An Efficient Multi-Group Key Management Protocol for Heterogeneous IoT Devices

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

An Efficient Multi-Group Key Management Protocol for Heterogeneous IoT Devices

Mohamed Ali Kandi[1], Hicham Lakhlef[1], Abdelmadjid Bouabdallah[1] and Yacine Challal[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
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 which are able to automatically communicate to computer systems, people and each other providing various services for the benefit of society. These devices have the particularity of being heterogeneous and so have different capabilities in terms of storage, computing, communication and energy. One of the main challenges facing the IoT is how to secure communication between these heterogeneous devices. Among all the issues, the Group Key Management is one of the most difficult. Although different approaches have been proposed to solve it, very few of them consider the heterogeneous nature of the IoT. We propose then a highly scalable Multi-Group Key Management protocol for IoT that ensures the forward and backward secrecy, efficiently recovers from collusion attacks, guarantees the secure coexistence of several services in a single network and balances the loads between its heterogeneous devices according to their capabilities. The evaluation of our solution shows that it is efficient for large-scale heterogeneous networks even if they contain highly resource-constrained devices.**

***Index Terms*—Internet of things, heterogeneity, security, Group Key Management, forward and backward secrecy.**

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. This emerging technology gives rise to revolutionary applications such as health care, environment monitoring, smart cities...etc. The IoT devices have the particularity of being heterogeneous and so have different capabilities in terms of storage, computing, communication and energy. More importantly, most of them are constrained by their small physical size and so have limited memories, computing abilities and energy supply. One of the main challenges facing the IoT is how to secure communication between these heterogeneous devices.

The Group Key Management (GKM) is the core of secure communication. Its main role is to establish secure links between the members of a group. To achieve this, the GKM provides them with a secret cryptographic key that is used to encrypt the data exchanged [20]. Nevertheless, when a member leaves the group, it must no longer be able to decipher the future communications (forward secrecy). Also, if a node joins the group, it must not be able to 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 secret key is revoked and a new one is distributed to the remaining members.

Although different approaches of GKM have been proposed, most of them assume that nodes have the same capability. Hence, they do not balance the charges between them and impose the same overheads on a powerful computer or a weak sensor. Thus, while a negligible part of the former's resources is used by the protocol, those of the latter may not even be enough. Furthermore, the existing protocols usually use the same parameters to secure all communications. Since the IoT provides various services at the same time, communications within a service will be accessible to all network members even those which did not subscribe to it. More importantly, the compromise of a member will jeopardize all services. Finally, the existing solutions rarely consider collusion as a first-class attack and instead resort to a total member reinitialization to recover from it. A collusion attack occurs when multiple compromised nodes cooperate to regain access to the secret key [17]. We propose then a highly scalable Multi-Group Key Management ($MGKM$) protocol for IoT that ensures the forward and backward secrecy, efficiently recovers from collusion attacks, guarantees the secure coexistence of several services in the network and balances the loads between its heterogeneous nodes according to their capabilities. The evaluation of our protocol shows that it is efficient for large-scale heterogeneous networks such as IoT.

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 performance evaluation of our solution. Finally, we conclude in Section V.

II. Related Works

According to the encryption technique used, the Group Key Management schemes can be classified into three categories: symmetric, asymmetric and hybrid [20]. A symmetric approach involves the use of the same key for encryption and decryption, while an asymmetric one uses two different keys.

Generally, symmetric approaches require less computation time, than the asymmetric ones, and are more suitable for networks containing limited resources devices such as wireless sensors [19]. However, most of them suffer from high communication and memory overhead, are not scalable, are not resilient against compromise or inefficiently recover from collusion attacks. Symmetric approaches are usually based on Logical Key Hierarchy (LKH) [6, 16], Exclusion Basis Systems (EBS) [5, 8], polynomials [7], matrices [9, 18]...etc.

On the other hand, asymmetric protocols are more secure and scalable. However, they usually require intensive computing, which makes them impractical on constrained devices. Despite this, some asymmetric schemes were proposed even for wireless sensor networks. Most of them implemented an Elliptic Curve Cryptography (ECC) [1, 13, 15], a CertificateLess Public Key Cryptography (CL-PKC) [12, 14], an ID-Based Encryption (IBE) [4]...etc. Some works [3] proposed hybrid protocols that combine symmetric and asymmetric techniques to take advantages of each and overcome its disadvantages.

Regardless of the type of encryption used, few researches consider the heterogeneous nature of the IoT [1, 2, 11, 13] and yet they usually divide the group into two classes only: powerful and constrained nodes. In a previous work [10], we proposed a $MGKM$ protocol for IoT which is based on the member join history [17]. The main idea of the protocol is to manage several groups with independent security parameters. Thus, different services can coexist in the network without jeopardizing each other. Nevertheless, our previous solution does not consider the heterogeneous nature of the IoT devices and distributes them uniformly on the subgroups of each group. The overheads are then the same on the powerful devices and the weak ones. In this paper, we propose a novel highly scalable $MGKM$ protocol which, in addition to ensuring the forward and backward secrecy, efficiently recovering from collusion attacks and guaranteeing the secure coexistence of several services in the network, balances the loads between the nodes according to their capabilities.

III. Our solution

Our solution uses two layers. The upper layer manages multiple groups and assigns nodes to them according to the services to which they subscribe. On the other hand, the lower layer distributes the nodes of each group into logical subgroups in order to reduce the protocol overheads on them. The network is then divided into several groups, each of which is also partitioned into several subgroups. By doing this, the security parameters of services will be independent and the protocol is lighter for the network nodes.

Each group is associated with an ID which is unique within the network. It contains then the nodes participating in a given combination of services. When a node joins the network, it is assigned to a group according to the combination of services to which it subscribes. On the other hand, if a current member subscribes or unsubscribes from services, it migrates from a group to another according to its new combination of services.

Each group is in turn partitioned into subgroups. It is important to note that these subgrouping is logical and transparent to the application layer. The aim behind this is to efficiently rekey the network when necessary. A subgroup S is associated with an ID, sid_S, which is unique within the group and reflects its subgroups total order. Given two subgroups S and T, $sid_S < sid_T$ if and only if S was created before T. Thereby, S is considered as an elder kindred of T whereas the latter is seen as a junior kindred of the former. Also, S is associated to two tokens: a forward, st_S^F, and a backward one, st_S^B.

When a node u joins a group G, it is assigned to one of its subgroup S. It is then associated with an ID, nid_u, which is unique within S and reflects, its members' total order. Given two nodes u and v, $nid_u < nid_v$ if and only if u has joined S before v. Thus, u is considered as an elder cognate of v whereas v is seen as a junior cognate of u. Also, u is associated to two tokens: a forward, t_u^F, and a backward one, t_u^B. Note that u does not know neither its tokens (t_u^F and t_u^B) nor those of S (st_S^F and st_S^B). However, it stores the backward and forward node tokens associated to its elder and junior cognates and the backward and forward subgroup tokens of the elder and junior kindred of S, respectively. The node also holds a secret key, K_u, and shares a subgroup key, K_S, with its cognates and a group key, K_G, with all the nodes of G. Finally, u stores a key for each service in which it participates.

Our solution manages several groups having independent security parameters so that the compromise of a service has no effect on the others. Security within a single group is based on the fact that nodes do not know their tokens. Thus, when a node is compromised or when a collusion attack occurs, the tokens of the compromised node(s) remain secret. The $MGKM$ can then rely on them to efficiently rekey the group. Due to space constraints, we only detail the subgroup management in this paper. For more details about services, groups and nodes management (nodes joining/leaving and the recovery from collusion attacks) please refer to our previous work [10]. Furthermore, since groups are managed independently of each other, only one group (i.e. one service) is considered in this paper. Finally, we assume that keys and tokens have the same size and use the two terms interchangeably.

Although our previous solution organizes nodes into subgroups, the subgrouping is homogeneous. The n nodes of a group are then uniformly distributed in p subgroups of m members each, i.e. $p = m = \sqrt{n}$. We then showed that the storage, computing and energy costs for a node vary according to $p + m = 2.\sqrt{n}$. Hence, a node has an overhead of $O(\sqrt{n})$ (the performance evaluation is detailed in [10]). However, in heterogeneous networks wherein nodes have different capabilities, the same costs are imposed on powerful and weak devices. This exhausts the resources of the constrained ones which can significantly degrade network performance and shorten its lifetime. More importantly, when the network becomes too large, it may happen that some constrained nodes cannot support the costs at all, while others can handle much more.

Since the costs on nodes depend on the size of their subgroups ($p + m$), we propose the use of a heterogeneous subgrouping. Thus, according to their capabilities, nodes of a heterogeneous network are distributed in subgroups having different sizes to balance the loads between them. Thereby, the costs will be less important on constrained nodes. Network performance is then improved and its lifetime increased. Moreover, the constrained nodes are more likely to support the overheads if the network becomes too large. On this basis, we propose a novel highly scalable $MGKM$ protocol for IoT. Thus, by using a bit more of the resources of powerful devices, our solution becomes much lighter for the constrained ones.

A. Formulation of the problem

Using our solution, the number of keys manipulated (stored, hashed...etc) by the nodes of a subgroup S, of size m_s, is proportional to $p + m_s$ [10]. Thus, two points come out of this. First, if the value of p is minimized, overheads are reduced on any node of the group. Moreover, the number of keys manipulated 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.

Therefore, we focus in this work 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 all the members of S must be able to handle at least $p + m_s$ keys. The size of S is then chosen so that $p + m_s$ does not exceed the capability of its members or, to put it more simply, the capability mc_s of its weakest node. Indeed, as mc_s is the minimum capability that a member of S can have, if its value is greater than $p + m_s$ then all the nodes of S will be able to handle the overheads. The problem is, therefore, to choose the minimum capabilities of subgroups and to assign them nodes so as to always satisfy:

$$min\ p \tag{1}$$

$$\text{under duress:} \quad \forall S, \quad mc_s \geq p + m_s \tag{2}$$

B. Capability Evaluation Function

The Capability Evaluation Function (CEF) is used to evaluate the number of keys a node can handle. Depending on the network requirements, several parameters can be taken into account. We choose then three of them: memory, processing and energy. Thus, the CEF we propose takes as input the following arguments: the storage capability of a node u (sc_u) and the amount of data that it can process per unit time (adt_u) and per unit energy (ade_u). Note that the CEF takes into account a percentage of the node resources only to balance the overhead associated to the $MGKM$ against other node requirements. To calculate c_u, the number of keys that can be managed by u, the CEF determines the minimum between the number of keys, of size ks, that the node can store ($\frac{sc_u}{ks}$) and the number of keys that it may calculate per unit time ($\frac{adt_u}{ks}$) and per unit energy ($\frac{ade_u}{ks}$) (Formula 3). According to the network and application requirements, weighting can be given to each parameter.

$$c_u = Min(\frac{sc_u}{ks}, \frac{adt_u}{ks}, \frac{ade_u}{ks}) \tag{3}$$

C. Heterogeneous subgrouping

The heterogeneous subgrouping management consists of manipulating subgroups of different sizes while minimizing their number and ensuring that the Constraint 2 is always satisfied. A minimum capability mc_s is then attributed to each subgroup S when created. To satisfy the Constraint 2, $mc_s - p$ nodes are assigned to S at most. Note that the size of a subgroup varies according to its minimum capability. Thus, the greater the capabilities of its members, the larger its size.

A node u, that can handle c_u keys, is assigned to S only if mc_s is the nearest value less than c_u ($mc_s \leq c_u < mc_s^+$, mc_s^+ is the value that follows mc_s). Thus, u will have storage, computing and energy overheads proportional to $p + m_s$, instead of $2.\sqrt{n}$ (if a homogeneous subgrouping is used). Note that depending on the values of p and m_s, $p + m_s$ can be either greater or lower than $2.\sqrt{n}$. However, since the Constraint 2 is satisfied for S, u can always support the costs.

After the assignment, depending on whether S is an existing subgroup or a new one, the value of p or m_s increases. It can happen that for a subgroup T (T may be S or not), the sum $p + m_t$ exceeds mc_t, and then some of its members may not be able to handle all the keys anymore. In this case, T is splitted into two subgroups having the same minimum capability mc_t. The size of the resulting subgroups is equal to the half of m_t and the Constraint 2 is true again for them.

Considering the Constraint 2 and the fact that S cannot be empty, a node u should be able to handle at least $p + 1$ keys. If u can manage only $p + 1$ keys then it is the only node of S and must be revoked when a new subgroup is created. Indeed, if p increases, u cannot handle the keys anymore. For simplicity, we assume that u is authorized to join the group only if $c_u \geq p$ (instead of $p + 1$). Therefore, smaller is p, the more likely it is that more constrained nodes can join the group. Subgroups may then be merged to reduce their number.

Finally comes the choice of the minimum capabilities. 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 different increasing sequences and found that the loads are well balanced and p minimized if the sequence grows exponentially. We then selected two sequences in particular: powers of two and Fibonacci sequence. The group may be partitioned so that a minimum capability is the double of the preceding one or the sum of the two preceding ones. Figure 1 shows an example of group partitioned using powers of two. The Constraint 2 is satisfied for all the subgroups and the value of p is minimal.

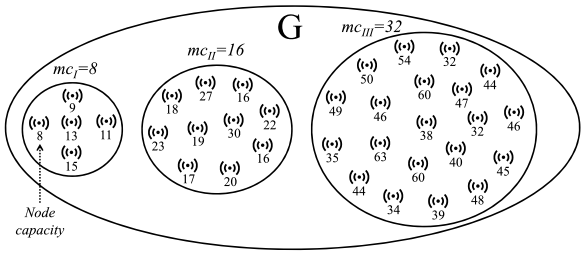

Fig. 1: Example of a group partitioned into three subgroups.

D. Assignment Algorithm

The Assignment Algorithm (Algorithm 1) is run by the $MGKM$ when a node u is authorized to join the group. It takes as input c_u, the number of keys that can be managed by u, and assigns it to a subgroup S 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, sid_s, its minimum capability, mc_s, and its size, m_s.

When u is authorized to join the group, the Assignment Algorithm starts by determining the minimum capability mc_u that matches it. It then rounds down c_u to the nearest power of two or term of a Fibonacci sequence. Next, it searches in lsg a subgroup S such as $mc_s = mc_u$. If no subgroup is found, a new one is created. It consists of determining its ID, sid_s, its key, K_S, its subgroup tokens, st_S^B and st_S^F, and to send the last two to all the nodes of the group, encrypted by means of K_G. Next, the algorithm assigns u to S and renews the group security material, following the steps described in our previous work [10], and updates lsg. Also, the algorithm checks if the Constraint 2 is still satisfied for all the subgroups. It browses then the list lsg and as long as there is a subgroup T for which $mc_t < p + m_t$, the latter is splitted.

Algorithm 1: Assignment Algorithm

Input : c_u = capability of the node u
Round down c_u to the nearest minimum capability mc_u;
Find in lsg a subgroup S so that $mc_s = mc_u$;
if *no subgroup is found* **then** Create a new one S ;
Assign u to S;
Update lsg;
while $\exists$ T *for which* $mc_t < p + m_t$ **do**
  Split T;
end

Subgroup splitting: Splitting S consists first of creating a new subgroup T ($mc_t = mc_s$). The $\frac{m_s}{2}$ last nodes that have joined S are then moved to T. We denote by S^+ the subgroup S after being splitted and by f the first node of S to join T, i.e. $\forall u \in S^+, nid_u < nid_f$ and $\forall v \in T, nid_v \geq nid_f$.

The algorithm determines first sid_t and st_T^B. Next, to ensure the forward secrecy, it randomly generates a master subgroup token, st_M, and two refresh keys, K_{RS} and K_{RT}. Then, using a pseudo-random key derivation function (KDF), it computes st_T^F, K_S^+ and K_T (Formulas 4, 5 and 6). Furthermore, the algorithm uses a one-way hash function (H) to update the forward tokens associated to the nodes moving to T and the backward tokens associated to the others. Also, as T is the last subgroup to be created, its members should no longer know any forward subgroup tokens but must receive the list of all the backward ones, LBS. Thus, the algorithm uses H to update the forward subgroup tokens known by the members of S and all the backward subgroup tokens. Finally, the algorithm sends the unicast messages $SM1$ and $SM2$ to each node $u \in S$ ($nid_u < nid_f$) and $v \in S$ ($nid_v \geq nid_f$), respectively. It also sends $SM3$ to each subgroup U ($U \neq S$ and $U \neq T$).

$$st_T^F = KDF(st_M || K_{RS}) \quad (4)$$

$$K_S^+ = KDF(K_S || K_{RS}) \quad (5)$$

$$K_T = KDF(K_S || K_{RT}) \quad (6)$$

$$SM1 : MGKM \rightarrow u :< nid_m, sid_t, \{st_M, K_{RS}\}\, K_u >$$

$$SM2 : MGKM \rightarrow v :< nid_m, sid_s, \{K_{RT}, LBS\}\, K_v >$$

$$SM3 : MGKM \rightarrow U :< sid_s, sid_t, \{st_M, K_{RS}\}\, K_U >$$

Forward secrecy: When a subgroup S is splitted, it must be rekeyed as well as the new subgroup T. Since no node joins the group and none of its members leaves it, it is not necessary to rekey all the group. The issue is then to prove that the nodes moving from S to T cannot get access to the new key of S, or any future incarnation of it, and that they are the only ones to receive the key of T. The $MGKM$ rekeys the members of S and T using the unicast messages $SM1$ and $SM2$ that are encrypted by means of their secret keys. Therefore, the moving nodes cannot decrypt $SM1$ and are excluded from the rekeying of S. Furthermore, the remaining ones do not have access to the key of T as they cannot decrypt $SM2$.

E. Reorder Algorithm

In order to reduce the number of subgroups, p, the Reorder Algorithm (Algorithm 2) is run after a node leaving or a recovery from a collusion attack. It takes as input the percentage of merging, pcm, and tries to remove or merge subgroups when it is possible.

When nodes leave a subgroup S, the algorithm checks the number of the remaining ones. If S becomes empty, it is removed by deleting its key and tokens and informing the nodes to remove either st_S^F or st_S^B. On the other hand, if the size of S falls below a certain threshold, the algorithm searches in lsg a subgroup T to merge with S. The threshold is the product of the percentage of merging and the maximum size of S ($thr = pcm.(mc_s - p)$). Furthermore, T must have the same minimum capability than S 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 $mc_s - p$. 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 *that one or more nodes have left* **do**
  if $m_s = 0$ **then** Remove S ;
  else
    $thr \leftarrow pcm.(mc_s - p)$;
    if $m_s < thr$ **then**
      Find T such as $m_t < thr$ and $mc_t = mc_s$;
      if *a subgroup* T *is found in* lsg **then**
        Merge S and T;
      end
    end
  end
end

Subgroup merging: Two subgroups with the same minimum capacity can be merged if they contain a number of nodes that is below a certain threshold. If we assume that $sid_S < sid_T$, merging S and T consists of deleting the latter after moving its nodes to the former. We denote by S^+ the subgroup S after being merged.

The $MGKM$ starts by generating a refresh key. Then, using the KDF, it updates K_S and K_T (Formulas 7 and 8) and utilizes them to compute the new subgroup key (Formula 9). Next, the $MGKM$ sends $MM1$, encrypted using K_S to the nodes of S. The message contains: the list of IDs of T and its members, LID_T, the updated subgroup key of T, K_T^* (Formula 8), and the list of the updated forward node tokens associated to the members of T, FNT_T. Also, the $MGKM$ sends, to the members of T, the message $MM2$ encrypted by means of K_T. It contains: the list of the IDs of S and all its members, LID_S, the updated subgroup key of S, K_S^* (Formula 7), the list of the updated backward node tokens associated to the nodes of S, BNT_S, and the list of the updated forward subgroup tokens, FST_U, of each subgroup U for which $sid_s < sid_u < sid_t$. Finally, the $MGKM$ broadcasts $MM3$, updates the backward subgroup tokens the nodes of T know but not those of S and deletes the tokens and key of T.

$$K_S^* = KDF(K_S||K_R) \quad (7) \qquad K_T^* = KDF(K_T||K_R) \quad (8)$$

$$K_S^+ = KDF(K_S^*||K_T^*) \quad (9)$$

$$MM1 : MGKM \rightarrow S < LID_T, \{K_T^*, FNT_T\}\, K_S >$$

$$MM2 : MGKM \rightarrow T < LID_S, \{K_S^*, BNT_S, FST_U\}\, K_T >$$

$$MM3 : MGKM \rightarrow G < sid_s, sid_t >$$

Backward secrecy: When the subgroups S and T are merged, only S must be rekeyed. Indeed, no new node joins G and none of its members leaves it. We need then to prove that the moving nodes cannot access the current key of S, or any previous incarnation of it. Thus, the $MGKM$ rekeys S and T by means of messages $MM1$ and $MM2$ encrypted using their current keys. Thus, nodes of S can decrypt $MM1$, retrieve its secret material and generate the new subgroup key of S. Moreover, nodes of T can decrypt $MM2$ and compute the new subgroup key of S without knowing the current one.

IV. Performance evaluation

To reduce the overheads on nodes and to increase the chances that the constrained ones support the protocol, the value of p must be minimized. The execution times of the Assignment and Reorder Algorithms also depend on p, as they mainly traverse the list of subgroups. Thus, to evaluate the performance of our solution and show that, in a heterogeneous network, it has improved, we analyze the behaviour of p according to several parameters and compare the obtained results to $\sqrt{n}$ (the number of subgroups if a homogeneous subgrouping is used). To give a concrete overview of these performances, we take as example a TmoteSky sensor node and compare the costs of the two protocols on it. To achieve this, we implemented a simulator, using C, that 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 type of subgrouping (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 and outputs p.

A. Effect of the parameters of the simulation on p

In this section, we analyze the effect of two of the parameters of the simulation on p: the maximum capability that can be generated and the percentage of merging, pcm. Starting with the maximum capability, we 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 2. Note that the more the group contains powerful nodes, the smaller the value of p. However, even when the maximum capability is small, the value of p is lower than $\sqrt{n}$. Our solution gives then better results even for networks of constrained nodes.

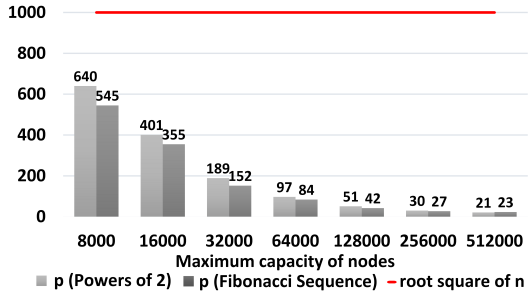

Fig. 2: Effect of the maximum capacity of nodes on p.

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 3. They show that actually the merging reduces the number of subgroups. Indeed, the greater the value of pcm, the smaller that of p.

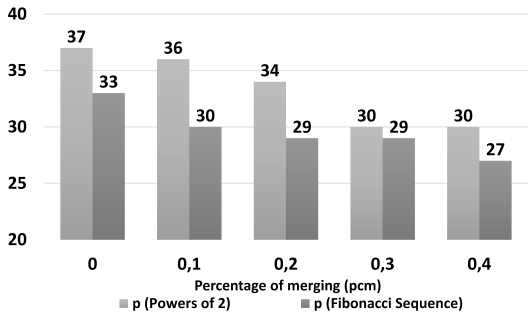

Fig. 3: Effect of pcm on p.

Finally, the results of all the simulations performed show that, in general, the use of powers of two or a Fibonacci sequence gives approximately the same results. However, we noticed that a Fibonacci sequence gives slightly better results for large groups (more than a million nodes) and conversely.

B. Concrete example: TmoteSky sensor node

To give a concrete overview of the performance of our solution, we take as example a TmoteSky sensor node and compare the overheads of the two protocols (the new and the previous one) on it. Due to space constraints, we only discuss the storage overhead. Computing and energy costs also depend on p. The same method can then be used to discuss them.We consider keys of 256 bits (using AES-256 for example). Thus, featuring 48 Kbytes, a TmoteSky can store at most 1536 keys.

For the node to support the storage cost of a heterogeneous subgrouping, it is enough if it can store at least p keys. The percentage of storage capability to indicate to the CEF must then be greater or equal to $p_{het} = \frac{p}{1536}$. On the other hand, if a homogeneous subgrouping is used, the percentage of memory required to store $2.\sqrt{n}$ keys is $p_{hom} = \frac{2.\sqrt{n}}{1536}$. We compare then the variation of the two values according to n. To achieve this, we set the values of pcm and the maximum capability to 0.4 and 256000, respectively, and we vary that of n.

The results (Figure 4) show that our method of load balancing requires much less storage space than using subgroups of the same size. Indeed, the value of p_{het} is much smaller than p_{hom} whatever the size of the group. More importantly, if the group contains more than five hundred thousand nodes and a homogeneous subgrouping is used, a TmoteSky cannot join the group because it does not have enough memory space. On the other hand, under the conditions of the simulations, only 2% of its storage capability is enough if a heterogenous subgrouping is used. Therefore, our protocol becomes much more scalable by implementing the new subgroup management. Our solution can then operate on much larger heterogeneous networks, which makes it well suitable for the IoT.

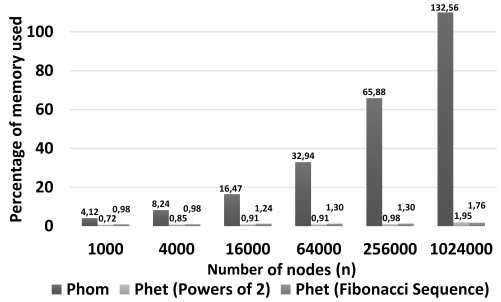

Fig. 4: Effect of the size of the group on p_{het} and p_{hom}.

V. Conclusion

In this paper, we presented a highly scalable Multi-Group Key Management protocol which ensures the forward and backward secrecy, efficiently recovers from collusion attacks, guarantees the secure coexistence of several services in the network and balances the loads between its heterogeneous nodes according to their capabilities. To achieve this, the network is divided into independent groups, each of which is also partitioned into several subgroups. A group contains the nodes participating in a given combination of services. A subgroup requires an overhead proportional to the capability of its members. When a node joins a group, the protocol affects it to the subgroup that matches its capability. Subgroups are created and can be splitted when it is necessary. On the other hand, when a node leaves a group or when a set of colluding nodes are evicted, the protocol tries to reduce the number of subgroups. It removes then those that become empty and merges the others to the extent possible. The evaluation of our solution shows that it is efficient for large scale heterogeneous networks even if they contain highly constrained devices. In future works, we intend to decentralize the protocol in order not to have a single point of failure.

Acknowledgments

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

References

[1] 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.

[2] S. Athmani, A. Bilami, and D.E. Boubiche. “EDAK: An Efficient Dynamic Authentication and Key Management Mechanism for heterogeneous WSNs”. In: *Future Generation Computer Systems* (2017).

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

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

[5] C. Chen, Z. Huang, Q. Wen, and Y. Fan. “A novel dynamic key management scheme for wireless sensor networks”. In: *2011 4th IEEE International Conference on Broadband Network and Multimedia Technology (IC-BNMT)*. IEEE. 2011, pp. 549–552.

[6] G. Dini and I.M. Savino. “S2rp: a secure and scalable rekeying protocol for wireless sensor networks”. In: *2006 IEEE International Conference on Mobile Adhoc and Sensor Systems*. 2006, pp. 457–466.

[7] Abdoulaye Diop, Yue Qi, and Qin Wang. “Efficient group key management using symmetric key and threshold cryptography for cluster based wireless sensor networks”. In: *International Journal of Computer Network and Information Security* 6.8 (2014), p. 9.

[8] T. Divya R.and Thirumurugan. “A novel dynamic key management scheme based on hamming distance for wireless sensor networks”. In: *2011 International Conference on Computer, Communication and Electrical Technology (ICCCET)*. IEEE. 2011, pp. 181–185.

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

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

[11] F. Kausar, S. Hussain, L.T. Yang, and A. Masood. “Scalable and efficient key management for heterogeneous sensor networks”. In: *The Journal of Supercomputing* 45.1 (2008), pp. 44–65.

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

[13] S. 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.

[14] S. 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.

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

[16] J. Son, J. Lee, and S. Seo. “Topological key hierarchy for energy-efficient group key management in wireless sensor networks”. In: *Wireless personal communications* 52.2 (2010), p. 359.

[17] 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.

[18] 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*, pp. 118–124.

[19] 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.

[20] 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.