message is encrypted by means of the node secret key and contains K_{R_1} as well as the $(k-1)^{th}$ hash of the pairwise subgroup key K_{j,j_2}. It also sends the unicast message $SM2$ to each node $u_{k_2} \in S_j$ ($nid_{k_2} \geq nid_f$) encrypted using the node secret key. $SM2$ contains K_{R_2} and L_{K_2}, the list of the $(k_2-1)^{th}$ hashes of the pairwise subgroup keys associated to S_{j_2}. Finally, the unicast message $SM3$ is sent to each node u_{k_3} of each subgroup S_{j_3} ($S_{j_3} \neq S_j$ and $S_{j_3} \neq S_{j_2}$). It is encrypted by means of the node secret key and contains the $(k_3-1)^{th}$ hash of the pairwise subgroup key K_{j_2,j_3}.

$$SM1 : KM \rightarrow u_k :< \left\{uid_f, K_{R_1}, H^{k-1}(K_{j,j_2})\right\} K_k >$$

$$SM2 : KM \rightarrow u_{k_2} :< \{uid_f, K_{R_2}, L_{k_2}\} K_{k_2} >$$

$$SM3 : KM \rightarrow u_{k_3} :< \left\{H^{(k_3-1)}(K_{j_2,j_3})\right\} K_{k_3} >$$

4) Reorder Algorithm:

After a node leaving (Section III-B4), the Reorder Algorithm (Algorithm 2) is run in order to reduce the number of subgroups, p. It takes as input the percentage of merging, pcm, and tries to remove or merge subgroups when it is possible. Thus, when a node leaves a subgroup S_j, the algorithm checks the number of the remaining ones. If S_j becomes empty, it is removed. On the other hand, if the size of S_j falls below a certain threshold, thr, the algorithm searches in lsg a subgroup S_{j_2} to merge with S_j. The threshold is the product of the percentage of merging and the maximum size of S_j ($thr = pcm.(mc_j - p)$). Furthermore, S_{j_2} must have the same minimum capability as S_j and its current size must also be less than the threshold. If it is the case, the two subgroups are merged. Note that pcm must not exceed 50% so that the size of the resulting subgroup does not exceed $mc_j - p$. Also, the greater is pcm, the more the subgroups are merged. This increases the merging's cost but reduces the value of p.

Algorithm 2: Reorder Algorithm

```
Input : pcm = percentage of merging
foreach subgroup S_j that a node has left do
    if m_j = 0 then Remove S_j ;
    else
        thr ← pcm.(mc_j − p);
        if m_j < thr then
            Find S_{j_2} such as m_{j_2} < thr and mc_{j_2} = mc_j;
            if a subgroup S_{j_2} is found in lsg then
                Merge S_j and S_{j_2};
            end
        end
    end
end
```

Subgroup removal: Removing a subgroup S_j consists of deleting its ID, sid_j, its key, K_j, and all the pairwise subgroup keys associated to it. The message RM, containing the ID of the subgroup, is then sent to each remaining subgroup so that its members can remove the pairwise subgroup key they share with the nodes of S_j.

$$RM : KM \rightarrow S_{j_2} :< \{sid_j\}_{K_{j_2}} >$$

Subgroup merging: Merging S_j and S_{j_2} consists of three steps. A new subgroup is first created. Next, the members of S_j and S_{j_2} are moved to the new subgroup. New pairwise keys are then generated for every pair of nodes u_k, u_{k_2} ($u_k \in S_j$ and $u_{k_2} \in S_{j_2}$) and sent to them (Messages $MM1$ and $MM2$). These messages are encrypted by means of the node keys. They contain, in addition to the new cognate ID and the pairwise key associated to it, the list of the hashes (L_k or L_{k_2}, respectively) of the pairwise subgroup keys related to the new subgroup. Finally, the two subgroups S_j and S_{j_2} are removed.

$$MM1 : KM \rightarrow u_k :< \{nid_{k_2}, K_{k,k_2}, L_k\}_{K_k} >$$

$$MM2 : KM \rightarrow u_{k_2} :< \{nid_k, K_{k,k_2}, L_{k_2}\}_{K_{k_2}} >$$

D. Layer 3: Service and group management

An IoT service is a transaction between two entities: a provider and a consumer [14]. The former measures the state of the latter or initiates actions which will cause a change to it [44]. The provider is usually a device while the consumer can be a human, the environment or an other device. The main role of the KM is to establish secure communications between the IoT devices. A network member can participate to a service as a provider, a consumer or both. It may also participate to different services, at the same time, and subscribe or unsubscribe from services at any time. The IoT can then be seen as a set of overlapping classes each gathering nodes which collaborate to provide a service and others that benefit from it (Figure 10). As these classes are overlapping, a group of the protocol cannot be associated to a service. Indeed, the independence of the group security parameters will then lose its meaning and the compromise of a node can jeopardize several groups. We propose then the creation of a group for each possible combination of services. A combination A_i of a_i services, of a finite set A of a services, is a subset of a_i elements of A. The number of combinations, nc, is equal to:

$$nc = \sum_{l=1}^{a} C_a^l = 2^a - 1 \tag{19}$$

The network N is then partitioned into groups. Each group G_i is associated with a unique ID, gid_i. It contains then the nodes participating in the services of the combination A_i associated to it. When a current member subscribes or unsubscribes from services, it migrates from a group to another according to its new combination of services. In other words, the node leaves a group (Section III-B4) and joins an other (Section III-B3). Note that repeated operations are done once. The number of groups can reach nc (Formula 19) if there are nodes participating in every possible combination of services. Also, it cannot exceed the number of network nodes, n, because empty groups are not allowed. The maximum number of groups, max_g, is therefore equal to:

$$max_g = Min(2^a - 1, n) \tag{20}$$

Groups are created and removed as and when required and the probability of having only one node in each group is low. Their number can then be much smaller than max_g. In Figure 10, two services E_1 and E_2 coexist in N. Three combinations are then possible: $A_1 = \{E_1, E_2\}$, $A_2 = \{E_1\}$ and $A_3 = \{E_2\}$. Each group G_i contains the nodes participating in the combination of services A_i associated to it (Figure 6).

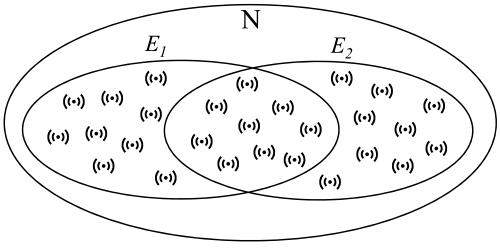


Fig. 10: Network partitioning according to services.

IV. Security analysis

In this section, we analyze the security of our solution and prove that it secures the two modes of IoT Communication.

A. Threat model

A malicious node can be inside or outside the network [6] and may jeopardize the security of both modes of Communication (Figure 11). An outsider node can store the messages exchanged between its members, while it is not allowed to join it, and then decipher them later, when it gains access. Furthermore, an evicted member can still pose a threat to the network, if it is still able to decipher the future communications. Also, if a node inside the network is captured, it may try to decrypt the Group Communication of the members of the other services or the Device-to-Device Communication of the other members of the services to which it belongs.

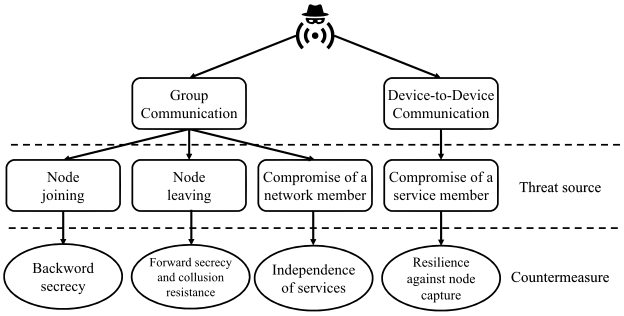


Fig. 11: Threat model and countermeasures

B. Theoretical analysis

The issue is to prove that, for Group Communication, our solution ensures the backward and forward secrecy, resists to collusion attacks and guarantees the secure coexistence of several services in the network. We also show that it is resilient against node capture, for Device-to-Device Communication.

1) Backward secrecy:

We prove that a joining node cannot access the current subgroup and service keys or any previous incarnation of them. The same goes for the pairwise subgroup keys related to its subgroup and the pairwise service keys associated to its group.

Proposition 1: Backward secrecy is guaranteed as the joining node never gets knowledge of the old security material.

Proof: Let us consider a node u_k that joins a subgroup S_j of the group G_i. The KM updates the keys mentioned above. Then, before u_k can actually join the group, the KM rekeys all current members of the network, by means of messages $JM1$ and $JM2$. These messages are encrypted by means of their node and subgroup keys, respectively. Since none of these keys are known to u_k, the joining node is excluded from the process of rekeying.

2) Forward secrecy and collusion resistance:

We prove that leaving nodes cannot access the new subgroup and service keys or any future incarnation of them. The same goes for the pairwise subgroup keys associated to its subgroup and the service keys related to its group.

Proposition 2: Our solution guarantees the forward secrecy and resists to collusion attacks after nodes leaving since they do not have access to the new security material.

Proof: Let us consider a node u_k that leaves a subgroup S_j of the group G_i. The KM rekeys the members of G_i and the rest of the nodes by means of the messages $LM1$ and $LM2$, respectively. The former is encrypted by means of the node keys and the latter using the subgroup keys. Since none of these keys are known to u_k, the leaving node is excluded from the process of rekeying. Furthermore, as these keys are independent of each other, several evicted nodes can not collude to decipher the rekeying messages.

3) Independence of services:

Using our solution, the security parameters of services are independent of each other. This is due, firstly, to the fact that the members of a group participate in the same services and, secondly, because nodes do not share any key if they have no service in common.

Proposition 3: Our solution ensures that the compromise of a service has no effect on the others.

Proof: Let us consider two groups G_i and G_{i_2} associated to the combinations of services A_i and A_{i_2}, respectively. When a member of G_i gets compromised, only the service keys of A_i are exposed. If G_i and G_{i_2} share some services ($A_i \cap A_{i_2} \neq \emptyset$), the keys of services to which the members of G_{i_2} participate but not those of G_i ($A_i \backslash A_{i_2}$) remain secret. Indeed, the compromised node does not know them. Furthermore, if the groups do not share services ($A_i \cap A_{i_2} = \emptyset$), any of the service keys of A_{i_2} gets compromised. In both cases, only the services in which the node participates are exposed and the others remain secret.

4) Resilience against node capture:

We assume that the number of services is negligible and consider then only one group. Although our solution uses a heterogeneous subgrouping, a homogeneous one allows us to evaluate resilience with no significant lack of generality. In this case, the n nodes ($n > 1$) of the group are uniformly distributed in p subgroups of m members each, i.e. $p = m = \sqrt{n}$.

Zero-level approach: Without using Hash Functions, a node shares a distinct pairwise key with each cognate and a single pairwise key for each subgroup. It can then decrypt, in addition to messages intended for it, those that are exchanged between its cognates and the other nodes (Figure 12).

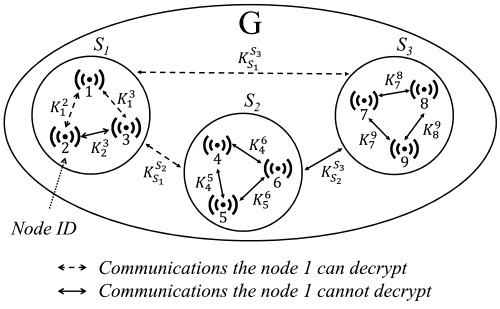


Fig. 12: An example of communications a node can decrypt.

Lemma 1: The number of links a node can decrypt is:

$$D_0 = (\sqrt{n} - 1)(n + 1) \tag{21}$$

Proof: A node can decrypt the communications linking it to the $n - 1$ other group members as well as those between its $\sqrt{n} - 1$ cognates and the $n - \sqrt{n}$ other group members ($D_0 = n - 1 + (\sqrt{n} - 1)(n - \sqrt{n})$).

Lemma 2: The number of links in a group of n nodes is:

$$T = \frac{n(n-1)}{2} \tag{22}$$

Proof: The total number of links is equal to the number of possible combinations that can be obtained by taking two nodes from n ($T = C_n^2$).

Proposition 4: The percentage of compromised links is:

$$P_0 = \frac{2(n+1)}{n(\sqrt{n}+1)} \tag{23}$$

Proof: From Lemmas 1 and 2 and the fact that $P_0 = \frac{D_0}{T}$, we deduce that $P_0 = \frac{2(\sqrt{n}-1)(n+1)}{n(n-1)}$.

One-level approach: Using a Hash Function ensures that nodes cannot decipher the messages exchanged between the nodes that have smaller IDs.

Lemma 3: The number of links a node u_k can decrypt is:

$$D_1 = (\sqrt{n} - 1)(n + 1 - (k-1)^2) \tag{24}$$

Proof: A node u_k cannot decrypt the links that connect each of its $k - 1$ elder cognates with the $k - 1$ members, that have smaller IDs, of each of the $\sqrt{n} - 1$ other subgroups ($D_1 = D_0 - (\sqrt{n} - 1)(k-1)^2$).

Proposition 5: The new percentage of links a compromised node u_k can decipher is:

$$P_1 = \frac{2(n + 1 - (k-1)^2)}{n(\sqrt{n}+1)} \tag{25}$$

Proof: From Lemmas 2 and 3 and the fact that $P_1 = \frac{D_1}{T}$, we deduce that $P_1 = \frac{2(\sqrt{n}-1)(n+1-(k-1)^2)}{n(n-1)}$.

Two-level approach: The Two-level approach makes a member u_k unable to decrypt, in addition to the messages exchanged between the nodes with smaller IDs, those exchanged by the members with larger IDs.

Lemma 4: The number of links a node u_k can decrypt is:

$$D_2 = (\sqrt{n} - 1)(n + 1 - (k-1)^2 - (\sqrt{n} - k)^2) \tag{26}$$

Proof: A node u_k cannot decrypt the links that connect each of its $\sqrt{n} - k$ junior cognates with the $\sqrt{n} - k$ members, that have a larger ID, of each of the $\sqrt{n} - 1$ other subgroups ($D_2 = D_1 - (\sqrt{n} - 1)(\sqrt{n} - k)^2$).

Proposition 6: The percentage of links a compromised node u_k can decipher is:

$$P_2 = \frac{2(n + 1 - (k-1)^2 - (\sqrt{n} - k)^2))}{n(\sqrt{n}+1)} \tag{27}$$

Proof: From Lemmas 2 and 4 and the fact that $P_2 = \frac{D_2}{T}$, we deduce that $P_2 = \frac{2(\sqrt{n}-1)(n+1-(k-1)^2-(\sqrt{n}-k)^2)}{n(n-1)}$.

C. Comparison

To compare the resilience of the three approaches, we consider a group of *10000* members, divided into *100* subgroups of *100* members each. We then compare the variation of the three percentage P_0, P_1 and P_2 according to the node ID. The results are plotted in Figure 13. They show that the use of Hash Functions reduces indeed the rate of compromised links. If a Zero-level approach is used, the percentage is maximal and constant regardless of the node ID. This method is still the best in terms of calculation cost. On the other hand, if a One-level approach is used, the bigger the ID of the captured node, the smaller the percentage of compromised links. This method is therefore interesting if we trust the old nodes more than the new ones. Finally, the Two-level approach provides the best resilience regardless of the node ID.

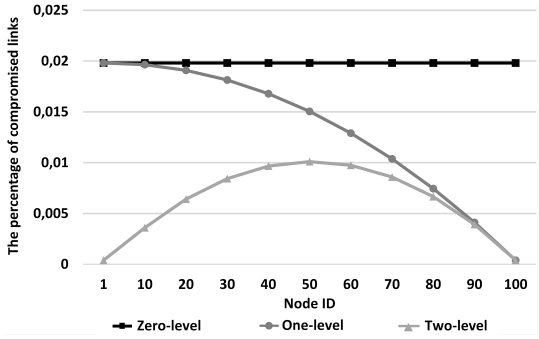


Fig. 13: Variation of the percentage of compromised links according to the node ID.

Now, we compare the resilience provided by our protocol with the perfect resilience offered by some deterministic schemes (e.g. Pairwise Key protocol [17]). Using the Zero-level approach, this percentage is equal to P_0 (Proportional to $\frac{1}{\sqrt{n}}$). On the other hand, a perfect resilience is achieved when each pair of nodes share a distinct key. Thus, a captured node can only decipher the $n-1$ communications linking it to the other network members. The percentage of compromised links is then equal to $\frac{2(n-1)}{n(n-1)} = \frac{2}{n}$ (Proportional to $\frac{1}{n}$). Providing a perfect resilience, none of the existing solutions can do better. Figure 14 shows that the value of P_0 is negligible for large networks ($n > 1000$) such as the IoT. It is even comparable to the rate provided by the perfectly resilient approaches. Our solution offers then a good level of resilience.

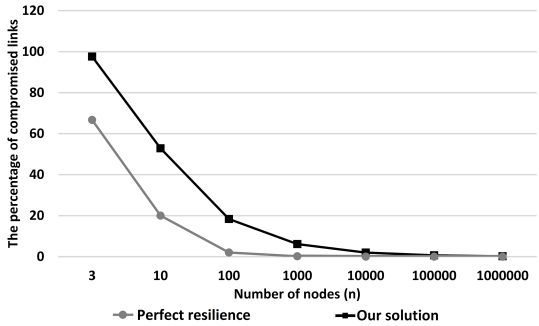


Fig. 14: Variation of the percentage of compromised links according to n.

V. Application and performance evaluation

To give a concrete overview of the performance of our solution, we consider the example of a smart city. This choice is motivated by the fact that a smart city contains a huge number of heterogeneous devices (servers, computers, smartphones, gateways, sensors...etc) spread across the city to provide various services to the benefit of society (healthcare, intelligent transportation system...etc.). These devices can use the two communication modes of the IoT. It is clear that the number of services is generally negligible with respect to the network size. Thus, the number of services does not have a significant influence on the protocol performance compared to that of nodes. For simplicity, we consider then only one group of p subgroups containing m_i members each. We also assume that no Hash Function is used. The KM may then broadcast some messages (e.g. $LM1$) to all the subgroup members instead of sending them in a unicast manner.

A. Theoretical analysis

We begin by briefly analyzing the overheads of our solution on the KM before detailing them on the network members.

1) Overheads on the KM:

The KM can be implemented on servers or gateways in a centralized or decentralized manner to secure communication.

Property 1: The communication overhead on the KM is proportional to the sum $p + m_j$.

Proof: Regardless of the rekeying operation performed (node joining S_j, node leaving S_j,...etc), the KM sends a unicast message to each of the m_j members of S_j and broadcasts a message for each of the other $p-1$ subgroups, in the worst case. The KM sends then a total number of messages proportional to the sum $p + m_j$.

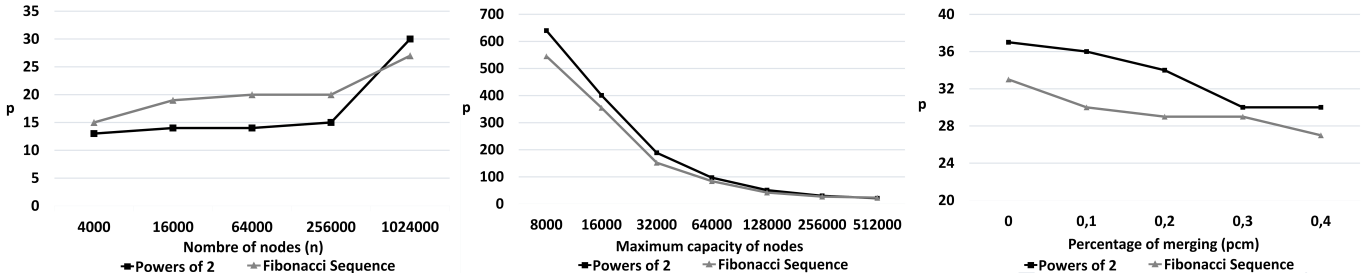
Property 2: The calculation overhead on the KM is proportional to the sum $p + m_j$.

Proof: Regardless of the rekeying operation performed (node joining S_j, node leaving S_j,...etc), the KM updates the keys which are or will be known by the node in question. The calculation overhead on the KM is therefore proportional to the storage cost on nodes, which will be proved to be of the order of $p + m_j$ in the next Section.

Property 3: The storage overheads on the KM is proportional to $O(n)$.

Proof: It is obvious that the number of nodes is more important than that of subgroups or services. Therefore, if we choose not to store the pairwise node keys (which are used for Device-to-Device Communication between the network members) in the KM's memory, the largest number of keys to store will then be that of the node secret keys. The KM will then store a total number of keys proportional to $O(n)$.

Discussion: The communication and calculation costs on the KM are proportional to $p + m_j$. The storage, on the other hand, is of the order of $O(n)$. Considering the significant improvement (presented in the next section) that our solution provides on the node side, the costs on the KM are very reasonable. Also, as the KM has usually plentiful of resources, we aimed to make the costs more affordable on the nodes.



(a) Effect of the number of nodes. (b) Effect of the nodes maximum capability. (c) Effect of pcm.

Fig. 15: Effect of the algorithm parameters on p.

2) Overheads on nodes:

After evaluating the costs of our solution on the KM, let us analyse the overheads on the node side.

Property 4: The communication cost on nodes is $O(1)$.

Proof: Regardless of the rekeying operation performed (node joining, node leaving,...etc), a node receive a constant number of rekeying messages (e.g. $JM1$ or $JM2$ upon a node joining). The communication overhead on nodes is therefore independent of the network size $(i.e. O(1))$. Since communication is the operation that consumes the most energy, our solution is efficient and highly scalable.

Property 5: The calculation overhead on nodes is proportional to the sum $p + m_j$.

Proof: Regardless of the rekeying operation performed (node joining S_j, node leaving S_j,...etc), a node can, in the worst case, update all the keys it knows. The calculation cost on nodes is then proportional to the storage, which will be proved to be of the order of $p + m_j$ in the following.

Property 6: The storage overhead on nodes is proportional to the sum $p + m_j$.

Proof: Using our protocol, a node $u_k^{i,j}$ stores one secret key, $m_j - 1$ pairwise node keys, one subgroup key, $p_i - 1$ pairwise subgroup keys and a negligible number of service and pairwise service keys. The storage overhead on nodes is therefore proportional to the sum $p + m_j$.

Discussion: The communication cost of our solution is $O(1)$, while the calculation and storage overheads are proportional to the sum $p + m_j$. Thus, to reduce these costs as well as those of the KM, we aimed to minimize the number of subgroups, p. We implemented then a simulator to analyze the behaviour of its value according to several parameters.

B. Simulation

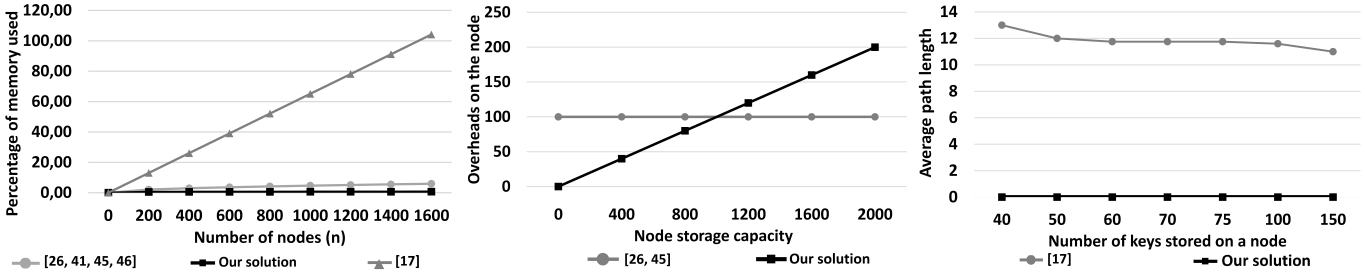
The simulator, we implemented in C, randomly generates node capabilities (based on a uniform distribution) and runs the Assignment Algorithm to assign them. It also simulates nodes leaving and runs the Reorder Algorithm. The simulator takes as inputs the subgrouping type (powers of two or Fibonacci sequence), the number of nodes, nodes maximum capability (the maximum value that the simulator can generate), the percentage of merging. It then divides the nodes into subgroups and outputs their number (p), which is represented by the size of lsg. This allows us to analyze the effect of these parameters on the value of p.

Starting with the network size, we set the values of pcm and the maximum capability to 0.4 and 256000, respectively, and we vary that of n. The results of the simulations are plotted in Figure 15a. They show that, regardless of the network size and the subgrouping type (powers of two or Fibonacci), by using our method of load balancing the number of subgroups is reasonable. Figure 15a shows that even when the size of the network exceeds one million of nodes, the value of p does not exceed a few dozen. This makes our solution scalable since the constrained nodes manipulate a reasonable number of keys.

Next, we analyze the effect of the maximum capability, that can be generated by the simulator, on the number of subgroups. We then set the values of n and pcm to 1024000 and 0.4, respectively, and we vary the maximum capability. The results are plotted in Figure 15b. They show that the more powerful the nodes are, the smaller the value of p is. This is because powerful devices are able to manage more keys and can be assigned to larger subgroups. Note that the larger the subgroups are, the more their number diminishes. Therefore, since the costs of our solution on the constrained nodes mainly depend on the number of subgroups, they are more likely to support the overheads if the network becomes too large. Moreover, even when the maximum capability is small, the value of p remains reasonable for a network containing over a million nodes. To sum up, our solution is scalable regardless of the nodes maximum capabilities but it can become even more if the network contains enough powerful members.

Now, we study the effect of pcm. Thus, we set the values of n and the maximum capability to 1024000 and 256000, respectively, and we vary that of pcm. The results of the simulations are plotted in Figure 15c. They show that the greater the percentage of merging is, the smaller the value of p is. Therefore, the merging operation actually reduces the number of subgroups and makes our solution lighter for the constrained devices and thereby more scalable. Note that most of the overheads imposed by the subgroup merging are at the level of the KM and have no significant influance on the performance of nodes.

Finally, the results of all the simulations performed show that the use of powers of two or a Fibonacci sequence generally gives approximately the same results. However, we noticed that a Fibonacci sequence gives slightly better results for large networks, such as the IoT, and conversely.



(a) Efficiency and scalability. (b) Heterogeneity. (c) Connectivity.

Fig. 16: Comparison.

C. Comparison

In this section, we show that our solution provides the best compromise between the IoT requirements for both modes of IoT communication (Figures 4 and 5).

1) Efficiency and scalability:

Both GKM and PKM schemes need to be efficient and scalable. The communication cost of our solution is $O(1)$ and therefore does not need to be discussed anymore. On the other hand, the calculation cost on nodes is proportional to storage. For these reasons, we only need to analyze the storage costs to compare the efficiency and scalability of our solution to those of the existing schemes. We then take as example a TmoteSky sensor node and compare the storage cost of our solution to different existent GKM (MGKMP [26] and GREP [45]) and PKM (Pairwise Key scheme [17], Trades [41] and Kronecker [46]) schemes. We consider keys of 256 bits (using AES-256 for example). Featuring 48 Kbytes, a TmoteSky can store up to 1536 keys (ignoring the other node's memory requirements).

For the node to support the storage cost of our solution, it is enough if it can store at least p keys. The percentage of storage capability to indicate to the SCEF must then be greater or equal to $P_o = \frac{p}{1536}$. On the other hand, the authors of GREP, MGKMP, Trades and Kronecker show that the storage cost of their solutions are proportional to $O(\sqrt{n})$. The memory rate required to store these $\sqrt{n}$ keys is therefore $P_r = \frac{\sqrt{n}}{1536}$. Finally, as the Pairwise Key scheme do not divide the network members into subgroups, the nodes storage cost is of the order of n. The percentage of memory required is $P_n = \frac{n}{1536}$. We compare then the variation of the three values according to n.

Figure 16a shows that our solution requires less storage on a TmoteSky than the other protocols. Indeed, the value of P_o is smaller than P_r and P_n, no matter the group size. More importantly, if the group contains more than 1536 nodes, the memory of the TmoteSky will not be enough to store all the keys of a Pairwise Key scheme. On the other hand, under the conditions of the simulations, less than 1% of its storage capability is enough if a our solution is used. This is because storage cost is well balanced between the group members according to their capabilities. Thus, by using a bit more of the resources of powerful devices, our solution becomes much lighter for the constrained ones. It can then operate on much larger heterogeneous networks such as the IoT.

2) Heterogeneity for Group Communication:

Unlike most of the existing GKM schemes, our solution balances the loads between the heterogeneous devices of the network according to their capabilities. To illustrate this difference, we consider the protocols MGKMP [26] and GREP [45]. The authors show that their calculation and storage costs are proportional to $O(\sqrt{n})$ for all the network members, while they are both proportional to the nodes' storage capability, using our solution. We consider then a network of 10000 nodes and analyze the variation of the calculation and storage cost according to the node's storage capability (number of keys it can store), for the three protocols. Note that the percentage of storage capability that we choose to indicate to the SCEF is 10% (i.e. only 10% of the real capability of the node is used). The results are plotted in Figures 16b.

We take as example two nodes u_1 and u_2 that can store 200 and 1800 keys, respectively. For both nodes, 10% of their memory is used by our solution, in the worst case. MGKMP and GREP, on the other hand, use 50% of the former and 5% of the latter. As the calculation overhead on node depends on the storage, these protocols quickly exhausts the resources of u_1 while u_2 has much more. More importantly, the nodes having a capability lower than 100 can not even store all the keys, while our solution uses 10% of their memory only. Thus, although the overheads imposed by MGKMP and GREP are lower than that of our solution for powerful devices (capability greater than 1000), they are much greater for the weak ones.

3) Connectivity and flexibility for Device-to-Device Communication:

Unlike most of the existing PKM schemes, our solution ensures a total connectivity coverage. While the probability that two neighboring nodes share a common key does not exceed 0.25 in [41] and is approximately lower bounded by 0.632 in [7], this probability is always equal to 1 using our solution. Indeed, each pair of communicators share a pairwise node or subgroup key and can establish a direct secure link. It provides then a good connectivity. Note that when network connectivity is low, some neighboring relay on intermediate nodes to establish a secure link. The path length represents the number of nodes separating two communicators, which is always equal to zero using our solution. The results presented in [23] give an overview about the average path length between

two nodes using a probabilistic scheme. Figure 16c shows the large gap between the value of this parameter using a probabilistic scheme [17] and our solution, regardless of the group size. It is important to note that the longer the path, the more the communication between nodes requires calculation and energy. This reduces the efficiency of the protocol.

Some PKM schemes [16, 32, 53] are based on the deployment knowledge to maximize the network connectivity. The application of this method is, nevertheless, restrictive if the deployment knowledge is not possible. It is therefore clear that the KM we propose is more flexible as it is based on a logical subgrouping and operates well regardless of the position of nodes. Furthermore, regardless of their category, most of the existing PKM schemes are based on key predistribution. These schemes suffer from poor flexibility as it is hard to add new nodes to the network. Our solution, on the other hand, supports the dynamic deployment of nodes thanks to its rekeying mechanism. Indeed, we previously showed that nodes can join and leave the network at any time without jeopardizing the security of the network. Our solution is therefore more flexible.

VI. Conclusion

In this paper, we presented a novel versatile Key Management protocol for the Internet of Things, which secures both Group and Device-to-Device Communication. For this purpose, our solution is organized into three layers. The upper layer divides the network into multiple groups. Nodes are then assigned to them according to the services to which they subscribe. The middle layer distributes the nodes of each group into logical subgroups. Each of them requires an overhead proportional to the capability of its members. The aim is to balance the loads between the heterogeneous devices according to their capabilities. The lower layer manages the network members and the keys they hold. Keys are classified in two types: Data Encryption Keys ($DEKs$) and Key Encryption Keys ($KEKs$). The $DEKs$ are used to encrypt the data exchanged between nodes. These keys may be either group keys (used to secure Group Communications) or pairwise keys (used to secure Device-to-Device Communication). The $KEKs$, on the other hand, are used to protect the $DEKs$. The aim is to ensure the backward and forward secrecy (for Group Communication) and the dynamic deployment of nodes (for Device-to-Device Communication).

We then analyze the security and performance of our solution and compare it to the existing schemes. For Group Communication, we show that our solution ensures the forward and backward secrecy and, unlike most of the existing GKM protocols, guarantees the secure coexistence of several services in the network. With regard to Device-to-Device Communication, we prove that our solution provides a good level of resilience compared to the existing PKM schemes. We finally demonstrate that, by balancing the loads between the heterogeneous devices according to their capabilities, our solution is both efficient and scalable. It provides then the best compromise between the IoT requirements.

In future works, we intend to decentralize the protocol. Cryptographic material will then be spread across more than one entity in order not to have a single point of failure and to make it more difficult to access or modify this secret material.

Acknowledgments

This work was carried out and funded by the INS2I STFOC project, Heudiasyc UMR CNRS 7253 and the Labex MS2T.

References

[1] F.A. Alaba, M. Othman, I.A.T.A Hashem and F. Alotaibi. “Internet of Things security: A survey”. In: *Journal of Network and Computer Applications* 88 (2017), pp. 10–28.

[2] M. Alagheband and M.R. Aref. “Dynamic and secure key management model for hierarchical heterogeneous sensor networks”. In: *IET Information Security* 6.4 (2012), pp. 271–280.

[3] J. Ayuso, L. Marin, A. Jara and A. F. G. Skarmeta. “Optimization of Public Key Cryptography (RSA and ECC) for 16-bits Devices based on 6LoWPAN”. In: *1st International Workshop on the Security of the Internet of Things, Tokyo, Japan*. 2010, pp. 1–8.

[4] R. Azarderakhsh, A. Reyhani-Masoleh and Z. Abid. “A key management scheme for cluster based wireless sensor networks”. In: *IEEE/IFIP Int. Conf. on Embedded and Ubiquitous Computing, 2008. EUC’08.* Vol. 2. IEEE. 2008, pp. 222–227.

[5] E. Baburaj et al. “Polynomial and multivariate mapping-based triple-key approach for secure key distribution in wireless sensor networks”. In: *Computers & Electrical Engineering* 59 (2017), pp. 274–290.

[6] N. Baracaldo, B. Palanisamy and J. Joshi. “G-sir: an insider attack resilient geo-social access control framework”. In: *IEEE Transactions on Dependable and Secure Computing* (2017).

[7] W. Bechkit, Y. Challal, A. Bouabdallah and V. Tarokh. “A highly scalable key pre-distribution scheme for wireless sensor networks”. In: *IEEE Transactions on Wireless Communications* 12.2 (2013), pp. 948–959.

[8] R. Blom. “An optimal class of symmetric key generation systems”. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1984, pp. 335–338.

[9] C. Blundo, A. De Santis, A. Herzberg, S. Kutten, U. Vaccaro and M. Yung. “Perfectly-secure key distribution for dynamic conferences”. In: *Annual international cryptology conference*. Springer. 1992.

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

[11] C. Chang, H. Yen and D. Deng. “V2V QoS guaranteed channel access in IEEE 802.11 p VANETs”. In: *IEEE Transactions on Dependable and Secure Computing* 13.1 (2016), pp. 5–17.

[12] K. Chatterjee, A. De and D. Gupta. “An improved ID-Based key management scheme in wireless sensor network”. In: *Int. Conf. in Swarm Intelligence*. Springer. 2012, pp. 351–359.

[13] O. Cheikhrouhou. “Secure group communication in wireless sensor networks: a survey”. In: *Journal of Network and Computer Applications* 61 (2016), pp. 115–132.

[14] R. Chen, F. Bao and J. Guo. “Trust-based service management for social internet of things systems”. In: *IEEE transactions on dependable and secure computing* 13.6 (2016), pp. 684–696.

[15] Y. Chen and W. Tzeng. “Group key management with efficient rekey mechanism: a semi-stateful approach for out-of-synchronized members”. In: *Computer Communications* 98 (2017), pp. 31–42.

[16] J. Choi, J. Bang, L. Kim, M. Ahn and T. Kwon. “Location-based key management strong against insider threats in wireless sensor networks”. In: *IEEE Systems Journal* 11.2 (2015), pp. 494–502.

[17] T. Choi, H.B. Acharya and M.G. Gouda. “The best keying protocol for sensor networks”. In: *Pervasive and Mobile Computing* 9.4 (2013), pp. 564–571.

[18] W. Du, J. Deng, Y. S. Han, S. Chen and P. K. Varshney. “A key management scheme for wireless sensor networks using deployment knowledge”. In: *IEEE INFOCOM 2004*. Vol. 1.

[19] W. Du, J. Deng, Y.S. Han, P.K. Varshney, J. Katz and A. Khalili. “A pairwise key predistribution scheme for wireless sensor networks”. In: *ACM Transactions on Information and System Security (TISSEC)* 8.2 (2005), pp. 228–258.

[20] M. Eltoweissy, M.H. Heydari, L. Morales and I.H. Sudborough. "Combinatorial optimization of group key management". In: *Journal of Network and Systems Management* 12.1 (2004), pp. 33–50.

[21] M. Eltoweissy, M. Moharrum and R. Mukkamala. "Dynamic key management in sensor networks". In: *IEEE Communications magazine* 44.4 (2006), pp. 122–130.

[22] M. Eltoweissy, A. Wadaa, S. Olariu and L. Wilson. "Group key management scheme for large-scale sensor networks". In: *Ad Hoc Networks* 3.5 (2005), pp. 668–688.

[23] L. Eschenauer and V.D. Gligor. "A key-management scheme for distributed sensor networks". In: *9th ACM conference on Computer and communications security*. 2002, pp. 41–47.

[24] A. William H. Harb and O.A. El-Mohsen. "Context aware group key management model for internet of things". In: *The Seventeenth International Conference on Networks* 28-34 (2018).

[25] D. Huang, M. Mehta, D. Medhi and L. Harn. "Location-aware key management scheme for wireless sensor networks". In: *Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*. ACM. 2004, pp. 29–42.

[26] M.A. Kandi, H. Lakhlef, A. Bouabdallah and Y. Challal. "An Efficient Multi-Group Key Management Protocol for Internet of Things". In: *26th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*. IEEE. 2018, pp. 1–6.

[27] M.A. Kandi, H. Lakhlef, A. Bouabdallah and Y.Challal. "An Efficient Multi-Group Key Management Protocol for Heterogeneous IoT Devices". In: *IEEE Wireless Communications and Networking Conference (WCNC)*. 2019.

[28] J.Y. Kim, W. Hu, H. Shafagh and S. Jha. "SEDA: Secure over-the-air code dissemination protocol for the internet of things". In: *IEEE Transactions on Dependable and Secure Computing* (2016).

[29] Y. Kung and H. Hsiao. "GroupIt: Lightweight group key management for dynamic IoT environments". In: *IEEE Internet of Things Journal* 5.6 (2018), pp. 5155–5165.

[30] A. Lei, C. Ogah, P. Asuquo, H. Cruickshank and Z. Sun. "A secure key management scheme for heterogeneous secure vehicular communication systems". In: *ZTE Communications* 21 (2016), p. 1.

[31] X.S. Li, Y.R. Yang, M.G. Gouda and S.S. Lam. "Batch rekeying for secure group communications". In: *group* 1 (2001), p. 9.

[32] D. Liu and P. Ning. "Improving key predistribution with deployment knowledge in static sensor networks". In: *ACM Transactions on Sensor Networks (TOSN)* 1.2 (2005), pp. 204–239.

[33] Z. Liu, X. Huang, Z. Hu, M.K. Khan, H. Seo and L. Zhou. "On emerging family of elliptic curves to secure internet of things: ECC comes of age". In: *IEEE Transactions on Dependable and Secure Computing* 14.3 (2017), pp. 237–248.

[34] D. Mall, K. Konaté and A.K. Pathan. "ECL-EKM: An enhanced Certificateless Effective Key Management protocol for dynamic WSN". In: *International Conference on Networking, Systems and Security (NSysS), 2017*. IEEE. 2017, pp. 150–155.

[35] D.A. McGrew and A.T. Sherman. "Key Establishment in Large Dynamic Groups using One-way Function Trees, TIS Labs at Network Associates". In: *Inc. Gleenwood, Maryland* (1998).

[36] L. Militano, G. Araniti, M. Condoluci, I. Farris and A. Iera. "Device-to-device communications for 5G internet of things". In: *EAI Endorsed Transactions on Internet of Things* 15.1 (2015), pp. 1–15.

[37] M.K. Pedhadiya, R.K. Jha and H.G. Bhatt. "Device to device communication: A survey". In: *Journal of Network and Computer Applications* (2018).

[38] Z. Qin, X. Zhang, K. Feng, Q. Zhang and J. Huang. "An efficient identity-based key management scheme for wireless sensor networks using the bloom filter". In: *Sensors* 14.10 (2014), pp. 17937–17951.

[39] S. Rafaeli and D. Hutchison. "A survey of key management for secure group communication". In: *ACM Computing Surveys (CSUR)* 35.3 (2003), pp. 309–329.

[40] S.M.M. Rahman and K. El-Khatib. "Private key agreement and secure communication for heterogeneous sensor networks". In: *Journal of Parallel and Distributed Computing* 70.8 (2010), pp. 858–870.

[41] S. Ruj, A. Nayak and I. Stojmenovic. "Pairwise and triple key distribution in wireless sensor networks with applications". In: *IEEE Transactions on Computers* 62.11 (2012), pp. 2224–2237.

[42] S.H. Seo, J. Won, S. Sultana and E. Bertino. "Effective key management in dynamic wireless sensor networks". In: *IEEE Transactions on Information Forensics and Security* 10.2 (2015), pp. 371–383.

[43] S.R. Singh, A.K. Khan and T.S. Singh. "A New Key Management Scheme for Wireless Senm Networks using an Elliptic Curve". In: *Indian Journal of Science and Technology* 10.13 (2017).

[44] M. Thoma, S. Meyer, K. Sperner, S. Meissner and T. Braun. "On iot-services: Survey, classification and enterprise integration". In: *2012 IEEE International Conference on Green Computing and Communications*. IEEE. 2012, pp. 257–260.

[45] M. Tiloca and G. Dini. "GREP: A group rekeying protocol based on member join history". In: *2016 IEEE Symposium on Computers and Communication (ISCC)*. IEEE. 2016, pp. 326–333.

[46] I. Tsai, C. Yu, H. Yokota and S. Kuo. "Key management in Internet of Things via Kronecker product". In: *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*. IEEE. 2017, pp. 118–124.

[47] L. Veltri, S. Cirani, S. Busanelli and G. Ferrari. "A novel batch-based group key management protocol applied to the internet of things". In: *Ad Hoc Networks* 11.8 (2013), pp. 2724–2737.

[48] D. Wallner, E. Harder and R. Agee. *Key management for multicast: Issues and architectures*. Tech. rep. 1999.

[49] C. Wan. "IBKES: Efficient Identity-Based Key Exchange with Scalability for Wireless Sensor Networks Using Algebraic Signature." In: *Adhoc & Sensor Wireless Networks* 39 (2017).

[50] J. Wang, H. Wang, X. A. Wang and Y. Cao. "An Authentication Key Agreement Scheme for Heterogeneous Sensor Network Based on Improved Counting Bloom Filter". In: *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing*. IEEE. 2015, pp. 815–820.

[51] C.K. Wong, M Gouda and S.S. Lam. "Secure group communications using key graphs". In: *IEEE/ACM transactions on networking* 8.1 (2000), pp. 16–30.

[52] M.F. Younis, K. Ghumman and M. Eltoweissy. "Location-aware combinatorial key management scheme for clustered sensor networks". In: *IEEE transactions on parallel and distributed systems* 17.8 (2006), pp. 865–882.

[53] Z. Yu and Y. Guan. "A robust group-based key management scheme for wireless sensor networks". In: *IEEE Wireless Communications and Networking Conference, 2005*. Vol. 4. IEEE. 2005, pp. 1915–1920.

[54] F. Zhan, N. Yao, Z. Gao and G. Tan. "A novel key generation method for wireless sensor networks based on system of equations". In: *Journal of Network and Computer Applications* 82 (2017), pp. 114–127.

[55] J. Zhang, H. Li and J. Li. "Key establishment scheme for wireless sensor networks based on polynomial and random key predistribution scheme". In: *Ad Hoc Networks* 71 (2018), pp. 68–77.

[56] J. Zhang and V. Varadharajan. "Wireless sensor network key management survey and taxonomy". In: *Journal of network and computer applications* 33.2 (2010), pp. 63–75.