Efficient and lightweight group rekeying protocol for communicating things

Hicham Lakhlef, Abdelmadjid Bouabdallah

Version of Record: https://www.sciencedirect.com/science/article/pii/S0045790621000458
Manuscript_4d13070d7bdb4b057c0fcaf0bb5d7eff

Efficient and Lightweight Group Rekeying Protocol for Communicating Things

Hicham Lakhlef, Abdelmadjid Bouabdallah
Sorbonne Universités, Université de Technologie de Compiègne, CNRS,
CS 60319, 60203 Compiègne, France
E-mails: hlakhlef@utc.fr, madjid.bouabdallah@hds.utc.fr

Abstract

Recent advances in emerging technologies have given birth to a new type of networks called *Internet of Things* (IoT). The emergence of IoT has led practitioners to envision networking a large number of sensors for event monitoring, data collection and filtering. Due to the use of wireless technologies, a secure communication is strongly needed to protect valuable information. Secure group communication is one of the most significant requirements for IoT applications. This employs a group rekeying mechanism for a secure and efficient delivery of data. In this paper, we present a new efficient and scalable rekeying protocol for wireless communicating things. This protocol uses $O(\log_2 \sqrt[3]{n})$ memory complexity, improving significantly the literature works, where n is the network size. Simulation results show that our protocol has better performances than existing works in several criteria.

Keywords: Communicating Things; Message-passing; Rekeying; Energy-efficiency; Security; Group Communication; Memory complexity

1. Introduction

The Internet of Things (IoT) can be composed of a variety of communicating objects.These communicating objects will be found in all areas, ranging from the public domain such as the objects of our homes, which are becoming more intelligent, to the smart city in general. Roughly speaking, IoT is making our

daily life easier and smarter. However, one of the main challenges facing the IoT is how to secure the communication links between these communicating objects. Indeed, the design of protocols to control them in order to achieve a common goal is far from being a simple task. In fact, due to resources limitation, a solution for an application in IoT devices should take into account the restrained capabilities (limited battery power, processing power and memory storage) of these devices by using as little memory and energy as possible, whilst maximize the life time of the network members [1].

To transmit data from a source node to several destinations, things can use *multicast* operation. The multicast operation in wireless networks allows to create efficient group communications. The broadcasting medium, however, makes the wireless network vulnerable to various security attacks since anyone can easily eavesdrop on messages transmitted in the air [2]. In order to implement the multicast, we need to an access control mechanism for the broadcast of messages. To implement an access control mechanism for a secure group communication we should employ a symmetric key, known as a group key, shared only by nodes in the group. However, the questions that arise are how to manage this key in case of joining and leaving of nodes ? And how to prevent from attacks of colluding nodes ?

The Group Key Management (GKM for short) is the core of secure communication. Its main role is to establish secure communication links between the members of a same group. To achieve this, the GKM provides them with a secret cryptography key that is used to encrypt the data exchanged between nodes belonging to the same group. 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 re-keying protocols. Thus, when a node joins or leaves the group, the group is updated, the secret key is revoked and a new one is distributed to the remaining members [4, 5].

Our contribution. The aim of our proposal is to present a new group key management that uses a good memory complexity. In addition, it must prevent multiple evicted nodes to cooperate to regain access to the current group key. Such an attack is referred to as collusion attack. In this paper, we present a new lightweight and efficient re-keying protocol for wireless communicating things. We propose a highly scalable GKM protocol for IoT devices that ensures the forward and backward secrecy, efficiently recovers from collusion attacks, and uses a good of memory complexity for each node. This protocol uses $O(\log_2 \sqrt[3]{n})$ memory complexity, where n is network size, improving significantly the literature works.

The rest of this paper is organized as follows: Section 2 discusses the related works. Section 3 presents the model and some definitions. Section 4 presents the proposed protocol. Section 5 presents the simulation results and comparison with the related works. Finally, Section 6 concludes our paper.

2. Related works

According to the encryption techniques used, the group key management schemes for disseminating group key can be classified into the following three categories: symmetric, asymmetric and hybrid.

2.1. Symmetric approaches

A symmetric approach involves the use of the same key for encryption and decryption. Symmetric approaches can, in turn, be classified into two categories: *pairwise key schemes* and *group key schemes*. Pairwise Key schemes consist of using a distinct key for each pair of nodes. This approach has the advantage of being resilient, since the compromise of a node will not jeopardize the communications of the other ones. However, a node has to store as many keys as the number of members in the network. It is also hard to add new nodes to the network because new keys need to be distributed to all the actual members. This approach is therefore not scalable. The pairwise key scheme was first introduced in [3]. Other pairwise key approaches were then proposed.

They are usually based on polynomials [6], matrices [7], Hierarchy LKH [8] ...etc. Generally, symmetric schemes require less computation time than the asymmetric ones and are then more suitable for limited resources devices like IoT. However, most of them suffer from high communication and memory overhead, which makes them not scalable. Moreover, these symmetric schemes are rarely dealing with collusion attacks. Recently, the authors in [9, 15] proposed efficient group rekeying protocols based on member join history. A group is divided into subgroups, one member belongs to only one subgroup and not moved to other subgroup. While joining a subgroup unique identifier is given.

2.2. Asymmetric approaches

Asymmetric protocols, also called public-key systems, use two different keys: a public key which may be disseminated widely. This key is used for encryption. And a private key which is known only to the owner. The private key is used for decryption. Asymmetric protocols are more secure and scalable. However, they usually require intensive computing, which makes them impractical on constrained devices as IoT devices. Despite this, some asymmetric schemes was proposed even for wireless sensor networks. Most of them implemented an Elliptic Curve Cryptography (ECC) [10], a CertificateLess Public Key Cryptography (CL-PKC) [11], an ID-Based Encryption (IBE) [12]...etc.

Most of the asymmetric schemes use a certificate to certify the ownership of a public key. A user must then check the certificate of another user before using its public key, which requires an intense amount of computation. For more efficiency, IBE was proposed in [13]. The main idea was to take as public key the user's identity (name, e-mail, IP address,... etc.). It is no longer necessary to certify public keys. However, the secret key must be computed by a key generator (KGC). Indeed, if a user can calculate his private key from his public key, he could also calculate those of the other users.

2.3. Hybrid approaches

Hybrid approaches combine symmetric and asymmetric techniques to take advantages of each and overcome its disadvantages like the work in [14]. A

hybrid scalable group key management approach for large dynamic multicast networks is proposed in [16], which tries to generate and distribute keys to the group members during leave or join of members by using key graph based Boolean minimization technique in order to improve scalability. Previous researches for hybrid key management [17, 18, 25] suggested using heterogeneous network with high end sensor nodes (HSNs) having high power for high computations of certificates verification and low end nodes, where HSNs are used to perform high power calculations such as certificate verification, exponentiation, elliptic curve scalar multiplications and additions and modular multiplications.

2.4. Discussion

Our literature review shows that none of the existing solutions meets all criteria. The more secure a protocol is, the less efficient it is, and conversely. Symmetric schemes usually require less computation time than the asymmetric ones, which makes them more suitable for IoT devices. On the one hand, the Pairwise Key approaches are secure but not scalable. On the other hand, the key group schemes are efficient but do not deal with collusion attacks. Since the asymmetric techniques are impractical on the constrained devices, the best possible solution is the complementary use of both Pairwise Key and Group Key methods. In Table 1, we present a classification and discussion of the recent related solutions.

3. System model

3.1. Things, initial knowledge, and the communication

The network consists of n communicating things denoted s_1, ..., s_n. Each node has an identity id_i, which is known only by itself and its neighbors. When computing bit complexities, we will assume that any node identity is encoded in $\log_2 n$ bits. When considering a node s_i, the integer i is called its index. Indexes are not known by the nodes. They are only a notation convenience used as a subscript to distinguish nodes and their local variables. The network group is controlled by a Group Controller GC, which is trusted and powerful.

Key category	Papers	Discussion
pairwise, symmetric post distribution	[19]	[+] efficient in static networks. [-] neither consider the group and multigroup communication.
pairwise, symmetric hybrid distribution	[20]	[+] efficient in dynamic networks. [-] lack scalabilty in limited-resource devices.
symmetric, group post distribution	[4, 9]	[+] efficient groups of dynamic networks. [-] lack scalabilty limited-resource devices.
symmetric group, hyb distribution	[21]	[+] efficient group and multi-group in dynamic networks. [-] lack scalabilty limited-resource devices.
post distribution symmetric key	[22]	[+] efficient and scalable in dynamic networks. [-] do not consider the device-to-device communication.
post distribution asymmetric pairwise	[24, 25]	[+] scalable in dynamic networks. [-] not suitable for the IoT constrained devices.

Table 1: Classification and discussion of some recent related solutions

3.2. Timing model

We assume that processing durations are equal to 0. This is justified by the following observations: (a) the duration of the local computations of a node is negligible with respect to message transfer delays, and (b) the processing duration of a message may be considered as a part of its transfer delay. Communication is synchronous in the sense that there is an upper bound D on message transfer delays (each message is received within the bound D), and this bound is known by all the nodes (global knowledge). From an algorithm design point of view, we consider that there is a global clock, denoted $CLOCK$, which is increased by 1, after each period of D physical time units. Each value of $CLOCK$ defines what is usually called a *time slot* or a *round*.

3.3. Communication operations

The nodes are provided with operations denoted broadcast(), send() and receive(). A node s_i invokes broadcast TAG(m) to send the message m, whose type is TAG, to all its neighbors. It is assumed that a node invokes broadcast() only at a beginning of a time slot. When a message TAG(m) arrives at a thing s_i, this node is immediately warned about it, which triggers the execution of operation receive() to obtain the message.

3.4. Definitions

Definition 1. *Let $u_1, u_2, ..., u_r$ be a set of r ordered positive integers that present the identities of nodes in a group G_x. And let $v_1, v_2, ..., v_t$ be a set of t ordered positive integers presenting the identities of other nodes in group G_y. We say G_x is greater (resp. smaller) than G_y and we write $G_x \succ G_y$ (resp. $G_x \prec G_y$) if*

- $\sum_{i=1}^{r} u_i > \sum_{i=1}^{t} v_i$ *(resp.* $\sum_{i=1}^{r} u_i < \sum_{i=1}^{t} v_i$ *) or*
- $u_r > v_t$ *(resp.* $u_r < v_t$ *), if* $\sum_{i=1}^{r} u_i = \sum_{i=1}^{t} v_i$

Definition 2. ***rekeying tool****. It is an information (may be a number) used as parameter to calculate a key. Nodes that use the same function with the same rekeying tool obtain the same key.*

To each node is associated two rekeying tools and to each group is associated two rekeying tools. It is important to note that each node and does not know its rekeying tools and rekeying tools of a group are not known by its nodes.

4. Proposed Protocol

Notation	Description
$CGK = (\mathcal{E}CGK, \mathcal{D}CGK)$	The current group key. Encrypt with $\mathcal{E}CGK$ and decrypt with $\mathcal{D}CGK$
K_{s_i}	A key shared between node s_i and GC
$K_{G_x^y}$	A key shared between nodes in the group G_x^y
$GenSecuPar()$	A function that generates a security parameter
$FDK()$	A function that generates keys using a security parameter
$H()$	A hash function

Table 2: Table of notations

The nodes of the group G are logically[1] partitioned as follows :

1. Create $\sqrt[3]{n}$ subgroups in G. Namely, the created subgroups are G_1, . . . , $G_{\sqrt[3]{n}}$. We call these groups *Level-2 groups*

2. Create in each subgroup G_j, $1 \leq j \leq \sqrt[3]{n}$, $\sqrt[3]{n}$ subgroups. Namely, for each G_j there are the subgroups $G_j^{(j-1)\sqrt[3]{n}+1}$, . . . , $G_j^{j\sqrt[3]{n}}$. We call these subgroups *Level-1 groups*

[1]Note that these grouping is logical and transparent to the application layer.

3. Put in each subgroup G_j^p, $1 \leq j \leq \sqrt[3]{n}$, $(j-1)\sqrt[3]{n}+1 \leq p \leq j\sqrt[3]{n}$, ρ nodes with $\rho \leq \sqrt[3]{n}$ nodes

4. For each node s_i, $1 \leq i \leq n$, create two rekeying tools $SRT_{>}^{i}, SRT_{<}^{i}$

5. For each subgroup G_j, $1 \leq j \leq \sqrt[3]{n}$, create two rekeying tools $GRT_{>}^{G_j}, GRT_{<}^{G_j}$

6. For each subgroup G_j^p, $1 \leq j \leq \sqrt[3]{n}$, $(j-1)\sqrt[3]{n}+1 \leq p \leq j\sqrt[3]{n}$, create two rekeying tools $G^2RT_{>}^{G_j^p}, G^2RT_{<}^{G_j^p}$

7. Each node s_i, $1 \leq i \leq n$, belonging to G_j^p, stores in its local memory :

   - in a set $S_{>}^{i}$ all $SRT_{>}^{f}$, with $id_i > id_f$ and $f \in G_j^p$, where $SRT_{>}^{f}$ is a rekeying tool of s_f
   - in a set $S_{<}^{i}$ all $SRT_{<}^{k}$, with $id_i < id_k$ and $k \in G_j^p$, where $SRT_{>}^{f}$ is a rekeying tool of s_k
   - in a set $S_{>}^{'i}$ all $GRT_{>}^{G_j^m}$, with $G_j^p \succ G_j^m$ and $G_j^{G_j^m} \in G_j$, where $GRT_{>}^{G_j^m}$ is a rekeying tool of group G_i^m
   - in a set $S_{<}^{'i}$ all $GRT_{<}^{G_j^p}$, with $G_j^p \prec G_j^l$ and $G_l^k \in G_j$, where $GRT_{>}^{G_j^p}$ is a rekeying tool of group G_i^l
   - in a set $S_{>}^{''i}$ all $G^2RT_{<}^{G_w}$, with $G_j \prec G_w$ and $G_w \in G$, where $G^2RT_{<}^{G_w}$ is a rekeying tool of group G_w
   - in a set $S_{<}^{''i}$ all $G^2RT_{>}^{G_x}$, with $G_j \succ G_x$ and $G_x \in G$, where $G^2RT_{>}^{G_x}$ is a rekeying tool of group G_x

Let RT_i be the set of all rekeying tools of s_i (e.g. $RT_i = S_{<}^{i} \cup S_{<}^{'i} \cup S_{<}^{''i} \cup S_{>}^{i} \cup S_{>}^{'i} \cup S_{>}^{''i}$).

The figure 1 presents a grouping example of a group of 26 nodes.

To present our solution, we use the notations shown in Table 2.

4.1. Rekeying upon joining

In case of joining, rekeying procedures must ensure that the new node to be added into the group G is unable to decrypt all information created prior its addition. To achieve this goal we must change the group key at the joining event.

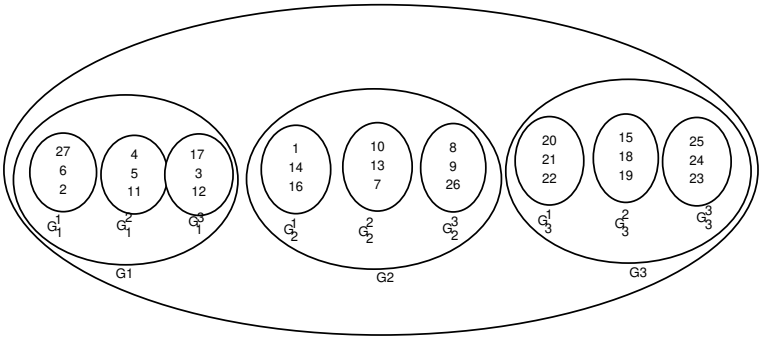

Figure 1: Example of grouping with 26 nodes

When a new node joins the group, the protocol in Algorithm 1 is executed. The input of this algorithm is the joining event and the output is the new group key calculated by each node in the new group.

Let *NwNd* be the ID of this new node. This node sends JOINING_REQUEST(*NwNd*) message destined to GC. Upon the reception of this message (line 01) from *NwNd*, GC runs the instructions 1 to 14.

In instruction 02, GC computes $MinG$, the level-1 group that has the smallest size (if there are several level-1 groups that have the smallest size, i.e $\left|G_{i_1}^{p_1}\right| = ... = \left|G_{i_m}^{p_m}\right|$ then GC chooses $G_{i_x}^{p_x}$ with $i_x < i_y, \forall y \in [1, m]$ to break the tie.) After it puts in $MinG$ the ID of the new node. Then, GC generates a new security parameter λ (line 3). Let G_x be the level-2 group containing $MinG$ and let G_x^y be this group ($G_x^y = MinG$). In line 4, GC calculates a new group key $CGK' = (\mathcal{E}CGK', \mathcal{D}CGK')$ and a new level-1 group key $K'_{G_x^y}$ using the FDK function and λ. To rekey nodes in $MinG$, GC constructs a message called REKEYING_MING(). This message contains the ID of the new node, the security parameter λ and the new group key encrypted by means of $\mathcal{E}K_{MinG}$. This message will be broadcasted to nodes in $MinG$ (line 05). Further, to rekey nodes in $G - MinG$, GC constructs a message called REKEYING_GEXMING() containing the new group key CGK' encrypted by means of $\mathcal{E}CGK$. This message will be broadcasted to nodes in G except nodes in $MinG$ (line 06). After, GC puts in a set called $S_{>}^{NwNd}$, the set of rekeying tools of nodes in $MinG$ that

Instructions for GC :

(01) **when** JOINING_REQUEST($NwNd$) **is received do**
(02) $MinG \leftarrow \min(G_i^p : 1 \leq i \leq \sqrt[3]{n},\ (i-1)\sqrt[3]{n}+1 \leq p \leq i\sqrt[3]{n})$;
(03) $MinG \leftarrow MinG \cup NwNd$; $\lambda \leftarrow GenSecuPar()$;
(04) $CGK' \leftarrow FDK(\lambda \| CGK)$; $K'_{G_x^y} \leftarrow FDK(\lambda \| K_{G_x^y})$;
(05) broadcast REKEYING_MING($MinG - \{NwNd\}, \{NwNd, \lambda, CGK'\}_{K_{MinG}}$);
(06) broadcast REKEYING_GEXMING($G - MinG, \{NwNd, \lambda, CGK'\}_{CGK}$);
(07) $S_{>}^{NwNd} \leftarrow$ all $SRT_{>}^{f} : NwNd > id_f, f \in MinG$;
(08) $S_{<}^{NwNd} \leftarrow$ all $SRT_{<}^{f} : NwNd < id_f, f \in MinG$;
(09) $S_{>}^{'NwNd} \leftarrow$ all $GRT_{>}^{f} : MinG \succ G_i^f, MinG \in G_i \wedge G_i^f \in G_i$;
(10) $S_{<}^{'NwNd} \leftarrow$ all $GRT_{<}^{f} : MinG \prec G_i^f, MinG \in G_i \wedge G_i^f \in G_i$;
(11) $S_{>}^{''NwNd} \leftarrow$ all $G^2RT_{>}^{f} : MinG \succ G_i^f, MinG \in G_i \wedge G_i^f \in G_i$;
(12) $S_{<}^{''NwNd} \leftarrow$ all $G^2RT_{<}^{f} : MinG \prec G_i^f, MinG \in G_i \wedge G_i^f \in G_i$;
(13) $m \leftarrow \left\{ CGK', K'_{G_x^y}, \left\{ S_>, S_<, S_>^{'NwNd}, S_<^{'NwNd}, S_<^{''NwNd}, S_>^{''NwNd} \right\}_{CGK'} \right\}_{K_{NwNd}}$
(14) send REKEYING_TOOLS ($NwNd, m$);

Instructions for node s_i :

(15) **when** REKEYING_MING($dest, m = \{NwNd, \lambda, CGK'\}_{K_{MinG}}$) **is received do**
(16) **if** ($dest \neq id_i$) **then** discard the message ; **end if**
(17) $dm \leftarrow \mathcal{D}K_{MinG}(m)$;
(18) $G_x^y \leftarrow G_x^y \cup \{dm.NwNd\}$;
(19) $K'_{G_x^y} \leftarrow H(dm.\lambda || K_{G_x^y})$; $CGK' \leftarrow dm.CGK'$;
(20) **if** ($id_i > dm.NwNd$)
(21) **then** $SRT_{>}^{dm.NwNd} \leftarrow H(dm.\lambda)$
(22) **else** $SRT_{<}^{dm.NwNd} \leftarrow H(dm.\lambda)$
(23) **end if**
(24) **when** REKEYING_GEXMING($dest, m = \{NwNd, \lambda, CGK'\}_{CGK}$) **is received do**
(25) **if** ($dest \neq id_i$) **then** discard the message **end if**;
(26) $dm \leftarrow \mathcal{D}CGK(m)$;
(27) $CGK' \leftarrow dm.CGK'$;
(28) $G \leftarrow G \cup \{dm.NwNd\}$
(29) **when** REKEYING_TOOLS($NwNd$,m) **received do**
(30) **if** ($dest \neq NwNd$) **then** discard the message **end if**;
(31) $dm \leftarrow (\mathcal{D}CGK', \mathcal{D}K_{NwNd})(m)$;
(32) $CGK' \leftarrow m.CGK'$; $K'_{G_x^y} \leftarrow m.K'_{G_x^y}$
(33) $S_{>}^{NwNd} \leftarrow dm.S_{>}^{NwNd}$; $S_{<}^{NwNd} \leftarrow dm.S_{<}^{NwNd}$;
(34) $S_{>}^{'NwNd} \leftarrow dm.S_{>}^{'NwNd}$; $S_{>}^{'NwNd} \leftarrow dm.S_{>}^{'NwNd}$;
(35) $S_{>}^{''NwNd} \leftarrow dm.S_{>}^{''NwNd}$; $S_{>}^{''NwNd} \leftarrow dm.S_{>}^{''NwNd}$;

Algorithm 1: Protocol executed when a node joins the group

have IDs superior than $NwNd$ (line 07) and in a set called $S_{<}^{NwNd}$, the rekeying tools of nodes in $MinG$ that have IDs inferior than $NwNd$ (line 08). It puts in $S_{>}^{'NwNd}$ the set of rekeying tools of level-1 group that are inferior (with respect to definition 1) than $MinG$ (line 9) and in the set $S_{<}^{'NwNd}$ the rekeying tools of level-1 groups that are superior than $MinG$ (line 10). It puts in $S_{>}^{''NwNd}$ the set of rekeying tools of level-2 groups that are inferior than G_x^y (line 11) and

in the set $S_{<}^{''NwNd}$ the rekeying tools of level 2 groups that are superior than G_x^y (line 12). In line 14, GC sends to $NwNd$ the message REKEYING_TOOLS () containing these rekeying tools, the level-1 group key $K'_{G_x^y}$ and the new group key. This message is encrypted by means of $\mathcal{E}K_{NwNd}$ and $\mathcal{E}CGK'$ (line 13).

Upon the reception of the message REKEYING_MING(), each node s_i in $MinG$ runs the instructions 16 to 23. Firstly, s_i decrypts the message using the level-1 group key $\mathcal{D}K_{MinG}$ (line 17) and adds the ID of the new node into G_x^y (line 18). It calculates the new level-1 group key $K'_{G_x^y}$ using the hash function and the security parameter λ sent by GC. After, it gets and installs the new group key CGK' as the current group key (line 19). Then, s_i calculates the rekeying tool of $NwNd$ using the hash function and λ (lines 20-23). If the ID of $NwNd$ is superior than id_i, s_i records in its local memory the rekeying tool $SRT_{>}^{id_{NwNd}}$ else its records $SRT_{<}^{id_{NwNd}}$.

Upon the reception of the message REKEYING_GEXMING(), each node s_i in $G - MinG$ runs the instructions 25 to 28. Node s_i decrypts the message using group key $\mathcal{D}CK$ (line 26) and installs CGK' as the current group key (line 27). After, s_i adds the new node into G (line 28).

At the reception of the message REKEYING_TOOLS(), the new node decrypts the message using the keys $\mathcal{D}CGK$ and $\mathcal{D}K_{NwNd}$. It gets the new level-1 group key $K'_{G_x^y}$, the new key group CGK' (line 32) and stores in its local memory the rekeying tools $S_{>}^{NwNd}, S_{<}^{NwNd}, S_{>}^{'NwNd}, S_{<}^{'NwNd}, S_{>}^{''NwNd}$ and $S_{<}^{''NwNd}$ (lines 31-35). These rekeying tools will be used when a node leaves (next section).

4.2. Rekeying upon leaving

In case of leaving, rekeying procedures must ensure that the evicted node is unable to decrypt all pieces of information created after its leaving. In case of leaving, Algorithm 2 is executed to change the group key.The input of this algorithm is the leaving event and the output is the new group key.

When a node of $LvNd$ leaves the group, the current group key CGK has to be changed in order to prevent $LvNd$ from decrypting new ciphertexts. Let G_x^y and G_x be the level-1 group and the level-2 group of $LvNd$, respectively.

(01) **when** LEAVING_EVENT($NwNd$) **is detected do**
(02) $G \leftarrow G - LvNd$; $\lambda \leftarrow GenSecuPar()$;
(03) $CGK' \leftarrow FDK(\lambda \| CGK)$; $K'_{G_x^y} \leftarrow FDK(\lambda \| K_{G_x^y})$;
(04) $A = (\mathcal{E}A, \mathcal{D}A) = FDK(SRT_{>}^{LvNd})$; $B = (\mathcal{E}B, \mathcal{D}B) = FDK(SRT_{<}^{LvNd})$;
(05) $C = (\mathcal{E}C, \mathcal{D}C) = FDK(GRT_{>}^{G_x^y})$; $D = (\mathcal{E}D, \mathcal{D}D) = FDK(GRT_{<}^{G_x^y})$;
(06) $E = (\mathcal{E}E, \mathcal{D}E) = FDK(G_2RT >^{G_x})$; $F = (\mathcal{E}F, \mathcal{D}F) = FDK(GRT_2 <^{G_x})$;
(07) broadcast REKEYING_$G_x^y(G_x^y, \{LvNd, \lambda, CGK'\}_A, \{LvNd, \lambda, CGK'\}_B)$;
(08) broadcast$rekeying_G_x Ex G_x^y(G_x - G_x^y, \{LvNd, \lambda, CGK'\}_C, \{LvNd, \lambda, CGK'\}_D)$;
(09) broadcast REKEYING_$GExG_x$ $(G - G_x, \{LvNd, \lambda, CGK'\}_E, \{LvNd, \lambda, CGK'\}_F)$;

Instructions for node s_i :
(10) **when** REKEYING_$G_x^y(dest, m1, m2)$ **is received do**
(11) **if** $(dest \neq id_i)$ **then** discard the message **end if**;
(12) **if** $(id_i > LvNd)$ **then** $A \leftarrow (\mathcal{E}A, \mathcal{D}A) = FDK(SRT_{>}^{LvNd})$;
(13) $dm \leftarrow \mathcal{D}A(m1)$;
(14) **else** $B \leftarrow (\mathcal{E}B, \mathcal{D}B) = FDK(SRT_{<}^{LvNd})$;
(15) $dm \leftarrow \mathcal{D}B(m2)$;
(16) **end if**;
(17) $CGK' \leftarrow dm.CGK'$; $K'_{G_x^y} \leftarrow H(dm.\lambda || K_{G_x^y})$;
(18) $G_x(i) \leftarrow G_x(i) - \{dm.LvNd\}$; **forall** $k \in RT_i, k \leftarrow H(dm.\lambda || k)$;
(19) **when** REKEYING_$G_x Ex G_x^y(dest, m1, m2)$ **is received do**
(20) **if** $(dest \neq id_i)$ **then** discard the message **end if**;
(21) **if** $(G_x^y(i) \succ G_x^y)$ **then** $C \leftarrow (\mathcal{E}C, \mathcal{D}C) = FDK(GRT_{<}^{G_x^y})$;
(22) $dm \leftarrow \mathcal{D}C(m1)$;
(23) **else** $D \leftarrow (\mathcal{E}C, \mathcal{D}D) = FDK(GRT_{>}^{G_x^y})$;
(24) $dm \leftarrow \mathcal{D}D(m2)$;
(25) **end if**;
(26) $CGK' \leftarrow dm.CGK'$;
(27) $G_x^y(i) \leftarrow G_x^y(i) - \{dm.LvNd\}$; **forall** $k \in RT_i, k \leftarrow H(dm.y || k)$;
(28) **when** REKEYING_$GExG_x(dest, m1, m2)$ **is received do**
(29) **if** $(dest \neq id_i)$ **then** discard the message **end if**;
(30) **if** $(G_x(i) \succ G_x)$ **then** $E \leftarrow (\mathcal{E}E, \mathcal{D}E) = FDK(G_2RT_{>}^{G_x})$;
(31) $dm \leftarrow \mathcal{D}E(m1)$;
(32) **else** $F \leftarrow (\mathcal{E}F, \mathcal{D}F) = FDK(G_2RT_{<}^{G_x})$;
(33) $dm \leftarrow \mathcal{D}F(m2)$;
(34) **end if**;
(35) $CGK' \leftarrow dm.CGK'$; $G_x(i) \leftarrow G_x(i) - \{dm.LvNd\}$; **forall** $k \in RT_i, k \leftarrow H(dm.\lambda || k)$;

Algorithm 2: Protocol executed when a node leaves the group

Let $G_x^y(i)$ and $G_x(i)$ be the level-1 group and the level-2 group of node s_i that receives the message in Algorithm 2.

Upon the detection of leaving event, GC runs the instructions 2 to 9. GC deletes the ID of the leaving node from the group G and generates a new security parameter λ (line 2). Then, it calculates a new group key $CGK' = (\mathcal{E}CGK', \mathcal{D}CGK')$ and a new level-1 group key, using the FDK function and λ (line 03). In addition, it calculates the keys $A = (\mathcal{E}A, \mathcal{D}A) = FDK(SRT_{>}^{LvNd})$,

$B = (\mathcal{E}B, \mathcal{D}B) = FDK(SRT_{<}^{LvNd})$, $C = (\mathcal{E}C, \mathcal{D}C) = FDK(GRT_{>}^{G_x^y})$, $D = FDK(GRT_{<}^{G_x^y})$, $E = FDK(G^2RT_{>}^{G_x})$, $F = FDK(GRT_{<}^{2\,G_x})$ (lines 04-06).

GC constructs the message REKEYING_$G_x^y()$, containing the new key of the group, the ID of the leaving node and the security parameter λ encrypted by means of keys A and B. Then, GC broadcasts this message to nodes in G_x^y. This message will be used to rekey nodes in G_x^y (line 07).

After it constructs the message REKEYING_$G_xExG_x^y()$, containing the new key of the group, the ID of the leaving node and the security parameter λ encrypted by means of keys C and D. GC broadcasts this message to nodes in $G_x - G_x^y$. Using this message nodes in $G_x - G_x^y$ will be able to get the new group key (line 08). Finally, GC constructs the message REKEYING_$GExG_x()$. This message contains the new group key, the ID of the leaving node and the security parameter λ encrypted by means of E and F. GC broadcasts this message to nodes in $G - G_x$. This message will be used to rekey nodes in $G - G_x$ (line 09).

Upon the reception of the message REKEYING_$G_x^y()$, each node s_i in G_x^y runs the instructions 11 to 18. If id_i is superior than $LvNd$ then s_i can compute the key A locally using $SRT_{>}^{LvNd}$ and the FDK function. If id_i is inferior than $LvNd$ then s_i can compute the key B locally using $SRT_{<}^{LvNd}$ and the FDK function. It decrypts the message using $\mathcal{D}A$ or $\mathcal{D}B$ and installs CGK' as the current group key. Further, it computes a new level-1 group key $K'_{G_y^x}$ using the hash function and the security parameter λ sent from GC (lines 12-17). In line 18, s_i deletes $LvNd$ from the group. Finally, s_i changes all its rekeying tools using the hash function and λ.

Upon the reception of the message REKEYING_$G_xExG_x^y()$ sent by GC, each node s_i in $G_x - G_x^y$ runs the instructions 20 to 27. If $G_x^y(i)$ is superior (with respect to definition 1) than G_x^y, s_i can compute the key C locally using $GRT_{>}^{G_x^y}$ and FDK function. Then, it decrypts the message using $\mathcal{D}C$. If $G_x^y(i)$ is inferior than G_x^y, s_i can compute the key D locally using $GRT_{<}^{G_x^y}$ and FDK. It decrypts the message using $\mathcal{D}D$ (lines 21-25). In line 26, s_i installs the new key CGK' as the current group key. In line 27, it deletes $LvNd$ from the group and changes

all its rekeying tools using the hash function and λ.

At the reception of REKEYING_$GExG_x()$, each node s_i in $G - G_x$ runs the instructions 30 to 35. If $G_x(i)$ is superior than G_x then s_i can compute the key E using $G^2RT_>^{G_x}$ and decrypt the message using that key (line31). If $G_x(i)$ is inferior (with respect to definition 1) than G_x, then s_i can compute the key F locally using $GRT_2 <^{G_x}$ (with respect to definition 1). s_i decrypts the received message using the computed key (line 33). Then, it deletes $LvNd$ from the group G and installs CGK' as the current group key. Finally, s_i changes all its rekeying tools using the hash function and λ (lines 35).

4.3. Preventing from collusion attacks

A collusion attack occurs when multiple compromised nodes from G cooperate to regain access to the secret group key. When a collusion attack is detected Algorithm 3 is executed.

Let $\mathscr{G}$ be the set of compromised level-2 groups (a group is compromised if it contains at least one compromised node). And let $Q = G - \mathscr{G}$ be the set of non compromised level-2 groups in G. Let $min\mathscr{G}$ and $max\mathscr{G}$ be the greater level-2 group and the smaller level-2 group in $\mathscr{G}$, respectively (w.r.t. definition 1). Let us remark that the rekeying tools $G^2RT_<^{min\mathscr{G}}$ and $G^2RT_>^{max\mathscr{G}}$ are not stored by compromised nodes in $\mathscr{G}$. Let $\mathscr{Q}$ be the set of level-2 groups that stores $G^2RT_>^{min\mathscr{G}}$ or $G^2RT_>^{max\mathscr{G}}$. All rekeying tools in the $Q - \mathscr{Q}$ are compromised.

Let $\mathscr{S} = \mathscr{S}_1, \mathscr{S}_2, ..., \mathscr{S}_t$ be the set of t level-1 compromised groups in the groups of level 2: $G_1, G_2, ..., G_t$ respectively. And let $P_t = G_t - \mathscr{S}_t$ be the set of non compromised groups in level-2 group G_t. Let $min\mathscr{S}_t$ and $max\mathscr{S}_t$ be the greater level-1 group and the smaller level-1 group in $\mathscr{S}_t$, respectively. Let us remark that the rekeying tools $GRT_<^{min\mathscr{S}_t}$ and $GRT_>^{max\mathscr{S}_t}$ are not stored by compromised nodes in $\mathscr{S}_t$. Let $\mathscr{P}_t$ be the set of groups of nodes that stores $GRT_>^{min\mathscr{G}}$ or $GRT_>^{max\mathscr{G}}$ in $\mathscr{S}_t$. All rekeying tools hold by nodes in $P_t - \mathscr{P}_t$ are compromised and therefore can not by used to rekey nodes in non compromised level-1 groups of $\mathscr{S}_t$. Let $\mathscr{C} = \mathscr{C}_1, \mathscr{C}_2, .., \mathscr{C}_r$ be the list of t sets of compromised nodes in the same group of level 1. And let $G_{x\mathscr{C}_r}^y$ be the level-1 group of nodes

Instructions for GC :

(01) **when** COLLUSION_EVENT() **is detected do**
(02) $G \leftarrow G - \mathscr{C}$; $\lambda \leftarrow GenSecuPar()$; $CGK' \leftarrow FDK(\lambda \| CGK)$;
(03) **forall** ($\mathscr{C}_r$ **in** $\mathscr{C}$) **do**
(04) $A_r = FDK(SRT_{<}^{min\mathscr{C}_r})$; $B_r = FDK(SRT_{>}^{max\mathscr{C}_r})$;
(05) broadcast REKEYING_COLL_NODES_1($\mathscr{O}_r$, $\{\mathscr{C}, \lambda, CGK'\}_{A_r}$, $\{\mathscr{C}, \lambda, CGK'\}_{B_r}$);
(06) **endforall**;
(07) **forall** s_i **in** $O_r - \mathscr{O}_r$ **do** broadcast REKEYING_COLL_NODES_2(id_i, $\{\mathscr{C}, CGK'\}_{k_i}$); **endforall**;
(08) **forall** ($\mathscr{S}_t$ **in** $\mathscr{S}$) **do**
(09) $C_t = FDK(GRT_{<}^{min\mathscr{S}_t})$; $D_t = FDK(GRT_{>}^{max\mathscr{S}_t})$;
(10) broadcast REKEYING_COLL_LEVEL1GROUPSS_1 ($\mathscr{P}_t$, $\{\mathscr{C}, CGK'\}_{C_t}$, $\{\mathscr{C}, CGK'\}_{D_t}$);
(11) **endforall**;
(12) **forall** s_i **in** $P_t - \mathscr{P}_t$ **do** broadcast REKEYING_COLL_L1G_2(id_i, $\{\mathscr{C}, CGK'\}_{k_i}$); **endforall;**
(13) $E = FDK(G^2RT_{>}^{min\mathscr{G}})$; $F = FDK(G^2RT_{>}^{max\mathscr{G}})$;
(14) broadcast REKEYING_COLL_L1G_1($\mathscr{G}$, $\{\mathscr{C}, CGK'\}_E$, $\{\mathscr{C}, CGK'\}_F$) ;
(15) **forall** s_i **in** $O - \mathscr{O}$ **do** broadcast REKEYING_COLL_L2G_2 (id_i, $\{\mathscr{C}, CGK'\}_{k_i}$); **endforall;**

Instructions for node s_i :

(16) **when** REKEYING_COLL_NODES_1($dest$, $m1 = \{\mathscr{C}, \lambda, CGK'\}_{A_r}$, $m2 = \{\mathscr{C}, \lambda, CGK'\}_{B_r}$) **is received do**
(17) **if** ($dest \neq id_i$) **then** discard the message **end if**;
(18) **if** $id_i < id_{min\mathscr{C}_r}$
(19) **then** $A \leftarrow FDK(SRT_{<}^{min\mathscr{C}})$;
(20) $dm \leftarrow \mathcal{D}A(m1)$;
(21) **else** $B \leftarrow FDK(SRT_{<}^{max\mathscr{C}})$;
(22) $dm \leftarrow \mathcal{D}B(m2)$;
(23) **end if**;
(24) $CGK \leftarrow dm.CGK'$; $G \leftarrow G - \{dm.\mathscr{C}\}$; $K'_{G_x^y}(i) \leftarrow H(dm.\lambda || K_{G_x^y})$;
(25) **when** REKEYING_COLL_NODES_2($dest$, $m = \{\mathscr{C}, CGK'\}_{K_i}$) **is received do**
(26) **if** ($dest \neq id_i$) **then** discard the message **end if**;
(27) $dm \leftarrow \mathcal{D}K_i(m)$; $K'_{G_x^y}(i) \leftarrow H(dm.\lambda || K_{G_x^y})$;
(28) $CGK \leftarrow dm.CGK'$; $G \leftarrow G - \{dm.\mathscr{C}\}$;
(29) **when** REKEYING_COLL_L1G_1($dest$, $m1 = \{\mathscr{C}, CGK'\}_{C_t}$, $m2 = \{\mathscr{C}, CGK'\}_{D_t}$) **is received do**
(30) **if** ($dest \neq id_i$ **then** discard the message **end if**;
(31) **if** $id_v < id_{min\mathscr{S}_t}$
(32) **then** $C \leftarrow FDK(GRT_{<}^{min\mathscr{S}_t})$;
(33) $dm \leftarrow \mathcal{D}C(m1)$;
(34) **else** $D \leftarrow FDK(GRT_{<}^{max\mathscr{S}_t})$;
(35) $dm \leftarrow \mathcal{D}D(m2)$;
(36) **end if**;
(37) $CGK \leftarrow dm.CGK'$; $G \leftarrow G - \{dm.\mathscr{C}\}$;
(38) **when** REKEYING_COLL_L1G_2($dest$, $m = \{\mathscr{C}, CGK'\}_{K_i}$) **is received do**
(39) **if** ($dest \neq id_i$) **then** discard the message **end if**;
(40) $dm \leftarrow \mathcal{D}K_i(m)$;
(41) $CGK \leftarrow dm.CGK'$; $G \leftarrow G - \{dm.\mathscr{C}\}$;
(42) **when** REKEYING_COLL_L1G_1($dest$, $m1 = \{\mathscr{C}, CGK'\}_E$, $m2 = \{\mathscr{C}, CGK'\}_F$) **is received do**
(43) **if** ($dest \neq id_i$) **then** discard the message **end if**;
(44) **if** $id_v < id_{min\mathscr{S}}$
(45) **then** $E \leftarrow FDK(G^2RT_{>}^{min\mathscr{G}})$;
(46) $dm \leftarrow \mathcal{D}E(m1)$;
(47) **else** $F \leftarrow FDK(G^2RT_{<}^{max\mathscr{G}})$;
(48) $dm \leftarrow \mathcal{D}F(m2)$;
(49) **end if**;
(50) $CGK \leftarrow dm.CGK'$; $G \leftarrow G - \{dm.\mathscr{C}\}$;
(51) **when** REKEYING_COLL_L2G_2($dest$, $m = \{\mathscr{C}, CGK'\}_{k_s}$) **is received do**
(52) **if** ($dest \neq id_i$) **then** discard the message **end if**;
(53) $dm \leftarrow \mathcal{D}K_i(m)$; $CGK \leftarrow dm.CGK'$; $G \leftarrow G - \{dm.\mathscr{C}\}$;

Algorithm 3: Protocol executed when collusion attack is detected

in $\mathscr{C}_r$. And let $O_r = G^y_{x\mathscr{C}_r} - \mathscr{C}_r$ be the set of non compromised nodes in $G^y_{x\mathscr{C}_r}$.

Let $min\mathscr{C}_r$ and $max\mathscr{C}_r$ be the nodes with the lowest ID and the highest ID in $\mathscr{C}_r$, respectively. Let us remark that $SRT^{min\mathscr{C}_r}_<$ and $SRT^{max\mathscr{C}_r}_>$ are not stored by compromised nodes. Let $\mathscr{O}_r$ be the set of nodes that stores $SRT^{min\mathscr{C}_r}_<$ or $SRT^{max\mathscr{C}_r}_>$. All rekeying tools hold by nodes in $O_r - \mathscr{O}_r$ are compromised and therefore can not be used by GC to rekey the nodes in the group O_r.

When a collusion attack is detected GC does the following actions:

1. GC deletes the colluding nodes from the group G and generates a new parameter λ, and calculates a new group key (line 01).

2. GC uses $SRT^{min\mathscr{C}_r}_<$ and $SRT^{max\mathscr{C}_r}_>$ to rekey nodes in compromised level groups: For every $\mathscr{C}_r \in \mathscr{C}$, using these rekeying tools and FDK function GC calculates the keys $A_r = FDK(SRT^{min\mathscr{C}_r}_<)$ and $B_r = FDK(SRT^{max\mathscr{C}_r}_>)$. Then, GC constructs a message $rekeying\_Coll\_nodes\_1()$ containing the new group key, the new λ and the compromised nodes. This message is encrypted using the keys A_r and B_r. Next, GC broadcasts it to nodes in $\mathscr{O}_r$ (lines 3-6). And for every node i in $O_r - \mathscr{O}_r$ GC constructs a message $rekeying\_Coll\_nodes\_2()$ encrypted using K_i and sends it to s_i (line 7).

3. GC uses $GRT^{min\mathscr{S}_t}_<$ and $GRT^{max\mathscr{S}_t}_>$ to rekey non compromised level-1 groups in compromised level-2 groups : GC calculates the keys $C_t = FDK(GRT^{min\mathscr{S}_t}_<)$ and $D_t = FDK(GRT^{max\mathscr{S}_t}_>)$ and constructs a message $rekeying\_Coll\_L1G\_1()$ encrypted by means of C_t and D_t and broadcasts it to nodes in $\mathscr{P}$ (lines 8-11). For every node i in $P_t - \mathscr{P}_t$, GC constructs $rekeying\_Coll\_L1G\_2()$ encrypted with K_i and sends its to s_i (line 12).

4. GC uses $G^2RT^{min\mathscr{G}}_<$ and $G^2RT^{max\mathscr{G}}_>$ to rekey the group $G - \mathscr{C}$ (i.e. to rekey non compromised groups of level 2): it calculates the keys $E = FDK(G^2RT^{min\mathscr{G}}_<)$ and $F = FDK(G^2RT^{max\mathscr{G}}_>)$. GC constructs the message $rekeying\_Coll\_L1G\_1()$ and encrypts it with E and F and broadcasts it to $\mathscr{G}$ (lines 13-14). For every node s_i in $Q - \mathscr{Q}$, GC constructs a message $rekeying\_Coll\_L2G\_2()$ encrypts it with K_i and sends it to s_i (lines 15).

Upon the reception of $rekeying\_Coll\_nodes\_1()$, each node s_i in $\mathscr{O}_r$ does the following actions :

- calculates the key A_r if $id_i < id_{min\mathscr{C}_r}$ ($A_r = FDK(SRT_{<}^{min\mathscr{C}_r})$), or calculate the key B_r if $id_i > id_{max\mathscr{C}_r}$ ($Br = FDK(SRT_{>}^{max\mathscr{C}_r})$) (lines 17-23).

- decrypts the message using the computed key, gets the new group key CGK', deletes the compromised nodes from the group and calculates its new level-1 group key using λ (line 24).

Upon the reception of $rekeying\_Coll\_nodes\_2()$, each node i in $O_r - \mathscr{O}_r$ decrypts the message using K_i, calculates its new level-1 group key using λ (line 26) and gets the new key CGK' (line 28).

Upon the reception of $rekeying\_Coll\_L1G\_1()$, each node s_i in $\mathscr{P}$ does the following actions:

- calculate the key C if $id_v < id_{min\mathscr{C}}$ ($C = FDK(SRT_{<}^{min\mathscr{C}})$), or calculate the key D if $id_v > id_{max\mathscr{C}}$ ($D = FDK(SRT_{<}^{max\mathscr{C}})$) lines (30-36).

- decrypt the message using the computed key, gets the new key CGK' and deletes the compromised nodes from the group (line 37).

Upon the reception of $rekeying\_Coll\_L1G\_2()$, each node i in $P - \mathscr{P}$ decrypts the message using K_v and gets the new key CGK'.

Upon the reception of $rekeying\_Coll\_L1G\_1()$, each node s_i in $\mathscr{P}$ calculates the key E if $id_v < id_{min\mathscr{C}}$ ($E = FDK(SRT_{<}^{min\mathscr{C}})$), or calculate the key F if $id_i > id_{max\mathscr{C}}$ ($F = FDK(SRT_{<}^{max\mathscr{C}})$) (lines 44-49). Then, it decrypts the message, gets the new key CGK' and deletes the compromised nodes from G (line 50). Finally, at the reception of $rekeying\_Coll\_L2G\_2()$, each node s_i in $Q - \mathscr{Q}$ gets the new key CGK' and deletes the compromised nodes (line 53).

4.4. Analysis and cost of the algorithm

Lemma 1. *In case of joining backward secrecy is guaranteed by Algorithm 1 and all nodes in the new group G agree on the same new group key.*

Proof

Let us suppose a new node $NwNd$ wants to join G at time t. It is obvious that $NwNd$ cannot decipher the messages sent from nodes in G before the time t. Because it does not have the group key CGK. At time t, $NwNd$ sends the request message encrypted by means of the key K_{NwNwd} to GC. Without loss of generality, let us assume that $NwNd$ will be added into the level-1 group G_x^y. Following this request, GC generates a new group key for G, and a new level-1 group key for G_x^y using a new security parameter λ. GC broadcasts theses keys and λ using the current keys CGK and $K_{G_x^y}$. Therefore, only nodes in G can decipher this messages and get the new keys. GC broadcasts to K_{NwNwd} the new keys and the rekeying tools that allow it to compute a new key when a node leaves the network. Only GC and $NwNd$ can decipher the message because the key K_{NwNwd} is shared only between GC and $NwNd$.

$\square_{Lemma\ 1}$

Lemma 2. *In case of leaving forward secrecy is guaranteed by Algorithm 2 and all nodes in the new group G agree on the same new group key.*

Proof

Let us suppose a new node has $LvNd$ as ID is quitting the group at time t. Let us assume that $LvNd$ was in level-1 group G_x^y. The objective is to proof that $LvNd$ cannot decipher messages sent by nodes in G after the time t. And at $t+1$ all nodes in the new group $(G - \{LvNd\})$ have the same new group key. Upon the leaving event detection, GC generates a new key using a new security parameter λ (line 2). It also generates the keys A, B, C, D, E, F using the rekeying tools $SRT_{>}^{LvNd}$, $SRT_{>}^{LvNd}$, $GRT_{>}^{G_x^y}$,$GRT_{<}^{G_x^y}$, $G^2RT_{>}^{G_x^y}$ and $G^2RT_{<}^{G_x^y}$ respectively. As $NvNd$ does not know all these rekeying tools, it cannot calculate the keys A, B, C, D, E and F. Therefore it can not decrypt any message containing the new group key and the new λ. All nodes in $G - NvNd$ can decrypt one of these messages. Indeed, each node i in G_x^y can decrypt the message using key A if $id_i > LvNd$ or key B if $id_i < LvNd$. Each node i in G_x can decrypt th messages using C if $id_i > LvNd$ or D if $id_i > LvNd$. And all

nodes in $G_x - G_x$ decrypt the messages using E or F. Therefore all nodes get the new group key except $NvNd$. These completes the proof.

$\square_{Lemma\ 2}$

Lemma 3. *In case of collusion detection, the colluding nodes are evicted and nodes in the new group G agree on same new group key.*

Proof

When a collusion event is detected GC deletes all colluding nodes from the group G. GC generates a new security parameter and a new group key. Firstly, let us observe that in each group $\mathscr{C}_r$ there are the nodes $min\mathscr{C}_r$ and $max\mathscr{C}_r$. Let us recall that, to each node s_i in $\mathscr{C}_r$ is associated two rekeying tools $SRT^{s_i}_<$ and $SRT^{s_i}_>$ (unknown for s_i). Node s_j in $\mathscr{C}_r$ stores in its local memory the rekeying tool $SRT^{s_i}_<$ if $id_j < id_i$ or $SRT^{s_i}_<$ if $id_j > id_i$. Therefore, $SRT^{min\mathscr{C}_r}_<$ of node $min\mathscr{C}_r$ is not stored by any node in $\mathscr{C}_r$. Similarly, $SRT^{max\mathscr{C}_r}_>$ is not stored by any node $\mathscr{C}_r$. Let us observe that each node z in $G^y_{x\mathscr{C}_r} - \mathscr{C}_r$ with $min\mathscr{C}_r < id_z < max\mathscr{C}_r$ does not store in its local memory $min\mathscr{C}_r$ nor $max\mathscr{C}_r$. Each node x in $G^y_{x\mathscr{C}_r} - \mathscr{C}_r$ with $id_x < min\mathscr{C}_x$ or $id_x > id_z < max\mathscr{C}_r$ stores in its local memory $min\mathscr{C}_r$ or $max\mathscr{C}_r$. Therefore, each node x can decrypt the message in line 07 using A_r or B_r and get the new group key. However, each node z can not decrypt the message because it can not computes A_r and B_r. For that reason GC sends for z the key encrypted the key k_z. Consequently, all non-colluding nodes in compromised level-1 groups get the new group key.

Let us observe that for each set of compromised level-1 groups in the same group of level 2 (i.e. for each $\mathscr{S}_r$ in G_r) there are the two level-1 groups $min\mathscr{S}_t$ and $max\mathscr{S}_t$ (with respect to definition 1). Let us recall that, to each level-1 group Z in $\mathscr{S}_r$ is associated two rekeying tools $GRT^Z_<$ and $GRT^Z_>$. Node j in G_r and not in Z stores in its local memory $GRT^Z_<$ if $G^y_x(j) \prec Z$ or $GRT^Z_>$ if $G^y_x(j) \succ Z$. Therefore, $GRT^{min\mathscr{S}_r}_<$ of group $min\mathscr{S}_r$ is not stored by any node in $\mathscr{S}_r$. Similarly, $GRT^{max\mathscr{S}_r}_>$ is not stored by any node $\mathscr{S}_r$. Let us observe that each node z in $G_r - \mathscr{S}_r$ with $min\mathscr{S}_r < G^y_x(z) < max\mathscr{S}_r$ does not store in its local memory $min\mathscr{C}_r$ nor $max\mathscr{C}_r$. Each node t in $G^y_{x\mathscr{S}_r} - \mathscr{S}_r$ with $G^y_x(t) < min\mathscr{C}_r$ or

$G_x^y(t) > G_x^y(t) < max\mathscr{S}_r$ stores in its local memory $GRT_<^{min\mathscr{S}_r}$ or $GRT_>^{max\mathscr{S}_r}$. Therefore, each node t can decrypt the message using key C_r or D_r and get the new key of the group. However, each node z can not decrypt the message because it can not computes the keys C_r or D_r. For that reason GC sends for z the key encrypted the key k_z. Consequently, all non-colluding in compromised level 2 groups get the new group key.

It remain to proof that all nodes in non-compromised level-2 groups get the new group key. Let us observe that for the set of compromised groups of level 2, Q (i.e. $Q = G - \mathscr{G}$) there are the groups $min\mathscr{G}_t$ and $max\mathscr{G}_t$ (with respect to definition 1). To each level-2 group in G is associated two rekeying tools $G^2RT_<^E$ and $G^2RT_>^E$. Nodes in each level-1 group T ($T \neq E$) stores in its local memory $G^2RT_>^E$ if $T < E$ or $G^2RT_>^E$ if $T > E$. Therefore $G^2RT_<^{min\mathscr{G}}$ of group $min\mathscr{G}$ is not stored by any node in $\mathscr{G}$. Likewise, $G^2RT_>^{max\mathscr{G}}$ is not stored by any node in $\mathscr{G}$. Let us observe that each node z in $G-\mathscr{G}$ with $min\mathscr{G} \prec G_x(z) \prec min\mathscr{G}$ does not store in its local memory $G^2RT_<^{min\mathscr{G}}$ nor $G^2RT_>^{max\mathscr{G}}$. Each node t in $G-\mathscr{G}$ with $G_x(t) < min\mathscr{G}$ or $G_x(t) > max\mathscr{G}$ stores in its local memory $G^2RT_<^{min\mathscr{G}}$ or $G^2RT_>^{max\mathscr{G}}$. Therefore, each node t can decrypt the message in line 20 using key E or F and get the new key. However, each node z can not decrypt the message because it can not computes the keys E and F. For that reason, GC sends for t the new group key encrypted by means of k_z. Consequently, all non-colluding nodes get the new group key. These complete the proof.

$\square_{Lemma\ 3}$

Lemma 4. *Each node s_i needs $O(\log_2 \sqrt[3]{n})$ of memory.*

Proof

Firstly let us observe that each node s_i stores in its local memory the group key CGK, the level-1 group key and the key K_i shared with GC. Let G_x and G_w^y be the level-1 group and the level-1 group respectively for node s_i. And let $a = \{t1, t2, ..., tm\}$ be the set of nodes where $id_i > id_s$ with id_s in G_x^y. And let $b = \{h1, h2, ..., hk\}$ be the set of nodes where $id_i < id_s$ with id_s in G_x^y. Therefore, s_i stores in its local memory $|S_>^s| = |a|$ and $|S_<^s| = |b|$ rekeying

tools. Given that the size of the group G_w^y is at most $\sqrt[3]{n}$, so $|a| + |b| = \sqrt[3]{n}$. Consequently, $|S_>^s| + |S_<^s| = \sqrt[3]{n}$.

Let $c = \left\{G_w^1, G_w^2, ..., G_w^q\right\}$ be the set of groups in G_x with $G_x^i < G_x^y$ and $d = \left\{G_x^1, G_x^2, ..., G_x^r\right\}$ be the set of groups in G_x with $G_x^i > G_x^y$. Therefore, s stores also in its local memory $\left|S_>^{'s}\right| = |c|$ and $\left|S_<^{'s}\right| = |d|$ rekeying tools. Given that the size of the group at most $\sqrt[3]{n}$, so $|c| + |d| = \sqrt[3]{n}$. Therefore, $\left|S_>^{'s}\right| + \left|S_<^{'s}\right| = \sqrt[3]{n}$. Let $e = \{G_1, G_2, ..., G_m\}$ be the number of groups in G with $G_i < G_x$ and $f = \{G_1, G_{2,...,G_z}\}$ be the number of groups with $G_i > G_x$. Therefore, s stores also in its local memory $\left|S_>^{''s}\right| = |e|$ and $\left|S_<^{''s}\right| = |f|$ rekeying tools. The size of the group is $\sqrt[3]{n}$ and $|e|+|f| = \sqrt[3]{n}$. Therefore, $\left|S_>^{''s}\right|+\left|S_<^{''s}\right| = \sqrt[3]{n}$. It follows that s stores 2 keys and $3\sqrt[3]{n}$, i.e. it grows as $O(\log_2 \sqrt[3]{n})$.

$\square_{Lemma\ 4}$

5. Experimental Evaluation

In this section, we evaluate experimentally our protocol. We compare its performance with two recent protocols: the protocol GREP in [4], a new group key management that uses $\sqrt{n}$ of memory and the protocol in [15]. Both protocols deals with collusion attacks as we do.

5.1. Target networks

We exercise both algorithms on randomly generated networks of different sizes. Within both algorithms, we simulate a collusion attack by choosing randomly a set of nodes that perform the collusion attacks. Then, we evaluate the consequences of these attacks on each algorithm and we compare the results.

5.2. Metrics

We evaluate our protocol and GREP in terms of the following metrics:

- **The number of compromised keys** in G when a collusion attack is performed.
- **The number of compromised re-keying tools** is the number of compromised re-keying in the group G when a collusion attack is performed.

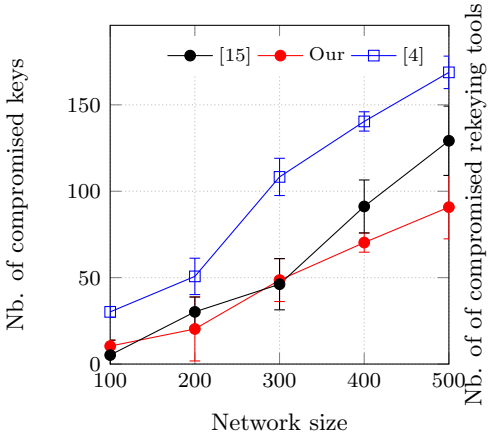

Figure 2: Figure compares the number of compromised keys between our protocol and the protocol GREP

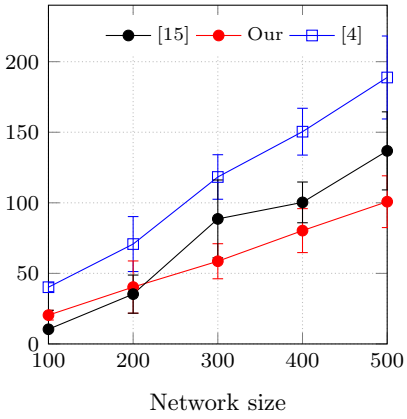

Figure 3: Figure compares the number of compromised rekeying tools between our protocol and the protocol GREP

- **The number of messages** required to rekey the whole group.

5.3. Results

Figure 2 (The number of compromised keys), and figure 3 (The number of compromised re-keying tools) show the results we obtain for network sizes varying from 50 to 500 nodes. In Figures 2 and 3 each point is averaged over 10 experiments. Errors bars show 95%-level confidence intervals computed using Student's test statistics. We remark that on all metrics, our proposed protocol outperforms the protocols in [4] and [15]. These protocols are hampered by their memory complexity, which slows it down (Figure 2), and causes them to use far more messages to re-key and change compromised re-keying tools.

Compared to the protocol in [4] we see that our protocol has less compromised keys and re-keying tools to change for random experiments. That is explained by the fact that within the protocol in [4] there is more victim groups (between colluding malicious nodes) in the level 1 groups that must be re-keyed. This is also the case for the protocol in [15], that uses one level of grouping and creates more level 1 groups for heterogeneous nodes (in terms of memory space).

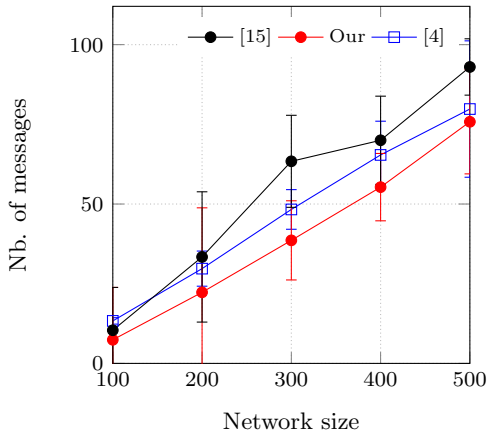

Figure 4: Figure compares the number of messages required for rekeying with protocol in [9]

Furthermore, in both protocols each compromised group has several (more than two) compromised re-keying tools. And in our protocol there are few victim groups compared to the two protocols and a victim group between the colluding nodes has only one key and two re-keying tools that must be changed. In addiction, using our protocol we can rekey more nodes using a single message compared to GREP, which explain the result in Figure 4.

6. Conclusion

There is a major difficulty in designing of efficient and secure group communication protocols when these protocols concern networks with limited resources and colluding malicious nodes. In this paper, we proposed a highly scalable Group Key Management protocol for communicating things networks which is both secure and efficient. Our new group key management protocol deals efficiently with collusion attacks and improves the existing methods in term of memory usage complexity and efficiency. Our protocol is the first to uses $O(\log_2 \sqrt[3]{n})$ memory complexity for each node.

From an algorithmic point of view, the proposed algorithm is versatile, making it an attractive starting point to address other related problems. For instance, in an heterogeneous network, the protocol could be modified to take into

account additional constraints arising from the capacities of individual nodes, such as their ability to use different memory capacities, different computing capabilities and only certain communication frequencies. Solving the problem of multi-group key management using efficient protocols remain a major challenge. The new difficulty is then to take into account the fact that node may be a part of several groups (for example in real case it proposes several services).

7. Acknowledgments

This work is supported by the Labex MS2T, which was funded by the French Government, the program "Investments for the future managed by the National Agency for Research (Reference ANR-11-IDEX-0004-02)".

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi : Internet of Things for Smart Cities. In *IEEE Internet of Things Journal, V. 1, No. 1, 2014*

[2] H. Lakhlef, M. Raynal, F. Taïani : Vertex Coloring with Communication Constraints in Synchronous Broadcast Networks. In *IEEE Trans. Parallel Distrib. Syst. 30(7), 2019*

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

[4] M. Tiloca, G. Dini: GREP: A group rekeying protocol based on member join history. In *IEEE Symposium on Computers and Communication, ISBN 978-1-5090-0679-3, 2016*

[5] H. Lakhlef, A. Bouabdallah, F. D'Andreagiovanni: A Memory-efficient Group Key Managment for Communicating Things.*Q2SWinet, The 16th ACM Symposium on QoS and Security for Wireless and Mobile Networks, 29-35, 2020*

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

[7] I. Tsai, C. Yu, H. Yokota and S. Kuo. : Key Management in Internet of Things via Kronecker Product. In *Dependable Computing, IEEE 22nd Pacific Rim International Symposium on. pp. 118–124, 2017*

[8] Q. Zhang, X. Wang, J. Yuan, L. Liu, R. Wang, H. Huang, Y. Li : A hierarchical group key agreement protocol using orientable attributes for cloud computing. In *Information Sciences, 480, 55-69, 2019*

[9] M. Tiloca, K. Nikitin, S. Raza: Axiom: DTLS-Based Secure IoT Group Communication. In *ACM Trans. Embedded Comput. Syst. 16(3): 66:1-66:29, 2017*

[10] 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 J. of Science and Technology 10.13, 2017*

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

[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.pp. 351–359, 2012*

[13] A. Shamir : Identity-based cryptosystems and signature schemes. *Workshop on the theory and application of cryptographic techniques.Springer, pp. 47–53, 1984*

[14] R. Azarderakhsh, A. Reyhani-Masoleh and Z. Abid : A key management scheme for cluster based wireless sensor networks. In *In: Embedded and Ubiquitous Computing, IEEE/IFIP International Conference on. V. 2, 2008*

[15] M. Kandi, H. Lakhlef, A. Bouabdallah, Y. Challal : A versatile Key Management protocol for secure Group and Device-to-Device Communication in the Internet of Things. In *J. Netw. Comput. Appl. 150, 2020*

[16] T. Srinivasan, S.Sath ish, R.Vi,jay Kumar, M.V.B.Vi,jayender : A Hybrid Scalable Group Key Management Approach for Large Dynamic Multicast Networks. in *The Sixth IEEE International Conference on Computer and Information Technology, 2006*

[17] R. Zhou and H. Yang : A Hybrid Key Management Scheme for Heterogeneous Wireless Sensor Networks Based on ECC and Trivariate Symmetric Polynomial. In *International Conference on Uncertainty Reasoning and Knowledge Engineering, pp. 251-255, 2011*

[18] Y. Xu, F. Liu : Hybrid key management scheme for preventing man-in-middle attack in heterogeneous sensor networks. In *Computer and Communications (ICCC) 3rd IEEE International Conference on, pp. 1421-1425, 2017*

[19] P. Ahlawat and M. Dave : A cost-effective attack matrix based key management scheme with dominance key set for wireless sensor network security. In *International Journal of Communication Systems 31.12, 2018*

[20] S. Aissani, M. Omar, A. Tari, and F. Bouakkaz : KMS: micro key management system for WSNs. In *IET Wirel. Sens. Syst. 8(2): 87-97, 2018*

[21] K. Hamsha K, G. S. Nagaraja : Threshold Cryptography Based Light Weight Key Management Technique for Hierarchical WSNs : in *Ubiquitous Communications and Network Computing. UBICNET 2019. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 276. Springer, 2019*

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

[23] Z. Liu, X. Huang, Z. Hu, M.K. Khan, H. Seo and L. Zhou : 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.*

[24] M. Messai, H. Seba, and M. Aliouat : A new hierarchical key management scheme for secure clustering in wireless sensor networks. In *International conference on wired/wireless internet communication. Springer. 2015, pp. 411-424.*

[25] G. Thevar and G. Rohini : Energy efficient geographical key management scheme for authentication in mobile wireless sensor networks. In *Wireless Networks 23.5 (2017), pp. 1479-1489.*

Hicham Lakhlef is associate professor at the University of Technology of Compiegne (UMR CNRS 7253). He coauthored more than 50 international publications. His research interests are in parallel and distributed algorithms, WSNs, clustering, self-reconfiguration, optimization, routing, and internet of things.

Abdelmadjid Bouabdallah is professor at University of Technology of Compiegne (UMR CNRS 7253), where he is leading the Networking and Security research group and the Interaction and Cooperation research of the Excellence Research Center LABEX MS2T. His research Interest includes Internet of things, QoS, security, unicast/multicast communication, Wireless Sensor Networks, and fault tolerance in wired/wireless networks.