Decentralized Lightweight Group Key Management for Dynamic Access Control in IoT Environments

Maissa Dammak, Sidi-Mohammed Senouci, Mohamed Ayoub Messous, Mohamed Houcine Elhdhili, Christophe Gransart

Decentralized Lightweight Group Key Management for Dynamic Access Control in IoT Environments

Maissa Dammak[1], Sidi Mohammed Senouci[1], Mohamed Ayoub Messous[1], Mohamed Houcine Elhdhili[2], Christophe Gransart[3]

1

***Abstract* — Rapid growth of Internet of Things (IoT) devices dealing with sensitive data has led to the emergence of new access control technologies in order to maintain this data safe from unauthorized use. In particular, a dynamic IoT environment, characterized by a high signaling overhead caused by subscribers' mobility, presents a significant concern to ensure secure data distribution to legitimate subscribers. Hence, for such dynamic environments, group key management (GKM) represents the fundamental mechanism for managing the dissemination of keys for access control and secure data distribution. However, existing access control schemes based on GKM and dedicated to IoT are mainly based on centralized models, which fail to address the scalability challenge introduced by the massive scale of IoT devices and the increased number of subscribers. Besides, none of the existing GKM schemes supports the independence of the members in the same group. They focus only on dependent symmetric group keys per subgroup communication, which is inefficient for subscribers with a highly dynamic behavior. To deal with these challenges, we introduce a novel *Decentralized Lightweight Group Key Management architecture for Access Control in the IoT* environment (*DLGKM-AC).* Based on a hierarchical architecture, composed of one *Key Distribution Center* (KDC) and several *Sub Key Distribution Centers* (SKDCs), the proposed scheme enhances the management of subscribers' groups and alleviate the rekeying overhead on the KDC. Moreover, a new master token management protocol for managing keys dissemination across a group of subscribers is introduced. This protocol reduces storage, computation, and communication overheads during join/leave events. The proposed approach accommodates a scalable IoT architecture, which mitigates the single point of failure by reducing the load caused by rekeying at the core network. *DLGKM-AC* guarantees secure group communication by preventing collusion attacks and ensuring backward/forward secrecy. Simulation results and analysis of the proposed scheme show considerable resource gain in terms of storage, computation, and communication overheads.**

***Index Terms*—IoT, Group Key Management, Access Control, Scalability, Dynamic environment, Security, Group communication.**

I. Introduction

IoT has been introduced as a universal and ubiquitous paradigm that connects transparently and seamlessly a multitude of digital devices to the Internet [1]. Recently, IoT devices are progressively becoming a large part of people's daily lives. They are widely used in various kinds of applications, such as environmental sensing and industrial monitoring (e.g., smart city, smart hotel, smart office, industry 4.0) [1]. A generic IoT network architecture, as shown in Fig.1, is composed of a set of physical objects (i.e., smart things) that are interconnected to exchange and collect data over the Internet. [2] has predicted that, by 2025, more than 41.6 billion connected IoT devices will be used worldwide. These smart things form a diverse and heterogeneous network of interconnected physical objects with a wide range of functionalities and requirements; one essential feature being data collection. The vast majority of IoT devices have minimal resources, in terms of computation, communication, and storage, preventing them from efficiently performing cryptographic operations, thus rising more security challenges. Indeed, large scale IoT deployments still face serious security issues that still need to be addressed, such as authentication, privacy preservation, and data integrity [3][4].

In order to safeguard IoT data from tampering and unauthorized access, an appropriate scheme for access control is more crucial than ever. To ensure this, GKM is one promising approach, which would be used to provide access control to data streams for legitimate users only [4]. In other words, it consists of creating a group key that will be shared between a device's group and its current subscribers, such that the device can encrypt its data, and only the subscribers can decrypt it. This mechanism is suitable for IoT environments as it does not require a trusted third entity. This can be achieved through a publish-subscribe messaging model, such as MQTT [5].

Given the dynamic nature of IoT environments, where member's group can intermittently join and leave the system (e.g., reservation IoT systems), safeguarding IoT data from unauthorized access represents a primordial security issue. Therefore, enforcing access control is reduced to solving the GKM problem. Moreover, in order to guarantee backward and forward secrecy, the shared keys need to be changed whenever a new member joins or an existing one leaves its group [6]. To efficiently reduce the overhead keys management, resulting mainly from rekeying, GKM is extensively studied in the

[1]M. Dammak, S. M. Senouci and M. A. Messous are with the DRIVE Laboratory, University of Burgundy, Nevers, France (e-mail: {maissa.dammak;sidi-mohammed.senouci;ayoub.messous}@u-bourgogne.fr).

[2]M.H. Elhdhili is with the CRISTAL Laboratory, ENSI, Manouba University, Manouba 2010, Tunisia (e-mail: mohamedhoucine.elhdhili@ensi-uma.tn)

[3]C. Gransart is with IFSTTAR/COSYS/LEOST laboratory, Villeneuve d'Ascq, France (e-mail: christophe.gransart@ifsttar.fr).

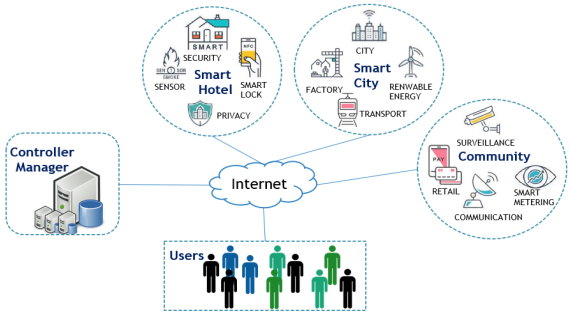


Fig.1. Generic IoT Environment

literature [12]. Most of the existing GKM schemes are not suitable for IoT applications. Indeed, existing GKM for IoT applications are designed to manage communication within a single group, and the GKM schemes introduced for access control in the IoT environment are mostly based on a centralized access management architecture. Consequently, they are not suitable for a scalable and dynamic IoT environment comprising multiple groups. In fact, many users can subscribe to numerous services offered by different IoT devices and change their interest frequently over time. Thus, maintaining an efficient GKM in a dynamic IoT environment remains a challenging issue due to the rekeying process that affects all members in the same group for joining/leaving events. Therefore, all members should update their shared group access keys. Hence, an efficient group key mechanism should be introduced to reduce the rekeying dependence of members in the same group, and thus reducing overhead.

To solve the rekeying dependence, minimize resulting overhead and achieve scalable access management for dynamic IoT environment, this work introduces a new *Decentralized Lightweight Group Key Management Architecture for Access Control* named *DLGKM-AC*. We consider in our use-case, in the context of the European project PARFAIT [28], an extensive reservation system for franchise hotels. In this scenario, key cards and smartphones might be interchangeably used to give access permissions for guests in different rooms. They can also be used to control the usage of various facilities according to room classes' and purchased services. When a guest checks out, and the room becomes vacant, the devices should stop sending the room's information and receiving information from other devices.

The main idea of *DLGKM-AC* is to create an efficient and flexible mechanism to secure distribution of contents to eligible subscribers. A hierarchical scheme comprising a central *Key Distribution Center* (KDC) and several *Sub Key Distribution Centers* (SKDCs) to manage groups of subscribers and to mitigate the single point of failure issue is introduced. Key management tasks in *DLGKM-AC* are offloaded to several SKDCs, which allow enhancing the system's performances in terms of computation and communication by reducing the overhead caused by membership changes (join/leave). The KDC manages device groups, while each SKDC manages user groups, which provides scalability for our *DLGKM-AC*. Furthermore, *DLGKM-AC* introduces a new key management mechanism that allows reducing the rekeying dependence of users in the same group. *DLGKM-AC* is a scalable and flexible access management protocol that is based on the GKM mechanism. It improves the computation capability, the storage capacity, and the communication overhead. The main contributions and novelties of this paper are summarized as follows:

- Presenting a new lightweight decentralized keys distribution architecture to ensure forwarding valuable and sensitive information to legitimate users in a scalable and secure manner,
- Designing a rekeying mechanism suitable for multiple groups of IoT devices and various users' groups whenever memberships change, which ensures a flexible access management system,
- Presenting a master token management algorithm that creates and updates a master token and multiple slave tokens for handling user groups, which achieves the independence of user during the rekeying process,
- Ensuring security requirements, backward and forward secrecy even with changes in users and devices membership, and resisting to the collusion attack,
- Minimizing computational and storage overheads for users and IoT devices, and also communication overhead for the overall system, which is proved through extensive analytical study and simulation work.

The remainder of this paper is structured as follows: First, related work is described in Section II. Then, we discuss the necessary background related to our scheme in Section III before presenting the overall system architecture, attacker model and different system requirements in Section IV. The proposed *DLGKM-AC* for IoT is introduced in Section V. Security and performance analysis in terms of storage, communication and computation overheads are summarized in Section VI. Finally, conclusions and future works are given in Section VII.

II. Related Work

The dynamic nature of group communications makes safeguarding data from unauthorized access a significant challenge. The larger problem of access control is reduced to GKM, where a group key is shared by the group members to define the access permissions. Table I summarizes and classifies existing GKM solutions based on different attributes and criteria as follow: (i) *Environment* of its application, such as wired Internet [6], wireless sensor networks (WSN) [7][9][11], ad hoc networks [8], wireless body area networks (WBAN) [10] and IoT environment [13][14]. *(ii) Network model* that could be centralized, decentralized or distributed. *(iii)*the used *Cryptography types,* and essential security services *(iv) backward secrecy and (v) forward secrecy*, where shared keys need to be updated whenever a new member joins, or an existing one leaves its group. *(vi) Key independence* to ensure the independence of keys from each other. (vii) *Vulnerability to collusion attack* (collaboration of adversaries to compromise a communication) for which rekeying is important to maintain security. However, this process may cause a lot of key management overhead and leads to *(viii) Single point of failure*, especially in a (ix) *Scalable* environment that supports (x) *Multiple group services* and composed of (xi) *Dynamic publishers* and dynamic subscribers. Hence, ensuring (xii) *Subscribers' independence* makes subscribers of one group independent from the entire group in the rekeying process of the group key after a join/leave event in the group.

TABLE. I: COMPARISON OF EXISTING GKM SCHEMES

	[10]	[11] [9]	[20] [21] [22]	[13]	[17] [18]	[8]	[14]	[25] [26]
(i)Environment	WBAN	WSN	CC	IoT	WSN	IPv6	IoT	IoT
(ii)Network model	Cent	Cent	Cent	Cent	Decent	Decent	Decent	Distr
(iii)Cryptography type	Sym , P	Sym , Asym	ABE	Sym, Asym	Sym, Asym	Sym	Sym	Asym
(iv)Forward secrecy	Yes	No	Yes	No	No	Yes	Yes	Yes
(v)Backward secrecy	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
(vi)key Independence	No	Yes	No	Yes	No	No	Yes	No
(vii)Vulnerable to collusion attack	No	Yes	Yes	Yes	No	No	Yes	No
(viii)Single point of failure	Yes	Yes	Yes	Yes	No	No	No	No
(ix)Scalability	No	No	No	No	High	Moderate	Limited	Limited
(x)Support multiple group services	No	Yes	No	Yes	Yes	Yes	No	No
(xi)Support publishers' dynamism	No	No	No	Yes	No	No	No	No
(xii)Support subscribers' independence	No	No	No	No	No	No	No	No

NB: CC: cloud computing, cent: centralized, decent: decentralized, Sym: symmetric, asym: asymmetric, P: polynomial, ABE: Attribute Based Encryption

Authors in [12] surveyed numerous key distribution schemes over wireless networks and classified them into centralized, decentralized, and distributed schemes. Centralized schemes use only one server known as the key distribution server (KDC) for creating and distributing encryption keys. Distributed schemes do not have a specific KDC; they rather generate group key either in a collaborative manner between the group members or by one member. Moreover, each member must keep track of the other members to make robust communication. Besides, membership change events (join/leave) cause a high processing and communication overheads [25], which may lead to a congestion problem in a dynamic IoT environment. In contrast, decentralized schemes divide the system into several subgroups, thus, reducing the load on the KDC and offering a solution to scalability issues. Furthermore, a subgroup manager is responsible for keeping track of the group's members, which may reduce computation and storage overhead on members.

The distribution of encryption keys in the different mentioned GKM architectures is further ensured by using two main cryptographic types (symmetric and asymmetric). Two fundamental and efficient GKM schemes were proposed: The Logical Key Hierarchy (LKH) [15] and the One-way Function Tree (OFT) [16] based on symmetric keys (traffic key and encryption key) to distribute the updated encryption keys. In contrast to LKH, all the OFT implementations suffer from collusion attacks and increase devices' computational overhead for obtaining group keys. Hence, OFT is far from ideal in an IoT environment, where the communicating devices may have limited computational power.

Additionally, [20] [21] schemes provided fine-grained access control Attribute-Based Encryption (ABE) to manage keys' update. However, ABE is a cumbersome mechanism that relies on asymmetric cryptography, which is unsuitable for running on resource-constrained IoT devices [22]. Besides, asymmetric encryption mechanisms are also used in key management schemes [23] [24]. Specifically, Porambage et al. [7] proposed a group key establishment protocol for multicast communication by using the Elliptic Curve Cryptographic (ECC) operations. Even though, the latter are known to be suitable for resource-constrained devices; their protocol does not efficiently manage the rekeying process. Furthermore, all previous mentioned schemes are designed for single multicast groups, but users may subscribe to multiple services. To ensure many multicast groups, Park et al. [11] accommodate various services' groups. Their scheme addressed rekeying in the wireless mobile environment, which is based on a centralized architecture and a LKH mechanism to manage multiple communications. Likewise, Mapoka et al. [17] proposed using a distribution list of the session key and key update slot for each subgroup. This list is centrally managed by a node called the area key distributor. The proposed protocol alleviates the 1-affect-n phenomenon and transmission overhead of the core network, but it does not ensure the forward secrecy. Hence, Zhong et al. [18] proposed another protocol called area based multiple GKM that securely provides services when users migrate to different wireless networks, which ensures forward secrecy. Nonetheless, its high overhead, due to revocation events, makes it unsuitable for dynamic IoT environments.

To address the rekeying issue in the IoT environment, Tsai et al. [19] proposed a lightweight symmetric key establishment based on the Kronecker product. However, their protocol does not consider the key update when users or devices join or leave the system, which lacks forward and backward secrecy. Furthermore, Abdmeziem et al. [14] proposed a decentralized batch-based group key that includes several subgroups managed by key servers. This scheme considered long term and short-term keys per group, which are common to all nodes. Nevertheless, [14] does not ensure communication between multiple groups and it requires large storage and computation resources. Their work was enhanced to decrease the communication overhead by adopting a Distributed Batch-based Group Key [26]. It is based on polynomial cryptography to set up the key for collaborative groups in the IoT environment. However, these schemes are limited for managing communications in one group and do not consider communications between different groups and services. Kung et al. [13] took advantage of the Chinese Remainder Theorem (CRT) based construction proposed by Park et al. [11] to accommodate multiple device groups. They established a two-tier centralized system KDC, where each group (devices or users) runs LKH to handle updates of keys efficiently. However, communication within a user group is based on symmetric group key, which leads to the dependence between all its members. Therefore, after each event (triggered by a join/leave user operation), the rekeying process induces all the members in the entire group to update their group key, and thus, increases the computation overhead.

In summary, and as mentioned in Table I, existing GKM solutions do not support the independence of members in the

same group, where each member needs to update its key after every join/leave event. Specifically, they focus only on symmetric group key per subgroup communication. Consequently, the rekeying performance is decreased when the number of subscribers is high and varies frequently. Moreover, lesser attention is paid to achieve efficient and scalable GKM for access control among a dynamic IoT environment, where many users (subscribers) can subscribe to different IoT services and frequently change their interest over time. Hence, throughout this paper, we propose a flexible access management protocol that is based on the GKM mechanism. More specifically, to the best of our knowledge, a new decentralized GKM to secure group communication, offers the scalable feature in a dynamic IoT environment, alleviates the rekeying overhead caused by the member changes, and reduces the load on the KDC.

III. Preliminaries

In this section, we briefly present the background and the main mechanisms used in our approach. We first describe the LKH scheme used for efficient key management of different device groups (DGs). Then, we present the Master Key Encryption (MKE) based Generalized Chinese Remainder Theorem (GCRT) that is used for managing multiple user groups (UGs) and various users.

A. Logical key Hierarchy

LKH uses a tree structure to manage the distribution of keys. This method reduces communication costs by multicasting multiple key-encryption keys $O(logn)$ for n devices per group [15]. LKH structure is composed of devices located at the leaf nodes of the tree and a central control center called KDC, which maintains the keys' virtual tree. Each leaf node shares a secret key with the KDC. The root of the tree holds the group Key (GK), and the internal nodes hold Key Encryption Keys (KEK). KEKs are known by each device in the leaf nodes within the same subtree rooted to a specific internal node. Furthermore, KEKs compose a Path key (PKt), which is used later to update group keys efficiently. In a complete tree with n devices, each device stores $\log(2n+1)$ keys [15]. To manage the group communication within IoT devices groups, we use the LKH scheme. Since, multiple users may subscribe to the same IoT device group, it would be more efficient if all these devices and all their subscribed users share a group key for encryption. Traffic Encryption Key (TEK) is a traffic key used to encrypt data published by a device group to its subscribers. This traffic key should be efficiently updated when a new user joins, or an old one leaves to ensure forward and backward secrecy. To do so, we explain the key management scheme used to manage communication with users in the next subsection.

B. Master Key Encryption (MKE)

Users subscribe to many *DGs* in the system to get data. For this purpose, each user gets all *TEKs* of *DGs* to which it is subscribed. Otherwise, the users may subscribe to the same *DGs*, which would lead to an increase in the overhead when an old user unsubscribes, or a new one subscribes. Thus, managing group communication with users is essential to reduce the cost of updating *TEK* after each join/leave event.

In this context, we define the concept of master key encryption (MKE), which is a key management scheme based on GCRT. MKE permits multiple decryption keys to decrypt the same message encrypted by an encryption key [11]. The main idea of the master key encryption scheme is to generate one master key and several slave keys, where the master key encrypts a message that can be decrypted by all legitimate salve keys. The MKE scheme can alleviate the rekeying cost resulting from the symmetric cryptography. Hence, Park et al. [11] have proposed a general MKE algorithm to lessen the rekeying cost of the group key using a master key.

***Theorem 1*:** Let $\{p_1, p_2, \dots, p_N, q_1, q_2, \dots, q_N\}$ a set of safe prime numbers. If all public keys satisfy the following condition, $e_1 \equiv e_2 \equiv \cdots \equiv e_N \, mod(4)$, Then, there exists a unique master key, e_M modulo $4x_1y_1x_2y_2 \dots x_Ny_N$, where $x_i = (p_i - 1)/2$ and $y_i = (q_i - 1)/2$, $1 \le i \le N$.

Theorem Proof: consider there are N public/private slave key pairs $(e_i, d_i), i \le N$ with (p_i, q_i) being the ith safe prime number pair, and one master key pair(e_M, d_M). For simplicity, we now consider the modulus of the prime pairs $\emptyset(p_iq_i) = (p_i - 1)(q_i - 1)$ are mutually prime to each other. For a plaintext P and a ciphertext C, the master key should satisfy:

$$P^{e_M} \equiv P^{e_i} \, mod \, (p_iq_i) \quad (1)$$

$$C^{d_M} \equiv C^{d_i} \, mod \, (p_iq_i), 1 \le i \le N \quad (2)$$

According to Euler's theorem, the necessary condition for the equation above is:

$$e_M \equiv e_i \, mod \, \big(\emptyset(p_iq_i)\big), \quad d_M \equiv d_i \, mod \, (\emptyset(p_iq_i))$$

The set of safe prime numbers satisfies the following condition: $e_1 \equiv e_2 \equiv \cdots \equiv e_N \, mod(4)$. Then, there exist a unique master key, $e_M \text{mod}(4x_1y_1x_2y_2 \dots x_Ny_N)$, where $x_i = (p_i - 1)/2$ and$y_i = (q_i - 1)/2$, $1 \le i \le N$, solution of a system congruence that can be calculated by the GCRT as follows: $e_M = \sum_{i=1}^{N} e_i \, M[i] \, N[i]$,
Where $M[i] = \big(\prod_{j=1}^{N} x_j y_j\big) / x_i y_i$ and $N[i]$ is an integer such that $M[i] \, N[i] \equiv 1 mod(4x_iy_i)$.

Based on *theorem 1*, [11] proposes a general MKE algorithm, which generates and modifies the master key and the key pairs, respectively. In our proposed scheme, we take advantage of this algorithm and propose an optimized algorithm for membership *renewal* and *revocation*. This algorithm is described in the *DLGKM*-AC scheme section. In the following, we define the system model and its underlying requirements.

IV. System Model

In this paper, we propose a decentralized group key management scheme where the numbers of users and devices

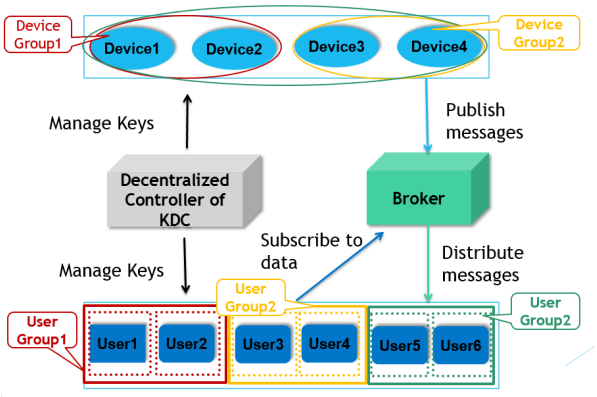


Fig.2. Proposed network model

change frequently. Before presenting the solution, we introduce the overall system, the attacker model and the system requirements. The architecture of the proposed network model, shown in Fig.2, illustrates a typical three-tier scheme used for a smart hotel for our use case scenario. The entire system considers three essential layers: publishers, subscribers and group key manager.

- The *publisher layer* contains IoT devices, such as smart door locks or IP cameras, collecting and sending data to subscribers. These constrained IoT devices have limited computation, storage and energy resources.
- The *subscriber layer* is composed of a set of users that want to get access to data of the publisher layer. A user can be a device owner with legitimate, full and permanent control or a guest user with only limited access. A user communicates and receives data from IoT devices via his/her smartphone.
- The *group key manager layer* is responsible for generating the system parameters and managing group members by providing required encryption keys used to control the access to data. The group key manager in our system is considered as a fully trusted third party.

The intended approach considers a dynamic reservation system in an IoT environment, where both the number of users and IoT devices might frequently change over time. Indeed, a user may join or leave at any time. Likewise, an IoT device can be introduced in or removed from the system at any time. Thus, it is crucial to manage the distribution of encryption keys to secure both group communication and data transmission from possible threats that will be defined in the next subsection.

A. *Threat model*

The proposed GKM system model may confront different type of attacks that may threaten the security of the network. Thus, we define the attacker capabilities in compromising the GKM access control scheme based on the active insider and active outsider adversary models. An attacker *A* may be either an outsider, who has no access to any IoT device, or an insider who attempts to increase the access possibility. For example, a revoked user who has no longer access to future communication and yet tries to retrieve information on access policies to extend access scope. Another example is an attacker *A* that aims to extract sensitive information, such as the encryption key, to break the current encryption scheme and get access to data without proper permissions. *A* may cooperate with other members in the system to derive keys that he/she cannot obtain individually, which is known as a collision attack. Besides, the attacker may also be a compromised device, where he/she may masquerade as a legitimate communication partner before initiating communication with other participant in the network to gain access to data that are unknown to him. However, he cannot compromise or break the cryptographic primitives.

B. *System requirements*

Several requirements are identified and discussed for effective GKM. Generally speaking, an efficient and practical GKM should address the following requirements [4] [12]:

a. *Security requirement*

In order to ensure transmitted data security, in a dynamic IoT environment, the system should achieve some services. On the one hand, it should avoid any leaving member from decrypting the future exchanged messages, to maintain forward secrecy. On the other hand, new members that join the system should be prevented from decrypting the previous communications to guarantee backward secrecy. Forward/backward secrecy are accomplished through an efficient key updating process, where all keys should be completely independent from each other in order to safeguard the key independence security service.

b. *Efficient functioning requirement*

The efficient functioning of key management protocols is justified by a minimum overhead cost of different metrics. First, it reduces the number of keys stored on both users and IoT devices, which results in low storage overhead. Second, it decreases the required computation power from users, IoT devices and servers, which increases the efficiency by reducing the system response time. Finally, it minimizes the number of exchanged messages on the system, which raises the flexibility of the overall system and thus achieves a low communication cost.

c. *Performance requirement*

The performance is mainly related to factors that affect group communication. It includes the scalability, which determines the capability to handle variable group sizes and high membership changes. Besides, key management schemes suffer from the 1-affects-n phenomenon, where a failure of a single server leads to the collapse of the whole system. Hence, it is essential to avoid this phenomenon and assure the availability in a large and scalable system.

V. Proposed *DLGKM-AC* Scheme Description

In this section, we introduce the proposed scheme, and we explain how it handles member's joining and revocation events.

A. *Overview*

DLGKM-AC is composed of three essential layers, shown in Fig.3. The upper and lower layers define groups of devices (*DGs)* and users (*UGs*), respectively. In contrast, the middle layer defines the decentralized controller, KDC, which is responsible for key management between and within groups.

- ***Device groups DGs*:** *DLGKM-AC* for IoT environment establishes a fixed number of DGs based on their

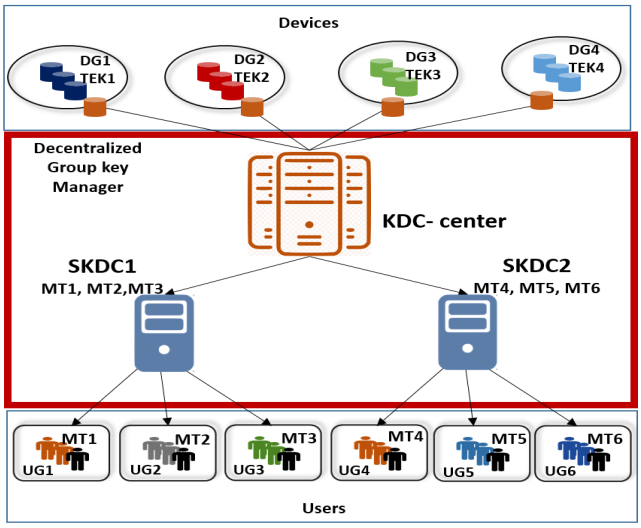


Fig.3. Proposed system model

functionalities, security levels, localization, etc. When a new IoT device joins the system, it is assigned to precisely one of the existing DGs. The LKH structure achieves the group communication within a DG.

- ***User groups UGs:*** *DLGKM*-AC for IoT environment creates user groups (*UGs*) based on user's interest and reservation's period. Each user joins one of those *UGs*, and encryption keys are distributed using the MKE technique within a *UG.*
- ***Decentralized group key manager:*** DLGKM-AC for IoT environment is a decentralized architecture of servers, which is composed of one KDC and several SKDC. The number of SKDC is not fixed and depends on the IoT application needs. More specifically, the number of SKDC is influenced by the characteristics of SKDC like storage, computation capacities, and the number of registered users

KDC is the central server that relates publishers to the rest of the system, and it manages the keys' update process within *DGs*. Further, KDC has a backup server that maintains the last updated version of keys in the system, which is sent to the backup periodically after the rekeying process. Besides, SKDCs manage the group communication within *UGs*, where users frequently join and leave the system. Hence, the decentralized aspect of the controller, where SKDCs are used, allows reducing the load on the KDC. Multiple user groups are under the control of one SKDC depending on users' localization, which solves the problem of single-point failure (SKDC failure) and ensures the scalability of the system. Besides, we assume that the decentralized KDC can establish a one-time secure channel with users and devices, which can be used to authenticate and configure a newly joined user/device (e.g., by installing a shared secret key) before sharing with them the encryption keys. We can summarize the different encryption keys in our scheme into two main categories: (i) Traffic Encryption Key (TEK) and (ii) Key Encryption Key (KEK). The traffic keys are used to encrypt/decrypt data, while the key-encryption keys are used to encrypt/decrypt traffic keys to distribute them securely. Table II presents the different keys used in this paper.

TABLE II SUMMARY OF DIFFERENT TYPES OF KEYS

Traffic Encryption Keys (encrypt data)	• **TEK**: encrypts data of DG • **DK**: encrypts data of one device
Key Encryption Key (encrypt traffic key)	• **KEK** & **GK** encrypt and distribute *TEK* within a DG
	• **MT** encrypts updated *TEK* keys to users in SKDC
	• **ST** decrypts updated *TEK* keys in UG

Definition*:* Let $\mathrm{U} = \{\mathrm{U}_1, \mathrm{U}_2, \dots, \mathrm{U}_n\}, \mathrm{n} \leq \mathrm{N}$ be the universe of users controlled by one SKDC. Each user in a network can subscribe to one or more services of device groups DGs among a total of M (DG) denoted by $\{\mathrm{DG}_1, \mathrm{DG}_2, \dots, \mathrm{DG}_M\}$. Let $\mathrm{UG} \subset \mathrm{U}$ be the set of users who subscribe to the same set of DGs during the same time T. Let $\{\mathrm{UG}_1, \mathrm{UG}_2, \dots, \mathrm{UG}_u\}$ be the set of user groups UGs. Here, each UG possesses an ID defined as follows:

$$ID_j = \{A_{j,b} \mid 1 \leq j \leq M \mid b \in [0,1]\},$$

$$ID_j = \begin{cases} A_{j,0} = 0\,, UG\ is\ not\ subscribed\ to\ the\ DG_j \\ A_{j,1} = 1,\ UG\ is\ subscribed\ to\ the\ DG_j \end{cases} \quad (3)$$

TABLE III SUMMARY OF SYMBOLS AND THEIR DESCRIPTION

Symbol	Description
TEK	Traffic Encryption Key
KEK	Key Encryption Key
M	Total number of Device Groups
N	Total number of Slave Keys under SKDC
(e_M, d_M)	Master Key
(e_i, d_i)	Slave Key
MT, ST	Master Token, Slave Token
DG_y	Device Group *y*
DK_j	Shared secret key between device *j* and KDC
UG_x	User Group *x*
UK_i	Shared secret key between user *i* and SKDC
GK_y	Group Key for device group *y*
PK_t	Path Key
$h(.),[.]_K$	Hash Function, Encryption function using encryption key K.

B. Scheme construction

In this section, we detail the structure of the proposed scheme and explain how to manage group keys according to users and devices changes efficiently. Our system can be divided into five parts: (i) Initialization, (ii) device group registration, (iii) User group registration, (iv) Dynamic changes in users' membership (Join/Leave), and (v) IoT device changes (Join/Leave). We give a brief description of all used symbol in Table III.

a. Initialization

The group key manager performs the initialization, and it includes both SKDCs and KDC initialization.

i. KDC initialization

KDC runs **MkeyGen** algorithm based on GCRT, to generate a master key and several salve keys to communicate with several SKDCs under its control. Besides, when a new SKDC is added to the system, KDC has to run the **MkeyGen** algorithm to generate a slave key for the new SKDC and update its master key. Moreover, KDC establishes a secure channel with devices and users, and creates DGs and assigns UGs to SKDC. We consider an example of architecture with 4 *DGs* and 6 *UGs* in Fig.3.

ii. SKDC initialization

Each SKDC in the system runs algorithm 1, the master key generation algorithm named **MkeyGen**, to initialize the system for many users. Let *N* be the maximum number of slave keys provided by SKDC. First of all, the SKDC generates a master key $(\boldsymbol{e_M}, \boldsymbol{d_M})$ and a set of *N* public-private key pairs, named slave keys, $SK = \{(\boldsymbol{e_i}, \boldsymbol{d_i});\ \boldsymbol{1 \leq i \leq N}\}$ through **MkeyGen.**

SKDC defines a function *f* which maps a pair key from a set of slave keys to {0, 1}as follows:

$$f: S \rightarrow \{0,1\}, \text{ where}$$

$$f: \begin{cases} f\,((\boldsymbol{e_i}, \boldsymbol{d_i})) = 1\,, & ((\boldsymbol{e_i}, \boldsymbol{d_i})\ is\ assigned\ to\ a\ user) \\ f\,((\boldsymbol{e_i}, \boldsymbol{d_i})) = 0\,, & ((\boldsymbol{e_i}, \boldsymbol{d_i}) is\ not\ assigned\ to\ any\ user) \end{cases} \quad (4)$$

After the generation of master and slave keys, SKDC initializes all pair keys using (4) as follows: $\boldsymbol{1 \leq i \leq N}, f((\boldsymbol{e_i}, \boldsymbol{d_i})) = 0$.

b. Device Groups registration

Multiple IoT *DGs* are established, and each *DG* accommodates devices with similar attributes (i.e., security levels, localization...). *KDC* generates *KEKs* for devices in each *DG*. First, KDC sets a binary LKH tree for the universe of devices in each *DG*, which will be used to distribute updated keys to devices. In the tree, each intermediate node holds a *KEK*. A set of *KEKs* on the path nodes from a leaf to the root are called Path Keys (PK_t).

Algorithm 1 Master Key Generation **MKeyGen**

Inputs: A set of safe prime numbers $p_1, p_2, \dots, p_N, q_1, q_2, \dots, q_N$.
Output: One master key e_M and N slave public-private key pairs $S = \{(e_i, d_i)\ ;\ 1 \le i \le N\}$

1: $S = \{\}$;
2: ***For*** $i = 1$ ***to*** N
$\varphi_i = (p_i - 1) \times (q_i - 1)$;
$x_i = (p_i - 1)/2$;
$y_i = (q_i - 1)/2$;
$e_i = 4 \times Random + 1$;
$d_i = e_i^{2(x_i-1)(y_i-1)} - 1 \ mod\ 4x_iy_i$;
$S = S + \{(e_i, d_i)\}$;
End For
3: $product = 1$;
4: ***For*** $i = 1$ ***to*** N
$product = product \times (x_iy_i)$;
End For
5: ***For*** $i = 1$ ***to*** N
$M[i] = n/(x_iy_i)$;
$N[i] = M[i](x_i - 1)(y_i - 1) - 1 \ mod\ (x_iy_i)$;
End For
6: $e_M = 0$;
7: ***For*** $i = 1$ ***to*** N
$e_M = (e_M + (e_i \times M[i] \times N[i]))$;
End For

Algorithm 2 Master Token Generation **MTokenGen**

Inputs: Number of user r, Time T, $\mathbf{e}_M$, S
Output: MT_K Master Token of UG_K and list S_K

1: $\mathbf{e}_{M_K} = \mathbf{e}_M$;
2: $Comp = 0$;
3: $S_K = \{\}$;
// Select a list of slave keys for UG_K, $S_K = \{e_i^{1K}, e_i^{2K}, \dots, e_i^{rK}\}$.
// $e_i^{1K} = e_i$ assigned to user in UG_K
4: ***While*** *(Comp <r)* do
Select a random (e_i, d_i) from $S = \{(e_i, d_i) \mid 1 \le \mathbf{i} \le \mathbf{N}\}$
5: ***If*** $f((e_i, d_i)) == 0$
Then
$S_K = S_K + \{(e_i, d_i)\}$;
$f((e_i, d_i)) = 1$;
$comp++$;
End if
End while
6: **For** *i* = 1 to *N*
If $e_i \notin S_K = \{e_i^{1K}, e_i^{2K}, \dots, e_i^{rK}\}$
Then
$\mathbf{e}_{M_K} = \mathbf{e}_{M_k} - \mathbf{e}_i\ \mathbf{M[i]}\ N[i]$;
End if
End For
7: $MT_K = (e_{M_K} + T)$

The LKH tree is constructed by KDC as follows:

- Devices in *DG* are assigned to the leaf nodes of the tree. Random keys DK_j are generated and assigned securely to each leaf node.
- The root node holds group key *GK* to communicate with devices and *TEK* to encrypt data of *DG.*
- Each device D_j in *DG* receives the path keys PK_l from the root node to the parent node of the tree, securely.

Then, the path keys will be used as KEKs to encrypt the group key by the KDC in each rekeying process and to distribute updated encryption keys to leaf nodes.

c. User Groups registration

In this phase, multiple user groups UG_K are constructed, and each UG_K accommodates r_k users with the same interest for a period T. Each user U_i in UG_K is authenticated before joining the system and shares a secret key UK_i with SKDC. The SKDC assigns a user group *ID de*noted by $ID_{UG_K} = \{A_{j,b} \mid 1 \le j \le M \mid b \in [0,1]\}$ using (3), where *j* defines DG_j, and b outlines the user group subscription to the corresponding DG_j when b=1. Otherwise, when b=0, this means that the user group is not subscribed to the corresponding DG_j.

The communication within user groups is based on Master Token Encryption (MTE), which reduces the communication and computational complexities. Besides, MTE also supports efficient key updating. Therefore, SKDC conducts **MTokenGen** (*algorithm 2*) to generate the group key MT_K and a set S_K of slave tokens *STs* for UG_K. Then, each user member U_i in UG_K receives a *ST* through a secure unicast.

The SKDC adds user group information $(ID_{UG_K}, MT_K, S_K, r_K, T)$ to the list of active user groups. Subscribers and IoT devices can join or leave the communication session over time. Hence, the keys should be changed in each join and leave event. Therefore, dynamic membership management is a critical component of any security architecture to ensure the backward and forward secrecy, which will be detailed in the next subsection.

d. User membership changes (join/leave)

In this section, the key updating scheme is illustrated according to two events; namely, the user join event, and the user leave event. In order to describe the keys' update process of DLGKM-AC, and for simplicity, we consider the case of a user that joins/leaves the user group *UG1,* where users are subscribed to *DG1, DG2,* and *DG4.*

i. When a user joins a group:

Consider a user U_{join} that joins an existing group UG_K; few steps are necessary as introduced below: First, U_{join} should register to SKDC after being authenticated. Then, U_{join} obtains a shared secret key UK_{join} with SKDC. Subsequently, SKDC conducts **JoKeyUpdate** algorithm to update the group key MT_K, e_M, and generates a new slave token *ST* for the new user U_{join}. It is noticed that the existing users in the joined UG_K can decrypt newly sent messages, encrypted with the new MT_K, using their *ST*s. After that, SKDC runs the **JoKeyDistribute** algorithm to distribute the rekeying message distribution process in the system when a user U_{join} joins UG_K. First, SKDC notifies the KDC about the joining event, and then, SKDC notifies all users subscribed to the same device groups through

Algorithm 3 JoKeyUpdate

Inputs: UG_K information $(ID, MT_K, S_K, r_K, T)$and (e_M, S).
Output: updated MT'_K, S'_K, r'_K, e'_M and S'.
A new user joins the UG_K
1: Find e_i from $S = \{(e_i, d_i) \mid 1 \le \mathbf{i} \le \mathbf{N}\}$ where $f((e_i, d_i)) = 0$
// $e_i^{\{join\}K} = e_i$, **is added to** S_K
2: $S'_K = \{e_i^{1K}, e_i^{2K}, \dots, e_i^{rK}\} + \{e_i^{\{join\}K}\}$
3: $r'_K = r_K + 1$;
4: $e_{M_K} = (MT_K - T)$;
5: $\mathbf{e}'_{M_K} = \mathbf{e}_{M_k} + e_i^{\{join\}K}\ \mathbf{M[i]}\ N[i]$;
6: $MT'_K = (e'_{M_K} + T)$;

a multicast message, to update TEK_j by using a hash function. Consequently, old users update TEK_j to minimize the communication overhead in the system. Hence, the new user cannot access to previous exchanged data. Finally, SKDC sends the updated keys to the new user U_{join} through a unicast message, including his *ST* key.

Suppose a user *U4* wants to get access to *DG1, DG2,* and *DG4,* as shown in Fig.4, meanwhile *U4* needs to get these traffic keys TEK_1, TEK_2, TEK_4. For that, *U4* requests to join *UG1* after being authenticated and authorized. First, SKDC creates a shared secret key *UK4* with *U4;* then, it multicasts a notification based on the identities of user groups subscribed to the same device group to update TEK_1, TEK_2, TEK_4, so that the new user cannot access to previous exchanged data.

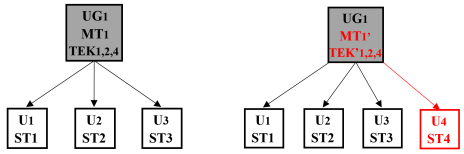


Fig.4. Structure inside UG_1 when U_4 joins

Besides, SKDC updates the group key *MT1'* of *UG1* as mentioned in **JoKeyUpdate** algorithm to protect previous communications between users and SKDC from intruders, and generates a new *ST* for *U4,* while existing users of *UG1* still be able to decrypt data of new group key *MT1'*. Moreover, devices of *DG1, DG2,* and *DG3* update $TEK'_1=h(TEK_1)$, $TEK'_2=h(TEK_2)$, $TEK'_4=h(TEK_4)$. Finally, *ST* and updated TEK'_1, TEK'_2, TEK'_4 keys are sent, in unicast, to the new user *U4* by the SKDC. The protocol steps are given in the **JoKeyDistribute** algorithm.

Algorithm 4 JoKeyDistribute

Inputs: *TEKs, DKs, MT*
Output: new and updated keys *ST, Ui, MT', TEKs', DKs'*
1: ***SKDC*** $\xrightarrow{\text{unicast}}$ ***User i*:** establish a shared secret key with *user i* U_i
2: ***SKDC*** $\xrightarrow{\text{multicast}}$ ***All*:** Notify *KDC*, old users of the joined group and other user groups which subscribed to the same *DG* to update $TEK'=h(TEK)$.
3: ***KDC*** $\xrightarrow{\text{multicast}}$ ***Devices*:** update their key $DK'=h(DK)$
4: ***SKDC*:** update *MT* of this group joined
5: ***SKDC*** $\xrightarrow{\text{unicast}}$ ***User*:** $[ST_i, DKs, TEK]_{UKi}$

ii. When a user leaves a group:

In this phase, assume that a user U_{leave} leaves a group UG_K. Thus, he is not allowed to obtain the exchanged messages after revocation to ensure the forward secrecy. Hence, SKDC conducts the **LeKeUpdate** algorithm to update the group key MT_K, e_M, and user group information. Otherwise, the updating of the master key MT_K is ensured by deleting the *ST* of the leaving user, while the remaining slave tokens are valid to decrypt data of the new MT_K. After that, SKDC runs the algorithm **LeKeyDistribute** to distribute the necessary rekeying message in the whole network when user U_{leave} leaves UG_K. Firstly, a user U_{leave} announces his willing to leave the system to SKDC, which verifies the request and unicasts a message to KDC to signal a leave event. Then, KDC updates all TEK_j to which U_{leave} was subscribed according to the group identity ID_{UG_K} by generating new TEK_j based on the updating method (TEK_j |random processes of KDC), and then, KDC broadcasts the new *TEKs* to SKDCs. At that point, SKDC enforces an access control level for the user group using its ID_{UG}; $\left[TEK_j^{new}, \forall\, j \,|A_{j,b} = 1\ of UG_K\right]$. Thus, according to ID_{UG}, SKDC encrypts the updated TEK_j^{new} using the corresponding *MT* of *UG* and encrypts the results with the master key of SKDC. Consequently, the message is broadcasted to all corresponding users. Notice that only users with a valid *ST* can decrypt the new TEK_j^{new}.

Algorithm 5 LeKeyUpdate

Inputs: $\boldsymbol{UG_K}$ information $(\boldsymbol{ID}, \boldsymbol{MT_K}, \boldsymbol{S_K}, \boldsymbol{r_K}, \boldsymbol{T})$and systen information$(\boldsymbol{e_M}, \boldsymbol{S})$.
Output: updated $\boldsymbol{MT'_K}$, $\boldsymbol{S'_K}$and $\boldsymbol{r'_K}$.
The i[th] user leaves the UG_K
1: $\boldsymbol{f(e_i^{\{leave\}K}) = 0}$
2: $\boldsymbol{e_i^{\{leave\}K}}$ **is revoked from** $\boldsymbol{S_K}$
3: $\boldsymbol{S'_K = \{e_i^{1K}, e_i^{2K}, \dots, e_i^{rK}\} \setminus \{e_i^{\{leave\}K}\}}$
4: $\boldsymbol{r'_K = r_K - 1}$;
5: $\boldsymbol{e'_i = e_i^{\{leave\}K} = 4 \times Random + 1}$;
6: $\boldsymbol{e_{M_K} = (MT_K - T)}$;
7: $\boldsymbol{e'_{M_K} = e_{M_k} - e_i^{\{leave\}K}\, M[i]\, N[i]}$;
8: $\boldsymbol{e'_M = e_M - e_i\, M[i]\, N[i] + e'_i\, M[i]\, N[i]}$;
9: $\boldsymbol{MT'_K = (e'_{M_K} + T)}$;

Suppose a user *U3* leaves the group *UG1,* as shown in Fig.5. Thus, she/he unsubscribes from *DG1, DG2,* and *DG4,* which leads to losing the access privilege to those *DGs.* Since the data of *DG1, DG2,* and *DG4* should not be visible to this user anymore, TEK_1, TEK_2, TEK_4 should be updated to meet the requirements of the forward secrecy [5].

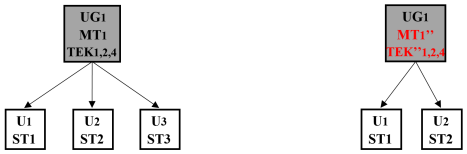


Fig.5. structure inside UG_1 when U_3 leaves

First, SKDC updates *MT1''* of the *UG1,* while all users of this left group still get access to their previous *STs.* Then, KDC broadcasts the newly generated *TEKs* via encrypted message (TEK_i, DK_j'|update methods, i=1,2,4)$_{MK}$ to SKDCs. After, SKDC transmits TEK_1', TEK_2', TEK_4' via an encrypted message with *MT1''* securely to *UG,* based on the user group identity. The remaining users decrypt, with their *STs,* the message to handle the updated information. Finally, devices in groups *DG1, DG2,* and *DG4* get the new *TEKs* keys encrypted with *KEK* and *GK*, sent in multicast by the KDC, to prevent a leaving user from obtaining additional data. The steps are defined in the **LeKeyDistribute** algorithm.

Algorithm 6 LeKeyDistribute

Inputs: *TEKs, DKs, MT*
Output: new generated keys *MT', TEKs', DKs'*
1: SKDC updates MT of the group UG has been left
2: SKDC $\xrightarrow{\text{unicast}}$**KDC:** notify that UG has been left
3: KDC $\xrightarrow{\text{multicast}}$ **SKDCs:** (TEK'| DK')$_{MK}$
4: KDC $\xrightarrow{\text{multicast}}$ **Devices:** (TEK')$_{GK}$
5: SKDC $\xrightarrow{\text{multicast}}$ **user groups UG:** ((TEK')$_{MT}$)$_{MK}$

e. IoT device membership changes (join/leave)

In order to describe the update key process of *DLGKM-AC* during IoT device join/leave events, we consider the case of the device group *DG1,* which publishes data to the user groups *UG1, UG3, UG4, UG6* based on their group identity.

i. When an IoT device joins a group:

Consider the update procedure of an IoT device D_{join} joining a *DG*. Therefore, the KDC runs the **DeJoKeUpdate** algorithm. First, KDC shares a secret key with D_{join} that joins the device group DG_y. Then, KDC updates the necessary part of the LKH tree in which the device resides, multicasts to the existing devices a notification to upgrade the group key *GK*. Finally, KDC sends to D_{join} the PK_t and *TEK* of the DG_y through unicast. Suppose a new device *D4* joining the system. D4 is assigned to the device group *DG1* as shown in Fig.6.

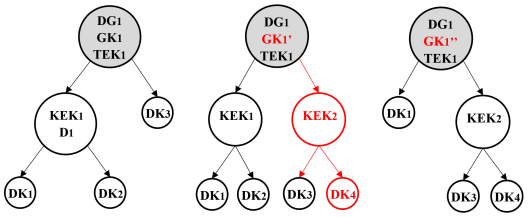


Fig.6. Examples of structure updates for device join/leave.

KDC notifies devices of *DG1* to update *GK'1=h(GK1)* and creates a shared secret key *D4* with device 4 to send necessary information TEK'_1, *GK'1*, *KEK2* encrypted with the secret key of D4 through a unicast communication. Finally, KDC sends *KEK2* to D3. The protocol steps are defined in the **DeJoKeUpdate** Algorithm.

Algorithm 7 DeJoKeUpdate

Inputs: *KEKs, GK*
Output: new and updated keys *DK, D_j, KEKs', GK'*
1: KDC → device Dj: establish a shared secret key with the new device (D_j)
2: KDC $\xrightarrow{\text{multiicast}}$ **old devices in DG:** update group key GK'= h(GK)
3: KDC $\xrightarrow{\text{unicast}}$ **devices:** update KEK' encrypted either by secret keys or shared KEK

ii. When an IoT device leaves a group:

When a device D_{leave} leaves a *DG*, the KDC rearranges the LKH tree structure in the group and runs the **DeLeKeUpdate** algorithm. Thus, KDC multicasts an updated group key *GK*' to the remaining devices encrypted with *KEKs,* which defines the LKH tree of the leaved *DG*. Then, KDC broadcasts a message to announce that D_{leave} is no longer a valid device.

As shown in Fig.6, when device *D2* leaves the group *DG1*, KDC makes a new device group key (*GK'1*|update method*)* and multicasts it to *D1* and *D3*. The protocol steps are defined in the **DeLeKeUpdate** Algorithm.

Algorithm 8 DeLeKeUpdate

Inputs: *KEKs, GK*
Output: new keys *KEKs', GK.'*
1: KDC $\xrightarrow{\text{broadcast}}$ **All:** "leaving device j is no longer available."
2: KDC $\xrightarrow{\text{multicast}}$ **DG:** update GK' and KEK's.

VI. Security Analysis

In this section, we analyze the proposed scheme effectiveness in terms of forward and backward secrecy and resistance to the collusion attack.

A. Forward Security

In this section, we analyze the forward security property of the proposed protocol.

***Theorem 1*:** *The proposed group key management scheme between SKDC and users provides forward security against an adversary. In other words, the revoked user cannot get access to the ongoing communication.*

Proof: Consider the case that the key pair (e_j, d_j) should be revoked when user U_j leaves the group UG_K. The SKDC updates e_M and e_{M_K}. Thus, the master key of UG_K satisfies (1) and (2):

$$P^{e'_{M_K}} \equiv P^{e_i} \bmod (p_i q_i) \quad (1)$$

$$C^{d'_{M_K}} \equiv C^{d_i} \bmod (p_i q_i);\ \forall\, i \in [1, r_K],\ i \neq j \quad (2)$$

At the data source, the plaintext P is encrypted as $P^{e'_{M_K}} \bmod (\prod_{i\neq j=1}^{N} \emptyset(p_i q_i)) = C^*$. After receiving the new ciphertext C^*, each user in the group can decrypt it with its individual private key $C^{*d_i} \bmod (p_i q_i) = P, \forall\, i \neq j$. Although the left user from UG_K knows the old keys (e_j, d_j), he/she cannot obtain the correct plaintext from the ciphertext C^*through the old keys $C^{*d_j} \bmod (p_j q_j) = P^* \neq P$.

Theorem 2: *The proposed group key management scheme between KDC and IoT devices provides forward security against an adversary. In other words, the revoked IoT device cannot get access to the current communication.*

Proof: This theorem is analyzed through the game G_1. Let A_1 be an adversary by colluding with the left IoT device D_j in the device group DG_K. It is worth that A_1obtains all information stored in left IoT device $(DK_j, GK_K, TEK, KEKs)$ and wants to derive the current group key, GK'_K. After the IoT device is revoked, KDC is responsible for updating the LKH tree of DG_k, likewise updating the path key from the revoked leaf node to the root node $\{KEK_i \in PK_t \ of\ D_j\}$ which are used to encrypt and broadcast the GK'_K to the remaining devices. Thus, A_1 cannot decrypt the keying message and get GK'_K. Therefore, our protocol provides forward secrecy in *DG*.

B. Backward Security

In this section, we analyze the backward security property of the proposed protocol.

Theorem 3: *The proposed group key management scheme between SKDC and users provides backward security against an adversary. In other words, the newly joined user cannot get access to previous communications.*

Proof: Suppose a new user $U_j$joining a group UG_K with the key pair (e_j, d_j). The previous data source P is encrypted as $P^{e_{M_K}} \bmod \left(\prod_{i\neq j=1}^{N} \emptyset(p_i q_i)\right) = C$, and the master key of UG_K satisfies $\forall\, i \in [1, r_K]\ and\ i \neq j$:

$P^{e_{M_K}} \equiv P^{e_i} \bmod (p_i q_i), C^{d_{M_K}} \equiv C^{d_i} \bmod (p_i q_i)$;

The SKDC updates e_M and e_{M_K} such that the master key of UG_K satisfies $\forall\, i \in [1, r_K]\ and\ i = j$:

$$P^{e'_{M_K}} \equiv P^{e_i} \bmod (p_i q_i), C^{d'_{M_K}} \equiv C^{d_i} \bmod (p_i q_i);$$

Thus, the user joining the group UG_K with the pairs keys (e_j, d_j), cannot obtain the correct previous plaintext from

the ciphertext C through the new keys because $C^{d_j} \bmod (p_j q_j) = P^* \neq P$.

Theorem 4*: The proposed group key management scheme between KDC and IoT device provides backward security against an adversary. In other words, the joined IoT device cannot get access to the previous communication.*

Proof: Suppose a new IoT device D_j joins the device group DG_K and has $(DK_j, GK_K, TEK, KEKs)$ new keys. Thus, the group key *GK* cannot be derived by D_j. Meanwhile, knowing the secret key and the new GK'_K, the newly joined device cannot learn anything about the previous group keys.

C. Resistance to Collusion Attack

In this section, we analyze the resistance to the collusion attack using the Random Oracle Model (ROM) standard [25].

Theorem 5: *The proposed GKM is collusion resistance secure.*

Proof: Let G^{cr} be the adversarial game for collusion resistance. This game is played between two adversaries; one acts as the challenger C^{cr} who interacts with the adversary A^{cr} trying to win C^{cr}. It is worth noting that C^{cr} has the ability to simulate all the oracles O^{join}, O^{leave}, $O^{ciphertxet}$ and $O^{decrypt}$ functions and output signed messages as a real signer. G^{cr} consists of the following phases:

Setup: C^{cr} runs the **MTokenGen** algorithm for a random choice of *ID* by A^{cr}. Rekeying operation is simulated after that, and the timeline is started (t=0).

Queries: It is allowed to query the oracle O^{join}, O^{leave}, $O^{ciphertxet}$ and $O^{decrypt}$ to control group dynamicity.

Challenge: A^{cr} issues one challenge query to C^{cr} at time $t_{challenge}$ (which is the choice of the A^{cr}). Before responding to the challenge, C^{cr} retrieves the set challenge $S_{challenge}$ from the list L_s, and forms the list of leaving members L_g, for all $ID \notin S_{challenge}$. Then, for each identity $ID \notin S_{challenge}$, C^{cr} issues the query $O^{extract}(ID)$ to obtain S_{ID}. Besides, C^{cr} encrypts (TEK, ST, $S_{challenge}$) to get ($A_{i,b}$', TEK'), where $A_{i,b}$' defines the authorized receivers of TEK challenged with C^{cr}. After, C^{cr} chooses a bit $b \in \{0, 1\}$ at random and sets K_b to TEK' and K_{b-1} to a random TEK from the key space. Finally, it challenges with ($A_{i,b}$', K_0, K_1).

Guess: A^{cr} *outputs a bit b'* $\in \{0,1\}$ as its guess. C^{cr} passes on b' as its guess to A^{cr}.

The adversary advantage in winning the game is defined as $Adv_{GKM}^{cr} = \left| pr[b' = b] - \frac{1}{2} \right|$; hence, we can see that the advantage that A^{cr} breaks the collision resistance of GKM is the same that C^{cr} breaks chosen-ciphertext attack (CCA), meanwhile, breaks the encrypted messages (Ex. AES). Thus, if there exists no adversary who can break CCA security with non-negligible advantage, then there cannot be any adversary A^{cr}, who can break the collision resistance of GKM with non-negligible probability.

VII. Performance Analysis and Evaluation

In this section, we analyze the performance of the proposed scheme in terms of storage overhead, computation overhead, and communication overhead. Then, we compare the results with existing methods. We also discuss the time complexity to renew the master token and revoke the slave token that we proposed for the communication with users in the same group.

A. Performance Analysis

This subsection presents the performance analysis of *DLGKM-AC*. In order to guarantee generality, we assume that IoT devices are equally distributed in each device group, and the LKH structures are all balanced binary trees.

a. Storage overhead

Storage overhead can be considered as the memory capacity required to maintain the keys. In this section, the storage overhead is formulated, both at each user in UG_x and each device in DG_y. A user belonging to UG_x has a slave token *ST*, which is an asymmetric key *AK*, and as many symmetric keys (*SK) TEKs* as the number of *DGs* for which UG_x is subscribed. Moreover, it has his secret key shared with SKDC. We can calculate the storage of keys for each user in UG_x as follow using Eq (5):

$$SO_{U \in UG_x} = AK + \left(\sum_{i=1}^{M} A_{i,b} + 1\right) SK \quad (5)$$

The analysis of a single *d*-degree key tree accommodating *n* member requires the tree depth denoted by $f_d(n)$. It is known that $f_d(n)$ is either L_0 or L_0 +1, where $L_0 = log_d(n)$. The authors of [11] made useful inequality (6) in order to analyze the storage overhead for key trees:

$$E[f_d(n)] \leq E[log_d(n)] + 1 \leq log_d E[n] + 1, \quad (6)$$

where the expectation, $E[.]$, is taken over the distribution of *n* devices and the length of the branches on the key trees.

A device belonging to DG_y, containing *n* devices, has a traffic key TEK, a group device key *GK,* and as many symmetric keys, including the KEKs and the individual key, as the length of the branch. As we consider that devices are distributed in binary trees, we can calculate the number of keys for each device in DG_y using the following Eq (7):

$$SO_{D \in DG_y} = (\log_2 n + 3) \times SK \quad (7)$$

b. Computation overhead

The computational overhead can be measured as the total time consumption for encryption and decryption cost and processing requirement. This action takes place on the server, user as well as on the device sides, after each member (user/ IoT device) joining or leaving actions. We explain the different computation costs as follows:

i. When a user joins a subgroup x UG_x:

The SKDC assigns a slave token to U_{join} and updates the master token of UG_x and its master key. The new user needs one symmetric decryption to gain the slave token *ST*, new *TEK*, and all *DKs* of the devices in the device groups to which they are subscribed. An existing user needs to do one hash function to update *TEK*. Finally, the devices need to perform one hash function to update their *TEK* and another hash function to derive their new device keys *DK*.

ii. When a user leaves a subgroup x UG_x:

The SKDC needs to update the master token of UG_x and its master key in order to send *TEK* securely to users. The remaining user groups need to perform one asymmetric decryption and one symmetric decryption to gain the update information. Devices to which user groups are subscribed, need to do one symmetric decryption to obtain the update information *TEK* and *DK*.

iii. When a device joins a device group y DG_y:

The old devices of the left device group require to do one hash function to update the device group key GK_y. Since the GKM scheme of a group device based on LKH, the structure will change, and some devices need to decrypt *O(log(n)) KEK* update messages. While the new device needs only to decrypt one message sent from KDC to obtain *KEKs*. Users subscribed to the group joined by the new device require to decrypt the message sent by KDC to gain the new device key.

iv. When a device leaves a device group:

The remaining devices execute one symmetric decryption to gain the new group key. While users subscribed to the leaving groups, they do not need to perform extra computation.

c. Communication overhead

This subsection evaluates the communication overhead of the new *DLGKM-AC* for the IoT environment, as shown in Table IV:

TABLE. IV: COMMUNICATION OVERHEAD

Events	Communication cost
User leave's event	SKDC broadcasts the new *TEK* and *DK* to subgroups KDC sends *log(n)* messages to devices
User join's event	SKDC unicasts a message to the new user SKDC notifies all users to update *TEK*
Device join's	KDC unicasts a message to the new device KDC broadcasts the subscribers with the new *DK*
Device leave's event	KDC notifies the subscribers that the leaving device is no longer available. KDC multicasts *log(n)* messages for the remaining devices to update group key.

B. Performance Evaluation

In this section, we present experimental results for the *DLGKM-AC* scheme developed on MATLAB. We evaluate DLGKM-AC performances in terms of storage, computation, and communication overheads caused by rekeying. The rekeying transmission overhead corresponds to the additional signaling load after each join/leave event. We compare the new proposed DLGKM-AC scheme with two other key management solutions designed for access control between subscribers and publisher; a centralized scheme that support groups of publishers(GroupIT[13]) and a decentralized scheme does not support groups of publishers (SMGKM [17]).

a. Storage overhead

The storage overhead is formulated at both sides, user, and IoT devices. To achieve a comparable security strength, we assume the symmetric encryption/decryption key length to AES-256 bits, the ECC-512 decryption key length to 512 bits. To calculate the storage overhead at the user, we consider two cases: case (1), where we vary the number of publishers *DGs* to which users are subscribed while fixing 20 users per *UG*, and case (2) where we change the number of users in each *UG* and consider the number of *DGs* settled to 4 and the number of devices fixed to 20 per *DG*.

Case (1): Through Fig.7, we notice that, unlike existing solutions such as GroupIT [13] and SMGKM [17], our scheme is less affected by the increase of DGs number to which users are subscribed. This can be explained by the fact that our scheme uses a decentralized architecture and divides IoT devices into groups which can alleviate the storage overhead compared to [13] and [17].

Case (2): Fig.8 shows that, in [13] [17] schemes, when the number of users increases, the storage on users increases too. Thus, the larger the number of users in each UG, the more these schemes incur users' storage overhead. In our scheme, the users are not affected by the number of users in their *UG*. This is explained by the use of the proposed MTE-based scheme for grouping users, which is not sensitive to the number of users in each UG. Hence, this can reduce the storage overhead per-user more efficiently.

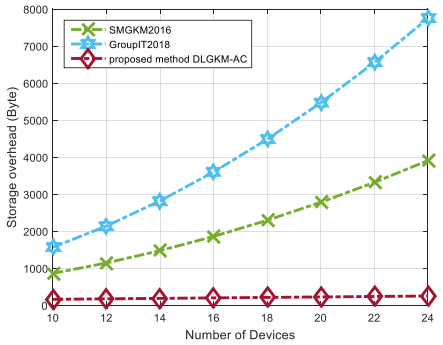


Fig.7: Users' storage overhead while varying the number of devices

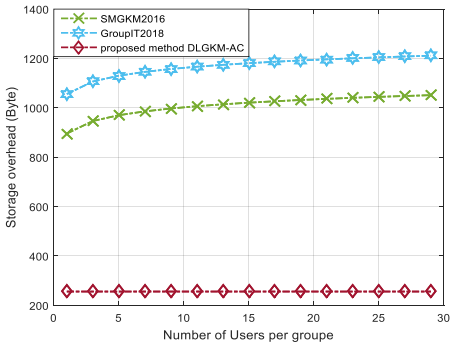


Fig. 8: Users' storage overhead while varying the number of users

The storage on devices is not affected neither by the number of users nor by the number of devices in other different *DGs* because devices are considered just as data publishers. We can notice that devices are only affected by the number of devices of their group. Fig.9 shows that [13] and our scheme have mainly the same storage. Otherwise, [17] does not hold the notion of grouping the devices (publishers). Thus, storage on devices (publisher) is not affected by the number of devices in the same group.

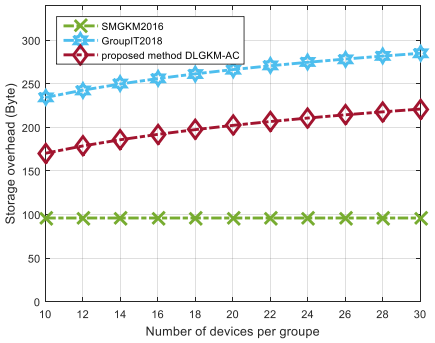


Fig.9: Devices storage overhead

b. Computation overhead

To evaluate the computation overhead, we simulate the cryptographic operations with Miracle Library. It is a cryptographic library designed for use in constrained environments in terms of computational power [26]. All simulations are implemented on a computer with the following features: an Intel i5-4200 CPU@ 2.5 GH with a physical memory of 8 GB; and Ubuntu 12.04 OS over VMware workstation 15. We provide the time cost for different cryptographic operations. As a result, we define T_h = 2,445µs be the time for one hashing operation using SHA-256 function on a 64-byte block. Then, T_{Enc}=T_{Dec}=2,7µs be respectively the time for one encryption/decryption operation using symmetric cryptography AES-256 encryption on a 64-bytes, and T_{ECC}=365,63µs represents the time for one elliptic curve cryptographic operation.

Since our protocol is designed for a dynamic IoT environment, the computational cost is measured based on leave and join operations of both users and devices. In the practical scenario, users frequently join or leave (subscribe or unsubscribe) *UG*. Hence, we focus on user leave/join events in the following subsection:

i. Computation overhead when a user leaves a UG:

Assume a user *U* of UG_K leaves UG_K. Thus, *it* is not allowed to obtain the rekeying message. Therefore, this operation is triggered by the server and transmitted to users and devices to ensure forward secrecy. We compare the computation cost as follows:

- Computation cost on the remaining users' side: We consider two cases. In the first case, we vary the number of publishers *DGs* to which users are subscribed while fixing 20 users per *UG*. Fig.10 shows that our scheme is less affected by the number of DGs to which users are subscribed than solutions developed in [13] [17]. The result explains that a decentralized architecture reduces the computation overhead resulting after a leave event. Otherwise, using SKDCs offload the computation overhead of updating keys in such an IoT environment.

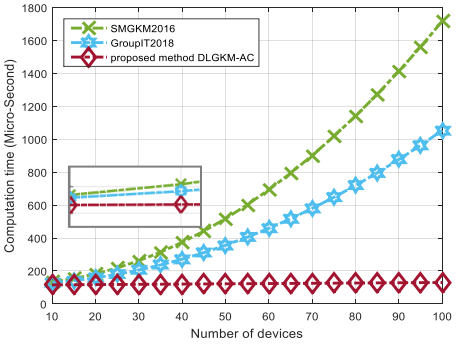


Fig.10: Remaining user computation overhead varying devices' number (leave)

In the second case, we modify the number of users in each *UG* and consider the number of *DGs* fixed to 4 and the number of devices fixed to 20 per *DG*. Fig.11 shows that, unlike our scheme, more the number of users in each UG is large, more the computation overhead is high in [13] [17] schemes. The proposed MTE algorithm for managing communication within user groups is one reason that explains why our proposed system has low computational cost while having a high number of users in UG.

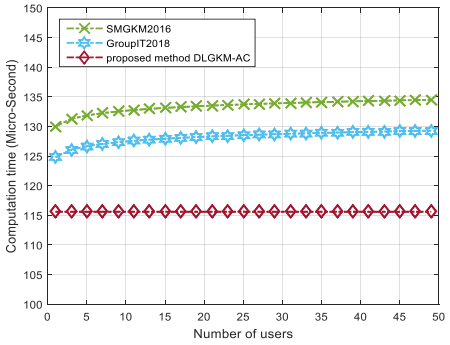


Fig. 11: Remaining user computation overhead varying users' number (leave)

- Computation on the server-side: The group key updating time of SKDC was considered to prove the efficiency of our group key updating scheme based on CRT. Fig.12 shows that, compared to traditional MKE, our solution consumes less time for the key updating when a user is revoked.

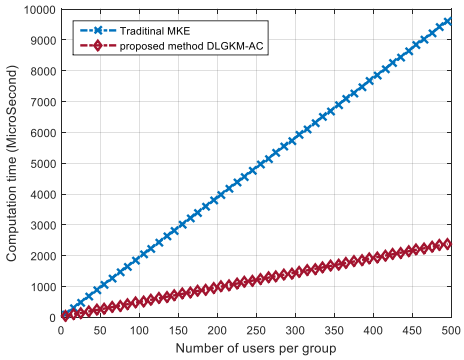


Fig.12: Server time update on the leaving event

ii. Computation overhead when a user joins a UG:

Assume a user *U* joining UG_K. *U* should not be allowed to access previous communications. Thus, the rekeying operation is triggered. Hence, we compare the updating overhead when a user joins a group as follows:

- Computation cost on old users' side: We also consider two cases. In the first case, we vary the number of publishers DGs to which users of UG are subscribed while fixing 20 users per UG. Fig.13 presents a comparison of the computation cost for old users of the joining user group *UG*. We note that our scheme is less affected by the number of DGs to which users are subscribed, than [13] [17] schemes. This outcome is explained by using subgroup controllers SKDCs to manage the update of keys and thus reducing computation for end-users.

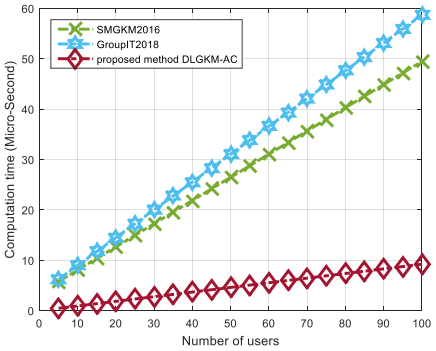


Fig.13: Old user computation overhead varying the devices' number (join)

In the second case, we vary the number of users in each *UG* and consider 4 *DGs* and 20 devices per *DG*. Fig.14 depicts the comparison of the update overhead when varying the number of users per group. We notice that our scheme is not affected by the number of users in the group as key management for user groups is based on MTE.

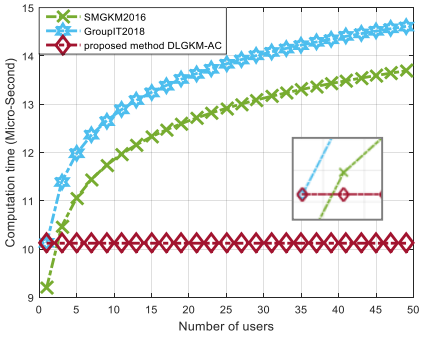


Fig.14: Old user computation overhead varying the users' number (join)

- Computation cost on new users' side: Fig.15 presents the update overhead on a new user who joins a user group. Our scheme and [17] present a negligible computation overhead compared to [13]. In fact, the new user needs only to decrypt received messages to get information. While in [13], a new user needs to compute the device keys to which he/she is subscribed.

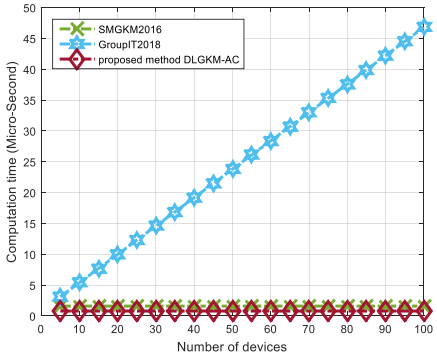


Fig.15: New user computation overhead varying devices

- Computation cost on the server side: Fig.16 shows the average time to update keys when there is a user joining operation. More specifically, the time needed to execute the **JoKeyUpdate** algorithm when varying the number of users per group. Unlike the traditional MKE, the execution time of our scheme increases slowly with the increase of the number of users per group.

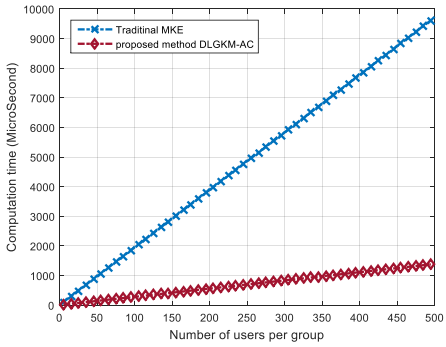


Fig.16: Server time update on the joining event

iii. Computation overhead when a device joins/leaves a DG

In this subsection, we compare the overhead update triggered by device join and leave operations as follows:

- The computation cost when a device joins DG: Fig.17 depicts the update overhead when a device joins a device group. The overhead is measured on both the existing devices and the new device sides. We notice that the computation cost of the new device in our scheme is less than GroupIT. In contrast, the existing devices in *DG* present the same cost to get the updated keys. However, the cost in SMGKM is fixed as they do not consider grouping devices.

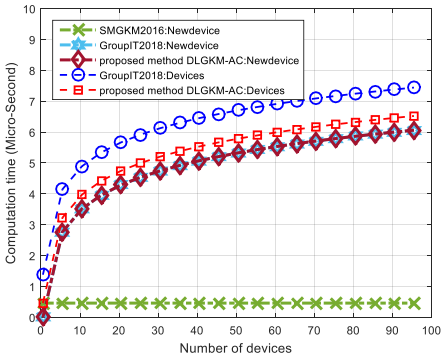


Fig.17: Computation overhead: device join

- The computation cost when a device leaves DG: Fig.18 depicts the update overhead when a device leaves a DG. The cost is measured on both the remaining devices of the group and the users subscribed to the DG. In our scheme, the users' computation cost is not affected by the operation of the leaving device, while, in [17], it increases with the increase in the number of devices. The advantage of grouping devices explains this result. Moreover, remaining devices in our scheme have less computation cost compared to [13] because using a decentralized scheme reduces the reload on KDC, and thus, KDC reduces the reload on devices.

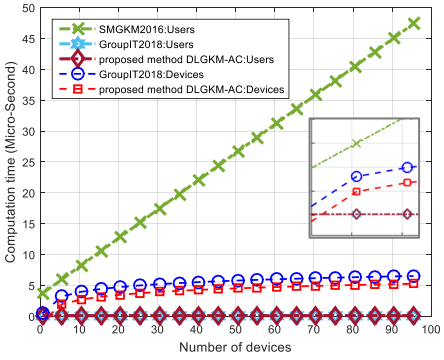


Fig.18: Computation overhead: device leave

c. Communication overhead

To evaluate the communication overhead of the new proposed *DLGKM-AC* for the IoT environment**,** we analyze the number of updating keys messages transmitted when a user leaves a *UG*. Fig.19 shows that [17] scheme causes many rekeying messages when a user leaves a user group, and the number of devices is high. Therefore, [13] incurs much less communication overhead than [17] but still causes little more communication overhead compared to our scheme, which is explained by using device groups and introducing MKE for grouping users.

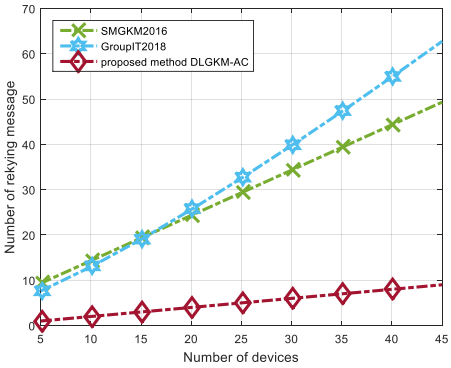


Fig. 19: Communication overhead: the leave event

VIII. Conclusion

A novel decentralized lightweight group key management for access control in a dynamic IoT environment named *DLGKM-AC* has been introduced in this paper. A hierarchical architecture is adopted using one KDC, for managing group keys and broadcasting update messages, and several SKDCs, for handling direct communication links between devices and users. Besides, a new master token encryption algorithm has been introduced in order to ensure members' independence in highly dynamic group communication. In *DLGKM-AC*, mobility is smoothly handled as we provide the backward and the forward secrecy with fewer rekeying operations. Furthermore, our protocol mitigates the 1-affects-n issue. Indeed, users can always get access to data even if one SKDC is affected. Extensive security analysis covering a wide range of desired security properties has also been provided. Additionally, performance analyses shows that our proposed scheme offers better performances by reducing storage, communication, and computation overheads. Finally, adopting a decentralized architecture increases scalability and reduces overhead for resource-constrained devices. As future work, to put it into practice via a proof-of-concept, we are already planning to deploy our architecture in a real-world setting, in the context of the European project PARFAIT [28], by constructing a physical network comprising a set of IoT devices and smartphones as users.

Acknowledgment

This work is achieved as part of the European project ITEA PARFAIT [28], which is partially funded by FEDER (European Regional Development Fund), BPIFRANCE, and the BFC region (Bourgogne-Franche-Comté).

Acronym Table

Acronyms	Descriptions
AES	Asymmetric Encryption Standard
ABE	Attribute-Based Encryption
CCA	Chosen-Ciphertext Attack
CRT	Chinese Remainder Theorem
DeJoKeUpdate	Device Join Key Update
DeLeKeUpdat	Device Leave Key Update
DG	Device Group
DK	Device Key
DLGKM-AC	Decentralized Lightweight Group Key Management for Access Control
ECC	Elliptic Curve Cryptographic
GCRT	Group Chinese Remainder Theorem
GK	Group Key
GKM	Group Key Management
IoT	Internet of Thing
JoKeyUpdate	Join Key Update
JoKeyDistribute	Join Key Distribute
KDC	Key Distribution Center
KEK	Key Encryption Keys
LeKeUpdate	Leave Key Update
LeKeDistribute	Leave Key Distribute
LKH	Logical Key Hierarchy
MKE	Master Key Encryption
MkeyGen	Master Key Generation
MT	Master Token
MTE	Master Token Encryption
MTokenGen	Master Token Generation
MQTT	Message Queuing Telemetry Transport
OFT	One-way Function Tree
PKt	Path key
ROM	Random Oracle Model
SHA	Secure Hash Algorithm
SKDC	Sub-Key Distribution Center
ST	Slave Token
TEK	Traffic Encryption Key
UG	User Group
WBAN	Wireless Body Area Network
WSN	Wireless Sensor Network

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*, vol. 1, no. 1, pp. 22-32, Feb. 2014.

[2] https://www.helpnetsecurity.com/2019/06/21/connected-iot-devices-forecast/.

[3] M. Dammak, O. R. M. Boudia, M. A. Messous, S. M. Senouci, and C. Gransart, "Token-Based Lightweight Authentication to Secure IoT Networks," 2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC), Las Vegas, NV, USA, 2019, pp.1-4.

[4] Panagiotis I. Radoglou Grammatikis, Panagiotis G. Sarigiannidis, Ioannis D. Moscholios, Securing the Internet of Things: Challenges, threats, and solutions, Internet of Things, Volume 5, 2019.

[5] A. Banks and R. Gupta, "MQTT version 3.1. 1," OASIS standard, 2014.

[6] AlMajed, H.N.; AlMogren, A.S. Simple and Effective Secure Group Communications in Dynamic Wireless Sensor Networks. *Sensors* 2019, *19*,

[7] P. Porambage, A. Braeken, C. Schmitt, A. Gurtov, M. Ylianttila, and B.Stiller, "Group key establishment for enabling secure multicast communication in wireless sensor networks deployed for IoT applications," IEEE Access, vol. 3, pp. 1503–1511, 2015.

[8] A. Mehdizadeh, F. Hashim, and M. Othman, "Lightweight decentralized multicast–unicast key management method in wireless ipv6 networks," Journal of Network and Computer Applications, vol. 42, 2014.

[9] Zhu, B.; Susilo, W.; Qin, J.; Guo, F.; Zhao, Z.; Ma, J. A Secure and Efficient Data Sharing and Searching Scheme in Wireless Sensor Networks. *Sensors*, 2019, *19*, 2583.

[10] Tan, H.; Chung, I. A Secure and Efficient Group Key Management Protocol with Cooperative Sensor Association in WBANs. Sensors 2018, 18, 3930

[11] M.-H. Park, Y.-H. Park, H.-Y. Jeong and S.-W. Seo, "Key management for multiple multicast groups in wireless networks," IEEE Transactions on Mobile Computing, vol. 12, no. 9, pp. 1712–1723, 2013.

[12] Cheikhrouhou O. Secure Group Communication in Wireless Sensor Networks: A survey. Journal of Network and Computer Applications.

[13] Y. Kung and H. Hsiao, "GroupIt: Lightweight Group Key Management for Dynamic IoT Environments," in IEEE Internet of Things Journal, vol. 5, no. 6, pp. 5155-5165, Dec. 2018.

[14] M. R. Abdmeziem, D. Tandjaoui, and I. Romdhani, "A decentralized batch-based group key management protocol for mobile internet of things (dbgk)," 2015 IEEE International Conference on Computer and Information Technology.

[15] H. Harney and E. Harder, "Logical key hierarchy protocol," Internet draft, Tech. Rep., 1999.

[16] D. Balenson, D. McGrew, and A. Sherman, "Key management for large dynamic groups: One-way function trees and amortized initialization,"

[17] T. T. Mapoka, S. J. Shepherd and R. A. Abd-Alhameed, "A New Multiple Service Key Management Scheme for Secure Wireless Mobile Multicast," in *IEEE Transactions on Mobile Computing*, vol. 14, no. 8, pp. 1545-1559, 1 Aug. 2015.

[18] Zhong, H., Luo, W., and Cui, J. (2017) Multiple multicast group key management for the Internet of People. Concurrency Computat.: Pract. Exper. 29:e3817, doi: 10.1002/cpe.3817.

[19] I.-C. Tsai, C.-M. Yu, H. Yokota, and S.-Y. Kuo, “Key management in internet of things via kronecker product,” in Dependable Computing (PRDC), 2017 IEEE 22nd Pacific Rim International Symposium on. IEEE, 2017.

[20] W. Ding *et al*., "An Extended Framework of Privacy-Preserving Computation with Flexible Access Control," in *IEEE Transactions on Network and Service Management*. TNSM.2019.

[21] M. Nabeel, N. Shang and E. Bertino, "Privacy Preserving Policy-Based Content Sharing in Public Clouds," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 11, pp. 2602-2614, Nov. 2013.

[22] X. Wang, J. Zhang, E. M. Schooler, and M. Ion, “Performance Evaluation of Attribute-Based Encryption: Toward Data Privacy in the IoT,” in IEEE ICC, 2014.

[23] S. Sciancalepore, A. Capossele, G. Piro, G. Boggia, and G. Bianchi, “Key management protocol with implicit certificates for iot systems,” in Proceedings of the 2015 Workshop on IoT challenges in Mobile and Industrial Systems. ACM, 2015, pp. 37–42.

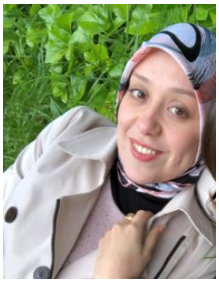
[24] Y. Tseng, C. Fan and C. Wu, "FGAC-NDN: Fine-Grained Access Control for Named Data Networks," in *IEEE Transactions on Network and Service Management*, vol. 16, no. 1, pp. 143-152, March 2019.

[25] Karuturi, N. N., Gopalakrishnan, R., Srinivasan, R., & Rangan, C. P. (2008). Foundations of Group Key Management-Framework, Security Model and a Generic Construction. IACR Cryptology EPrint.

[26] Abdmeziem M.R., Charoy F. (2018) Fault-Tolerant and Scalable Key Management Protocol for IoT-Based Collaborative Groups. In: Lin X., Ghorbani A., Ren K., Zhu S., Zhang A. (eds) Security and Privacy in Communication Networks. SecureComm 2017.Springer

[27] https://github.com/miracl/MIRACL

[28] http://www.itea3-parfait.com/

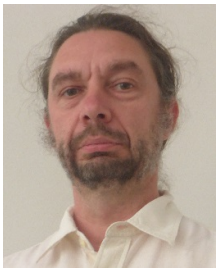


Maissa DAMMAK received the master engineer degrees in telecommunication from SUPCOM (High School of Communication in Tunis), Tunis, in 2016. She was an R&D researcher in SAGEMCOM during 2017. She is currently a Ph.D. student in computer science and cybersecurity at laboratory DRIVE EA 1859 collocated in ISAT Nevers, France, and part of the University of Burgundy. She has currently involved in the European R&D projects FUI PARFAIT. Her research interests include the security of the Internet of Things (IoT), lightweight authentication protocols and applied cryptography, decentralized architecture design, key management for access control. Blockchain applications with the Internet of Things.

Sidi-Mohammed Senouci received his Ph.D. in Computer Science in 2003 from the University of Paris 6 and his HDR from INP Toulouse, France. Since September 2010, he is professor at ISAT, a major French post-graduate school located in Nevers, France, and part of the University of Bourgogne. Since 2017, he is director the laboratory DRIVE EA 1859 collocated in ISAT. He participates to several national and European-wide research projects. Among them FP7 FOTsis, ITEA CarCoDe, ITEA FUSE-IT and FUI PARFAIT. He holds 7 international patents on these topics and published his work in major IEEE conferences and renowned journals. He was co-chair of AHSN Symposium in IEEE Globecom 2011, co-chair of NGN Symposium in IEEE ICC of 2012 and 2017, co-chair VCT Symposium in IEEE WCMC2010, and TPC co-chair of VehiCom2009 Workshop. He was the Chair of IEEE ComSoc IIN Technical Committee, TCIIN (2014-2016). He was the guest editor in Ad-hoc Networks Journal (Elsevier) in 2018, guest editor “IEEE-Access Special Section” in 2018, guest editor of the UBICC journal on Ubiquitous Roads, guest editor of French journal REE in 2014. He is also a Member of IEEE and the Communications Society and Expert Senior of the French society SEE (Society of Electricity and Electronics).

Dr. Mohamed Ayoub MESSOUS is currently a Senior Research Engineer/Post-Doc at the DRIVE Research lab at the University of Burgundy (France). He has more than 9 years of R&D experience in the fields of Distributed & Mobile Computing, Computer Network and Cyber-Physical Systems. He has been or currently is involved in different European R&D projects (ITEA3 PARFAIT, ITEA2 FuseIT and ITEA2 CarCode) and several academic research projects (University of Klagenfurt, University of Burgundy, National Center for the Development of Advanced Technologies and University of Blida). He holds a Ph.D. (2017) in Computer Science from the University of Burgundy (France) and a MAGISTÈRE degree (2014) in Distributed & Mobile Computing and a Diplôme Ingénieur d’État (2010) in Computer science (Major: Artificial Intelligence) from the University of Blida (Algeria).

Mohamed Elhoucine Elhdhili earned his engineering and master's degrees in computer science at the National School of Computer Sciences (ENSI), University of Manouba, Tunisia. Then, he received his PhD degree from the same school. Since 2018, he has held the position of assistant professor at the National School of Computer Sciences. Previously, he worked as an assistant professor at Higher Institute of Informatics, Ariana, Tunisia. He has been a researcher at CRISTAL Laboratory at ENSI since 2003. His research focuses on issues related to wireless networks: Security, Trust, Privacy, Key Management, and quality of service (QoS).



Christophe Gransart received the Ph.D. degree from the University of Lille, Villeneuve d’Ascq, France, in 1995. He is currently a Senior Researcher with the French Institute of Science and Technology for Transport, Development, and Networks, Villeneuve d’Ascq, with 15 years’ experience of participating in industrial and academic research projects dealing with distributed systems and middleware for trans- portation systems, vehicle-to-vehicle and vehicle- to-infrastructure communications, adaptive middleware, and cybersecurity. His main research interests include computer science, distributed architecture design, and middleware expertise. Dr. Gransart was involved in various National and European projects. He has also participated to Sixth Framework Programme (FP6), Seventh Framework Programme (FP7), Horizon 2020 (H2020), and Shift2Rail programs.