GROUPIT: Lightweight Group Key Management for Dynamic IoT Environments

Yi-Hsuan Kung and Hsu-Chun Hsiao, *Graduate Student Member, IEEE*

***Abstract*—With the proliferation of Internet of Things (IoT) devices that collect sensitive data, access control is more crucial than ever to safeguard IoT data from unauthorized use. To enforce access control policies without trusted online entity, one promising approach is to maintain a group key shared between a device and its current subscribers, such that the device can encrypt its data and only the subscribers can decrypt it. However, prior group key management (GKM) schemes fail to efficiently address new challenges introduced by the massive scale of IoT devices, dynamic memberships of users, and changes in the number of devices. This paper explores efficient GKM to accommodate multiple devices (in addition to multiple users) and to handle frequent membership and device number changes. Inspired by the observation that devices with similar functionalities often have similar access permissions, we propose a two-tier GKM architecture called GROUPIT, in which each device is assigned to one of many predefined groups, and key management is performed within each group as well as between groups to improve efficiency. Despite being conceptually simple, GROUPIT addresses technical challenges including: 1) preventing a malicious device from obtaining extra information about other devices in the same group and 2) ensuring forward/backward secrecy and preventing collusion attacks when gluing two existing GKMs together. The probability of a successful collusion attack quickly drops to 0.3% after five membership changes even in a small device group (e.g., 8). This paper provides both theoretical analysis and a proof-of-concept implementation based on Alljoyn, an opensource IoT communication framework to demonstrate the feasibility of GROUPIT.**

***Index Terms*—Dynamic membership, group key management (GKM), Internet of Things (IoT), multiple devices and owners.**

I. INTRODUCTION

INTERNET-OF-THINGS (IoT) devices are gradually becoming a large part of people's daily lives. IoT devices are network-connected physical devices with a variety of forms and functionalities; one important functionality being data collection. Devices such as motion sensors, heartbeat sensors and IP cameras collect sensitive data with privacy concerns. Most IoT devices have limited computational power, preventing them from efficiently performing cryptographic operations. Moreover, users often fail to conduct proper setup (e.g., changing default passwords), thus making IoT devices vulnerable to attacks.

Attacks targeting IoT devices are on the rise. Mirai malware [1], for example, compromised tens of thousands of IoT devices using only 61 username–password pairs. These attacks pose severe threats not only to cyber security but also human safety, because data collected by IoT devices are often more sensitive and private than traditional machines.

To safeguard IoT data from unauthorized access, one promising approach is to maintain a group key shared between a device and its current subscribers, such that the device can encrypt its data and only the subscribers can decrypt them. Such encryption-based access control is favorable in IoT, as it requires no online trusted entity and can work with the publish-subscribe messaging model, such as MQTT [5]. In other words, the larger problem of enforcing access control is reduced to the problem of group key management (GKM), including key issuance, updates, and revocation. Although GKM is well-studied in traditional settings [9], [15], efficient GKM in *dynamic IoT* environments[1] remains an unsolved challenge. In this paper, the term *dynamic* refers to the varying membership changes between users and devices. A straightforward solution that runs one GKM instance for each device suffers from linear overhead, which is unsatisfactory considering the scale of IoT. Existing GKM schemes that attempt to efficiently accommodate multiple devices cannot be efficiently updated when the number of devices changes dynamically [16]. In addition, since many IoT devices have limited resources, existing schemes that heavily rely on asymmetric cryptography (e.g., attribute-based encryption) are unsuitable.

This paper explores lightweight GKM for dynamic IoT environments. Consider the following scenario. A hotel uses key cards to control guests' access permissions in different rooms and to control the usage of different facilities according to room classes. Whenever a guest checks in or out, or new equipment is installed in some facilities, the hotel control center needs to update the shared keys among the users and devices. When a guest checks out and the room becomes vacant, the devices inside the room should stop sending the room's information and receiving information from other devices. One naive solution here is that each device constructs its own GKM

Manuscript received March 2, 2018; revised May 16, 2018; accepted May 17, 2018. Date of publication May 24, 2018; date of current version January 16, 2019. This work was supported in part by the Taiwan Information Security Center, Academia Sinica, and in part by the Ministry of Science and Technology of Taiwan under Grant MOST 106-3114-E-011-003 and Grant 107-2636-E-002-005. *(Corresponding author: Hsu-Chun Hsiao.)*

The authors are with the Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan (e-mail: r04922019@ntu.edu.tw; hchsiao@csie.ntu.edu.tw).

Digital Object Identifier 10.1109/JIOT.2018.2840321

[1] In a dynamic IoT environment, both users and devices can join or leave the system at any moment, the number of users and devices can potentially be high (namely, hundreds of users and devices in the region), and the memberships (which user subscribes to which device) can vary over time.

structure and handles its subscribers, which is nevertheless inefficient regarding storage and transmission costs.

To support a large number of IoT devices, as well as frequent membership and device number changes at a minimum cost, we propose GROUPIT, a two-tier GKM architecture in which similar devices are grouped together for improved efficiency. This architecture is inspired by the observation that devices with similar functionalities often have similar access permissions [13].[2] Thus, users can subscribe to groups of devices, rather than one at a time, without sacrificing much flexibility. A major technical challenge of this architecture is to ensure that devices in the same group own individual secrets, such that a device cannot access other devices' data in the same group without authorization. In our construction, each device in the same group can encrypt data by deriving its own device key based on a shared traffic encryption key in the group, while newly subscribed users can also generate the same device keys after a secure update procedure. Note that the concept of this two-tier GKM architecture can be flexibly applied to combine other existing GKM schemes. In this paper, we describe in detail the combination of logical key hierarchy (LKH) [9] and a CRT-based scheme [16], because their combination is easy to understand and yields good performance.

Our analysis shows that the proposed method can provide data security in a dynamic environment with a decent update overhead compared with existing methods. When there are 16 users subscribing to 50 devices, our method reduces a user's key storage overhead to 29% when all devices are grouped together, 30.3% when they are divided into four groups (according to security-based categorization proposed by Kim *et al.* [13]), and 33% when divided into ten groups compared with the traditional LKH scheme. Compared with the master key encryption (MKE) GKM scheme, we also eliminate users' large computational overhead induced when the number of devices changes from $O(m)$ to $O(1)$, where m is the total number of devices. Also, we demonstrate the feasibility of our proposed method in our implementation based on Alljoyn, an open source IoT framework designed for data transmission.

II. BACKGROUND AND RELATED WORK

In this section, we briefly introduce the important background mechanics used in our method, including LKH for efficient key management, and the Chinese remainder theorem (CRT) for managing multiple groups.

A. Logical Key Hierarchy

Since an IoT device can be subscribed to by multiple users, it would be more efficient if the device and all its users shared a group key for encryption. The group key also needs to be efficiently updated when new users join or old users leave to ensure forward and backward secrecy [14]. Rafaeli and Hutchison [18] provided a thorough survey of GKM schemes. One of the most common is LKH [9], which represents users as leaf nodes in a tree structure. The LKH structure reduces communication costs by associating multiple key-encryption keys with each user. The update communication requires $O(\log n)$ multicasts for n users.

1) LKH Structure: In LKH, users are located at the leaf nodes of a tree. All users share a group key GK, located at the root node of the tree. Each user has a unique secret key shared with a central control center called a key distribution center (KDC). Each internal node in the tree is associated with a key encryption key (KEK), known by the users in the leaf nodes of the subtree rooted at the internal node. KEKs are used later to efficiently update group keys. In a complete tree with n users, each user stores $\log_2 n + 1$ keys. For example, consider a tree of seven users in Fig. 1(a); user 1 has its secret key U_1, group key GK, and KEKs K_{14} and K_{12}.

2) User Join: When a new user joins the group, the KDC first establishes a shared key with the user. To add the user into the key tree, the KDC needs to update the group key and KEKs along the path from the new user's node to the root. The KDC sends the updated keys encrypted by the shared key to the new user, and sends the modified KEKs encrypted using other keys to the rest of the users, if necessary.

Consider the scenario in Fig. 1(b). When user eight joins the group, KDC performs a key exchange procedure with the new user, establishing a secret key U_8. To ensure backward secrecy, KDC needs to change the group key and KEKs that reside on the path from the root to the node of user 8. KDC then unicasts necessary KEKs and the new group key GK' to user 8, and unicasts the new k_{78} to user 7. To update the group key for other existing members in the tree, KDC can multicast the message to subgroups of the tree encrypted by their shared KEKs. Specifically:

1) KDC multicasts K_{58} encrypted with K_{56} to $U5$ and $U6$;
2) KDC multicasts K_{58} encrypted with K_{78} to $U7$;
3) KDC multicasts GK' encrypted with K_{14} to $U1$ to $U4$;
4) KDC multicasts GK' encrypted with K_{58} to $U5$ to $U7$.

3) User Leave: When a user leaves the group, the KDC needs to update the group key, as well as the KEKs related to the leaving user, and send them to other existing users. In Fig. 1(c), when user four leaves the group, the KDC needs to change the group key to GK'' and update the KEKs that belonged to user 4.

B. Key Management Based on the Chinese Remainder Theorem

Although each device can individually manage its group key using LKH, the overhead grows linearly with the number of devices. For example, when a user subscribes to m devices, the user needs to store and maintain m sets of keys including KEKs, secret keys, and group keys. If there are multiple users subscribing to the same m devices, there will be m GKM trees with identical users. When a user unsubscribes to all devices and leaves the system, all devices need to update their tree structures and remaining users need to update m sets of keys

[2]The devices can be partitioned based on device functionalities or user habits, because devices with similar functionalities often have similar access permissions [13]. GROUPIT can flexibly adapt to any grouping methods. For example, devices' accessibility can be grouped based on whether they requires owner presence. Another way is to limit users' accessibility level so that they cannot gain access to critical devices related to security and privacy.

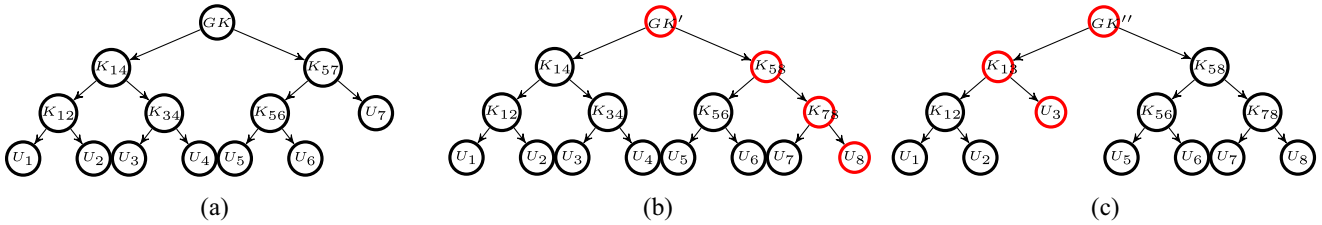

Fig. 1. LKH GKM. (a) LKH group with seven users. (b) LKH group when $U8$ joins. (c) LKH group when $U4$ leaves.

that have the same update procedure but with different key values.

Here, we describe a key management scheme leveraging the CRT. This allows multiple decryption keys to decrypt the same message encrypted by an encryption key [16].

The main idea of a CRT-based GKM is to generate one master key and several slave keys, such that the message encrypted by the master key can be decrypted by all legitimate slave keys. For example, consider there are n public/private slave key pairs $e_i,\ d_i,\ i \leq n$ with $p_i,\ q_i$ being the ith prime number pair, and one master key pair e_M, d_M. For simplicity, we now consider the modulus of the prime pairs $\phi(p_iq_i) = (p-1)\times(q-1)$ are mutually prime to each other. For a plaintext P and a ciphertext C, the master key should satisfy

$$P^{e_M} \equiv P^{e_i} \pmod{p_iq_i} \quad C^{d_M} \equiv C^{d_i} \pmod{p_iq_i}.$$

The sufficient condition for the equation above, according to Euler's theorem, is

$$e_M \equiv e_i \pmod{\phi(p_iq_i)} \quad d_M \equiv d_i \pmod{\phi(p_iq_i)}.$$

Since we consider each modulus mutually prime to each other, e_M has a solution similar to finding the solution of the system of congruences.

We first calculate N_j, which is the product of $\phi(p_jq_j),\ j \leq n,\ j \neq i$. We can find integers M_j and m_j such that

$$M_j \times\ N_j + m_j \times \phi(p_jq_j) = 1.$$

Then the key value should be

$$e_M = \sum_{i=1}^{n} e_i \times M_i \times N_i.$$

When e_M is divided by $\phi(p_jq_j)$, only $e_jM_jN_j$ remains since all other terms are multiples of $\phi(p_jq_j)$

$$\begin{aligned} e_M &\equiv e_jM_jN_j \\ &\equiv e_j\big(M_jN_j + m_j\phi\big(p_jq_j\big)\big) - e_jm_j\phi\big(p_jq_j\big) \\ &\equiv e_j \pmod{\phi\big(p_jq_j\big)}. \end{aligned}$$

An advantage of this CRT-based scheme is that if one of the slave keys needs to be changed to $e'_j,\ d'_j$ (e.g., the corresponding member has left the group), the master key can be recalculated without needing to update other slave keys

$$e'_M = e_M - e_j \times M_j \times N_j + e'_j \times M_j \times N_j.$$

This changes one slave key while other unchanged slave keys remain valid. However, to change the number of slave keys, we need to generate a new prime pair p_{n+1}, q_{n+1}, which $\phi(p_{n+1}q_{n+1})$ needs to be mutually prime to all other $\phi(p_iq_i), i \leq n$. As for master key calculation, all N_i and M_i need to be recalculated, which is a $O(n)$ overhead. Hence, the CRT-based scheme is extremely inefficient when the number of devices changes frequently.

C. Key Derivation Function

A key derivation function derives one or more keys of a given size from a secret value. In our proposed method, we leverage a one-way cryptographic function such as SHA-256 [8] to derive keys, such that it is infeasible to obtain secret values from the derived keys.

D. Related Work

Extensive work has been done regarding secure and efficient GKM in IoT and senor networks. Zhang and Varadharajan [25] surveyed several key distribution schemes in wireless sensor networks. Roman *et al.* [19] analyzed how existing key management used in actual Internet scenarios can be applied to IoT, and analyzed the applicability of link-layer oriented key management systems. LKH [9] and one way function tree [4] are two efficient GKM schemes. However, because ill-implemented OFT suffers from collusion attacks [10] and OFT increases devices' computational overhead for obtaining group keys, it is far from ideal in an IoT environment, where the devices may have limited computational power. We leverage LKH as the skeleton of our proposed method and try to accommodate multiple devices.

Tsai *et al.* [21] proposed a lightweight symmetric key establishment based on the Kronecker product. In our proposed method, we further consider the key update when users or devices join and leave the system to ensure forward and backward secrecy. Abdmeziem *et al.* [3] developed a decentralized batch-based GKM in which devices are divided into subgroups to reduce key update overhead and to mitigate the single point of failure issue. Veltri *et al.* [22] reduced membership change overhead by partitioning time into fixed-length intervals, and handles rekeying acts based on time intervals. Our proposed method further considers that users may join multiple groups at the same time.

Asymmetric encryption mechanisms are also used in key management schemes [20], [23]. Porambage *et al.* [17] provided group key establishment protocols for multicast communication by leveraging elliptic curve cryptographic (ECC) operations suitable for resource-constrained devices. Our method takes advantage of the CRT-based construction proposed by Park *et al.* [16] to accommodate multiple device groups. However, Park *et al.* did not consider dynamic device

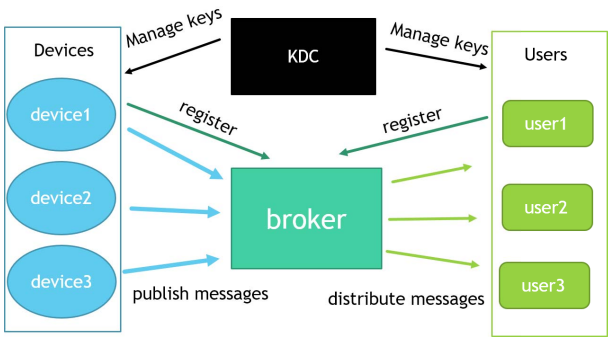

Fig. 2. System model.

groups, and members of device groups also needing to update their keys efficiently. In this paper, we propose an efficient key management scheme on the device side as well. We also propose an ID update method to mitigate possible collusion attacks.

Attribute-based encryption [6], [7], [12] can be used to support fine-grained access control. However, because attribute-based encryption heavily relies on asymmetric cryptography, it is unsuitable to run on resource-constrained IoT devices [24]. In addition, its high revocation overhead makes it unsuitable for dynamic IoT environments.

III. Problem Definition

This paper aims to develop a GKM scheme that can efficiently accommodate dynamic membership and device number changes. Before presenting our solution, we introduce the system model and attacker model.

A. System Model

Fig. 2 illustrates a typical smart home or smart city environment consisting of IoT sensor *devices* publishing data, *users* subscribing to data, a *broker* that stores and forwards data, and a KDC managing keys that are used to control access to the data. The broker is online and untrusted, whereas the KDC is offline and trusted.

A sensor device, such as a door opening sensor or IP camera, collects data and sends it to its subscribers. A user can be a device owner with legitimate and permanent control of the device or a guest user who has limited access to the device. A user can also be a sensor or actuator, which receives information from other devices. This last class of users usually possesses limited computational power and storage space. To collect and distribute the information generated by sensor devices, and to send commands or notifications to actuators and users, a centralized controller called a broker or gateway is often needed as a communication bridge between users and devices.

Combined with the gateway or working as an independent component, the KDC is designed to generate, manage, and provide necessary encryption keys to users and devices. We assume that the KDC can establish a one-time secure channel between users and devices, which can be used to authenticate and configure a newly joined user/device (e.g., by installing a shared secret key).

1) Dynamic Membership and Device Changes: Both the number of users and the number of devices can change over time. That is, a user can join or leave and a device can be installed in or removed from the system at any given time. In addition, membership—the set of devices a user subscribes to—can also change.

B. Attacker Model

The attacker's goal is to access device data without proper permission. The attacker can either be an outsider, who has no access to any devices, or an insider attempting to expand the access scope, such as trying to decrypt data generated before or after the authorized time period. The attacker can also be a compromised device that attempts to gain access to data that are unknown to it. To do so, the attacker can either try to derive encryption keys (e.g., exploiting a vulnerable key generation procedure) or collude with devices in the system so as to derive keys that they cannot obtain individually.

C. System's Requirements

1) Accommodate Multiple Dynamic Users and Devices: To support dynamic IoT environments, a GKM scheme should be able to manage multiple devices as well as multiple users. Moreover, devices can also join and leave the system at any moment. To ensure data security, the system should achieve *forward and backward secrecy*. Forward secrecy means that future keys cannot be obtained by former group members, and backward secrecy means a new user cannot obtain group keys that were used before he or she joined.

2) Minimize Update Overhead: To achieve forward and backward secrecy, the encryption keys and KEKs might need to change when users and devices join or leave the system. We want to minimize the length of update messages and the message sending time.

IV. Proposed Scheme: GroupIt

In this section, we introduce our method and explain how it handles membership and device number changes. Table I is a summary of notations used in our construction.

A. Overview

GroupIt is a two-tier GKM architecture, where the upper and lower tiers are responsible for key management *between groups* and *within groups*, respectively. Specifically, users and devices are both partitioned into the following groups.

1) *Device Groups:* GroupIt establishes a fixed number of device groups based on different functionalities, security levels, etc. When a new device joins the system, it is assigned to exactly one of the device groups.
2) *User Groups:* Given M device groups, GroupIt establishes $2^M - 1$ user groups. Each user is assigned to exactly one of the user groups based on the device groups he subscribes to.

Each device group and user group runs a GKM scheme (e.g., LKH) to efficiently handle key updates that take place when

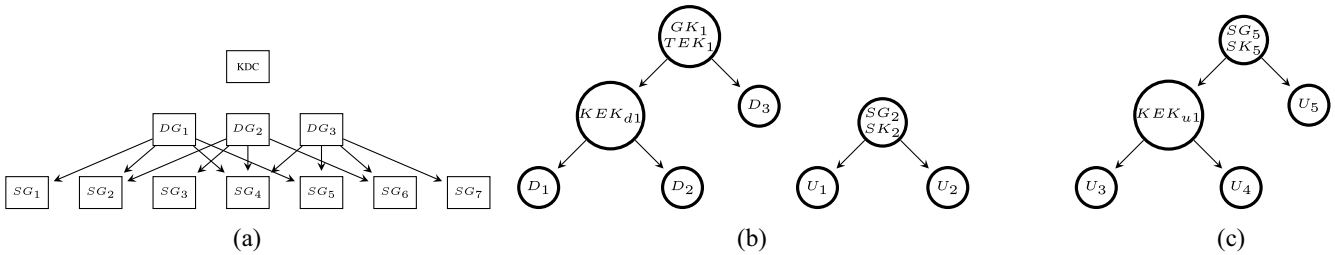

Fig. 3. Initial structure overview. (a) Three device groups and seven user groups. Structure inside (b) DG_1, (c) SG_2, and (d) SG_5.

TABLE I
SUMMARY OF NOTATIONS USED IN OUR CONSTRUCTION

KDC	Key Distribution Center
KEK	Key encryption key
TEK	Traffic encryption key
M	Total number of device groups
N	Total number of user groups
n	Total number of users
SG_x	user group x
n_x	Total number of users in user group x
m	Total number of devices
DG_y	device group y
m_y	Total number of devices in device group y
MK	Master key stored in the KDC
GK_y	Group key for device group y
SK_x	x-th slave key, which is known by all users in user group SG_x
ID_{yj}	device ID of device j in device group y
DK_{yj}	Encryption key of the j-th device in device group y
U_i	Shared secret key shared between user i and KDC.
D_j	Shared secret key between device j and KDC.
g_x	Number of device groups the x-th user group subscribes to.
$h(.)$	Hash function.
$(.)_k$	Encryption function using encryption key k.

there are membership changes inside the group. The upper tier of GROUPIT is used to avoid unnecessary communication overhead when KDC needs to update multiple user groups.

After the system is constructed, KDC is in charge of managing key updates when the membership or device number changes. The GKM schemes used in two levels can be freely chosen from existing GKM methods to meet different needs as long as the whole structure guarantees forward/backward secrecy of transmitted data and device keys; meaning, the information can be obtained only by legitimate users. Different GKM schemes also lead to different overhead tradeoffs. In our proposed method described below, we adopt LKH (mentioned in Section II) for GKM within user and device groups, and using MKE GKM based on CRT for KDC to communicate with multiple user groups.

B. Construction

1) Initialization: The GROUPIT architecture consists of three components: 1) KDC; 2) device groups; and 3) user groups. Multiple device groups are established and each accommodates devices with similar attributes. Considering all combinations to which device groups users may subscribe, user groups are constructed and each user is assigned to one of the user groups according to its membership status. KDC generates secret keys for each user and device, group keys for each user and device group, device ID for each device, traffic encryption keys and KEKs to handle key updates.

Algorithm 1 Message Sending When User i Joins Group SG_x

1: $KDC \xrightarrow{user\ i} : U_i$.
2: $KDC \xrightarrow{broadcast} All$: "devices to which new user subscribes and old users subscribing to them should update $TEK' = h(TEK)$"
3: $KDC \rightarrow \{SG_x - user\ i\}$: updated KEK' encrypted by either secret keys or shared KEK.
4: $KDC \rightarrow user\ i : (SK, TEK', IDs)_{U_i}$

As shown in Fig. 3, devices are grouped into three different groups, and each user is assigned to one of the seven user groups, according to the different device groups they subscribed to. KDC calculates the master key MK and slave keys SK_x, for $x < 2^M = N$ for each user group, which are LKH structures for users with SK_x as the root node. Also, for each device group, KDC constructs an LKH tree management structure with a group key GK_y as the root.

As shown in Fig. 3(b), device group 1 contains three devices. Each device has its own ID $\mathrm{ID}_1 j$, unknown to other devices, and encryption key $DK_{1j} = h(TEK_1|\mathrm{ID}_1 j)$, $1 \le j \le 3$ which is used to encrypt the data they transmit. In addition, to efficiently communicate with KDC and update encryption keys, the devices also need to store the root group key GK_1 and necessary KEKs (KEK_{d1} for devices 1 and 2).

As shown in Fig. 3(c) and (d), users 1 and 2 both subscribe to device groups 1 and 2, and users 3–5 subscribe to device group 1 and 3. All users establish a shared secret key U_i with KDC, and they have TEK of the device groups they subscribe to, which are TEK_1 and TEK_2 for users 1 and 2, and TEK_1 and TEK_3 for users 3–5. They can compute device keys of each device in the group with the device IDs. Users also need to store KEKs on the path from their node to the user group root for updating purposes.

2) On User Joining Group x: Consider the update procedure described in Algorithm 1.

KDC: The KDC first establishes a shared secret key U_i with the newly joined user. Second, the KDC can broadcast a notification to other existing users in the group, and other user groups which subscribe to the same device groups, to update their TEKs. The newly joined user cannot deduct the old TEK. Then, the KDC updates the LKH structure of user group x and send the necessary KEKs, new $TEKs$, and device IDs to the new user through unicast.

Users: Existing users who subscribe to the same device groups with the new user have to update their group keys, which will be $TEK'_y = h(TEK_y)$. The newly joined user, upon

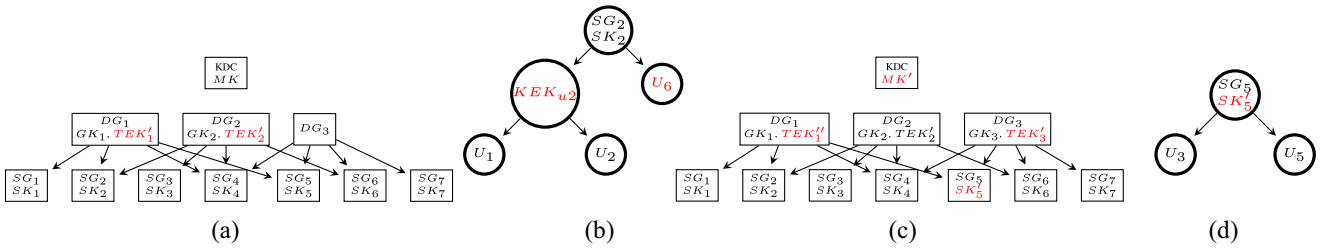

Fig. 4. Example of structure update for user join/leave events. (a) Structure when $U6$ joins. (b) Structure inside SG_2 when $U6$ joins. (c) Structure when $U4$ leaves. (d) Structure inside SG_5 when $U4$ leaves.

Algorithm 2 Key Update When User i Leaves Group SG_x

1: $KDC \rightarrow SG_x$: SK' encrypted by KEK or secret keys
2: $KDC \xrightarrow{broadcast} All$: $((TEK_i'|update\ methods)_{MK'})_{TEK_i}$
3: $KDC \rightarrow device\ group$: $(TEK')_{DK}$
4: $KDC \rightarrow devices$: random numbers for updating IDs encrypted by KEK or secret keys.

Algorithm 3 Key Update When Device j Joins Group DG_y

1: $KDC \rightarrow device\ j$: (D_j) ID_{yj}
2: $KDC \xrightarrow{*} All$: old devices need to update group key $GK_y' = h(GK_y)$
3: $KDC \rightarrow devices$: updated KEK' encrypted by either secret keys or shared KEK
4: $KDC \rightarrow usergroups$: ID of the new device encrypted by TEK.

receiving U_i and encrypted information sent by KDC, can calculate device keys.

Devices: The devices, upon receiving the notification, update their own encryption key $DK_{yj}' = h(TEK_y'|\mathrm{ID}_{yj})$, where $TEK_y' = h(TEK_y)$. In Fig. 4(a), when user six joins the system and subscribes to device groups DG_1 and DG_2, KDC needs to establish a secure transmission channel, give a secret key U_6 to the user, and assign the user to user group SG_2. Then, KDC needs to broadcast an update notification to devices and existing users to change their TEK for device groups 1 and 2, which are $TEK_1' = h(TEK_1)$, $TEK_2' = h(TEK_2)$. The structure of user group SG_2 needs to be reconstructed since a new user has joined. New necessary KEK is sent to existing users. In this case, KEK_3 is sent to U_1 and U_2, encrypted by U_1 and U_2 separately. KDC then sends necessary KEKs encrypted by U_6 to user 6, which is SK_2 in this case, and unicasts new TEKs along with IDs for all devices in the groups encrypted with U_6 to user 6.

3) On User Leaving: Consider the update procedure described in Algorithm 2.

KDC: The KDC generates a new user group key SK_x' for the group being left, and generates a new master key MK' accordingly. KDC sends the updated SK_x' encrypted with KEKs or users' secret keys to the remaining users in SG_x. The KDC then broadcast the newly generated group key via encrypted message $((TEK_i'|update\ methods)_{MK'})_{TEK_i}$, which only users with old TEK_i and a valid SK can decrypt.

Remaining Users: The remaining users decrypt the message and calculate new device keys $DK_{yj}' = h(TEK_y'|\mathrm{ID}_{yj}')$.

Devices: Devices in the groups that are being left need to change their device keys. To prevent a leaving user from leveraging the old ID and obtaining additional data, the ID needs to be updated as well. The new TEK_y' and ID update method is sent by KDC.

In Fig. 4(c) and (d), when user 4 leaves user group 5, the tree structure in the group reconstructs, and the node representing user 3 moves up to the position of its former parent node. Users 3 and 5 both receive the broadcasted message $((TEK_1''|updatemethods)_{MK'})_{TEK_1'}$, $((TEK_3'|updatemethods)_{MK'})_{TEK_3}$. They then can decrypt the message and update the device keys, and so can the users in user groups that subscribe to either DG_1 and DG_3. Before we discuss the update method for device IDs, we briefly describe a potential collusion attack that may be performed by a leaving user and a malicious device controlled or compromised by the leaving user.

a) Collusion attack: To compute every device key, the user has every ID of each device in the group. If the ID does not update when a user leaves the group, the user can try and collude with a device which has new TEK but does not have other devices' IDs. Together, they can combine their knowledge to obtain other devices' new device keys. To prevent this kind of attack, an ID update for every user leaving event is necessary. However, it would cause a huge overhead if the IDs are individually generated and sent to each device and user. We would like to make users and devices update the IDs themselves by sending shorter messages, while retaining a certain degree of diversity for the updates between devices.

The update method for device IDs is explained below. This method is randomly generated by KDC and may vary between each device; the KDC may choose $\log(m_y)$ random numbers and choose one device as a pivot. Similar to the tree partition when updating group keys, the KDC sends $\log(m_y)$ multicast, each containing a unique random number, to the devices like the pivot device is leaving the group. As shown in Fig. 5(a), device 3 is chosen as the pivot, so KDC partitions the tree the same way as when device 3 is leaving the group. This means devices 1 and 2 are sharing the same update number encrypted with KEK_{d1}, and device 3 gets another update number encrypted by secret key D_3. Fig. 5(b) shows another way to partition, where device 2 is chosen as the pivot. Then, all three devices receive different update numbers encrypted by their secret keys D_j. The devices update their own IDs with the number

$$\mathrm{ID}_{yj}' = h\big(\mathrm{ID}_{yj}|\mathrm{number}\big). \tag{1}$$

4) On Device Joining: Consider the update procedure described in Algorithm 3.

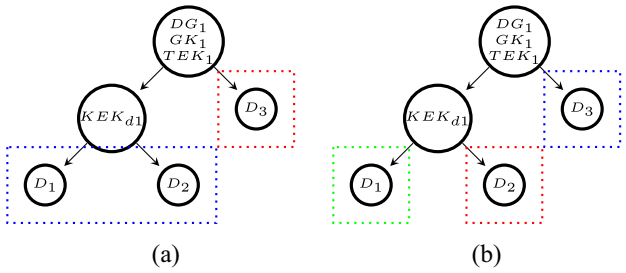

Fig. 5. How KDC distributes update methods to devices. (a) Update method in DG1 if $D3$ is chosen as the pivot. (b) Update method in DG1 if $D2$ is chosen as the pivot.

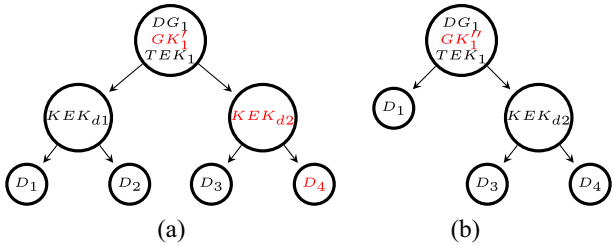

Fig. 6. Examples of structure update for device join/leave. (a) When D_4 joins DG_1. (b) When D_2 leaves DG_1.

Algorithm 4 Key Update When Device j Leaves Group DG_y

1: $KDC \xrightarrow{broadcast} All$: "leaving device j is no longer available"
2: $KDC \rightarrow DG_y$: updated GK' and KEK's.

KDC: The KDC generates a unique device ID for the newly joined device that joins device group DG_y, and also shares a secret key with it. The KDC updates the necessary part of the LKH in which the device resides, notifies the existing devices to update the group key, then sends the KEKs, newly updated group key GK'_y, ID_{yj} and TEK_y to the device through unicast. KDC also needs to notify the users that a new device is joining the group and sends the ID to them.

Devices: Existing devices update the group key $GK'_y = h(GK_y)$. The newly joined device calculates its own device key $DK_{yj} = h(TEK_y|\mathrm{ID}_{yj})$.

Users: Users that subscribe to a device group get notified by KDC that there is a new device joining the group, and they can calculate the device key with the ID_{yj} sent by the KDC encrypted by the user group key SK.

In Fig. 6(a), when device 4 joins the system and is assigned to device group 1, the KDC notifies existing users to update GK_1 to GK'_1. Next, KDC establishes a shared secret key D_4 with device 4. KDC then sends necessary information TEK_1, GK'_1, KEK($d2$) to device 4 encrypted with D_4, and sends KEK($d2$) to device 3.

5) On Device Leaving: As Algorithm 4 describes, when a device leaves the group, the KDC rearranges the tree structure in the group and sends an updated group key GK to the remaining devices. Then, the KDC broadcasts a message to notify the users that the leaving device is no longer available. In Fig. 6(b), when D_2 leaves the group, KDC generates a new device group key GK''_1 and sends it to D_1, D_3, and D_4. Then, KDC notifies the users who subscribe to device group 1 that D_2 is no longer a valid device.

V. Implementation

To demonstrate feasibility and applicability, we implement our proposed method based on Alljoyn, an open-source framework for IoT devices and applications to discover and communicate with each other. First, we briefly describe the concept and how devices and users communicate in the Alljoyn framework. Then we describe our implementation layout and implemented components.

Alljoyn uses an event-and-action mechanism to support communication between devices and applications. A device can emit messages or receive information from other devices or applications to trigger actions. To advertise a message, a device can use either an announcement or a known ID to announce and discover each other. For example, a server can announce itself by sending a signal including a session port, device name, supporting language, etc. A client then discovers the signal, learns the information about the server, and subscribes to its signal. Alljoyn's framework is applicable to a wide range of data collection and distribution scenarios, including smart home appliances. In previous studies, it has been shown that users tend to categorize devices into a fixed number of groups based on devices' security levels [13], which is a suitable application scenario to put our proposed architecture into practice. Our implementation uses near field communication (NFC) [11] as a means of a one-time secure channel, which is used to distribute a secret shared key between a user and the KDC. Attackers cannot obtain a legitimate key if they cannot reach the NFC-range of the KDC. To simulate the computational power of resource-constrained devices, we execute our program on Linux virtual machines. An AES-256 encryption on a 64-byte block takes 720 ns, while a ECC-224 decryption takes about 84300 ns, and a SHA-256 function on a 64-byte block takes 600 ns. The ECC operation is the most energy-consuming step in the proposed method, and has been supported by other studies [17] that the overhead is tolerable for resource-constrained devices.

In the implementation, we create device simulators to simulate different types of devices generating various messages. We have a centralized server in charge of receiving messages from message providers and sending messages to subscribers. For example, the server first establishes a service name called *alljoyn.temperature*. A user application can search for the name and subscribe to it; a smart thermometer can also send a message to the *alljoyn.temperature* object and change the value. Once the value is changed, the server sends a message to the subscribed clients. The implementation is to ensure the sent message is encrypted and can only be decrypted by the corresponding sender and the receiver. For easy demonstration, the KDC is built inside the server such that the updated key can be sent from the server to the client and the device. In practice, the KDC can be an individual entity apart from the server, such that the server will be unable to obtain the decrypted information.

To implement the concept of device groups, devices in the same group can subscribe to a common service name. For instance, both a smart lock and a smart camera can send their messages to *alljoyn.security*. They can append their device name to the encrypted messages they generate. Once the message is received by the clients, they can parse the messages and decrypt them based on the appended device names.

VI. Analysis

In this section, we discuss the overhead of our proposed method in different aspects (storage, computation, and communication) and compare with existing methods. We also discuss the probability of collusion of the ID update method we proposed above for devices in the same group.

Without losing generality, we assume that IoT devices and users are equally distributed in each device group and subgroup, and the LKH structures are all balanced binary trees. In the next sections, we discuss the overhead of users in subgroup SG_x, and devices in DG_y.

A. Storage Overhead

Users need to store $g_x * m_y$ IDs, one secret key, g_x *TEK*s, and $\log(n_x)$ KEKs including *SK*. Devices stores their own IDs and the secret keys with the server, one *TEK*, one *GK*, and $\log(m_y)$ KEKs.

B. Computation Overhead

1) When User Leaves Subgroup x: Devices need to do one symmetric decryption, hash-update their own IDs and gain the new device key. The remaining group users need to do one symmetric and one asymmetric decryption to gain the update information. and g_x*m_y hash to gain new device keys.

2) When User Joins Subgroup x: Devices need to perform one hash function to update their *TEK* and another hash function to derive their new device keys. The new user needs one symmetric decryption to gain the subgroup key *SK*, new *TEK*, KEK for the tree, and all IDs of the devices in the device groups that are subscribed to. Existing users, on the other hand, need to perform $O\log(n_x)$ symmetric decryption to gain new KEKs.

3) When Device Joins Device Group y: Old devices need to perform one hash calculation to update the device group key GK_y. Since the LKH structure of the group might change, some devices need to decrypt $O(\log(m_y))$ KEK update messages. The new device decrypts the message sent from KDC to obtain KEKs and calculate its own device key. Users subscribed to the group which the new device is joining need to decrypt the message sent by KDC. The users then perform one hash calculation to gain the new device key.

4) When Device Leaves: The remaining devices perform one symmetric decryption to gain the new group key. Users do not need to perform extra computation. When they receive the message sent from KDC, they simply stop accepting messages from the leaving device.

C. Communication Overhead

When a user leaves group x, the KDC needs to send $\log(n_x)$ mulitcasts to the remaining users to update the subgroup slave key, one broadcast to all the subgroups about the new *TEK* and IDs, and $\log(m_y)$ multicasts to devices in order to update the IDs. When a user joins, the KDC establishes one secret key with the new user, sends one unicast to the new user, and broadcasts key update messages to existing users. When a device joins, the KDC establishes one secret key with the device, sends one broadcast to the subscribed users of the new ID of the device, and sends one unicast containing necessary keys to the device. When a device leaves device group y, the KDC broadcasts a message that the leaving device is no longer available, and multicasts $\log(m_y)$ messages to update device group keys for the remaining devices in group y.

D. Comparison With Existing Methods

In this section, we compare the update overhead between our method and other existing key management methods. In the comparison, we set the number of user groups to 16, since the number of users only affects the scale of comparison, and the number of devices users subscribe to is the variable, ranging from 1 to 200 to analyze performance difference between each method with respect to device quantity. We also assume the encryption key length is 32 bytes, so the length of device IDs in our method is 32 bytes. On an Intel i7-6700HQ CPU, an AES-256 encryption on a 64-byte block takes roughly 800 ns, while a ECC-224 decryption takes about 114 000 ns, and a SHA-256 function on a 64-byte block takes 460 ns. It may take longer for devices with limited computational power to solve these operations, but the time difference between operations should remain in a similar range.

1) LKH: In order to handle multiple devices individually, the traditional LKH method needs to construct an LKH tree for each user group of devices. Unlike the proposed method, which only needs to handle the KEK of a single tree, the user will have to store m_y encryption group keys as well as $m_y \times \log(n_x)$ KEKs. Devices, however, do not need to store much information in the traditional LKH scheme, because the devices are not grouped and do not need to store group keys and KEKs. When a new user joins the system and subscribes to m_y devices, all m_y trees will have to reconstruct in order to accommodate the new user. All other users in these groups will have to update the group keys and receive $O(\log(n_x))$ new KEKs. If an old user subscribes to multiple devices, this overhead will multiply by the number of devices subscribed to by both the new and old users. On the other hand, in our proposed method, users only need to store one set of KEKs for one subgroup tree and one short ID per device, instead of an encryption key with a complete length. In update scenarios, the new user only needs to handle a KEK update in a single tree as well. For existing users, only those who have the exact same subscription case with the new user will need to update the LKH KEKs in the subgroup.

Although users need to perform an asymmetric decryption, which is a large overhead compared with symmetric computation, when there are users leaving the group, the overhead

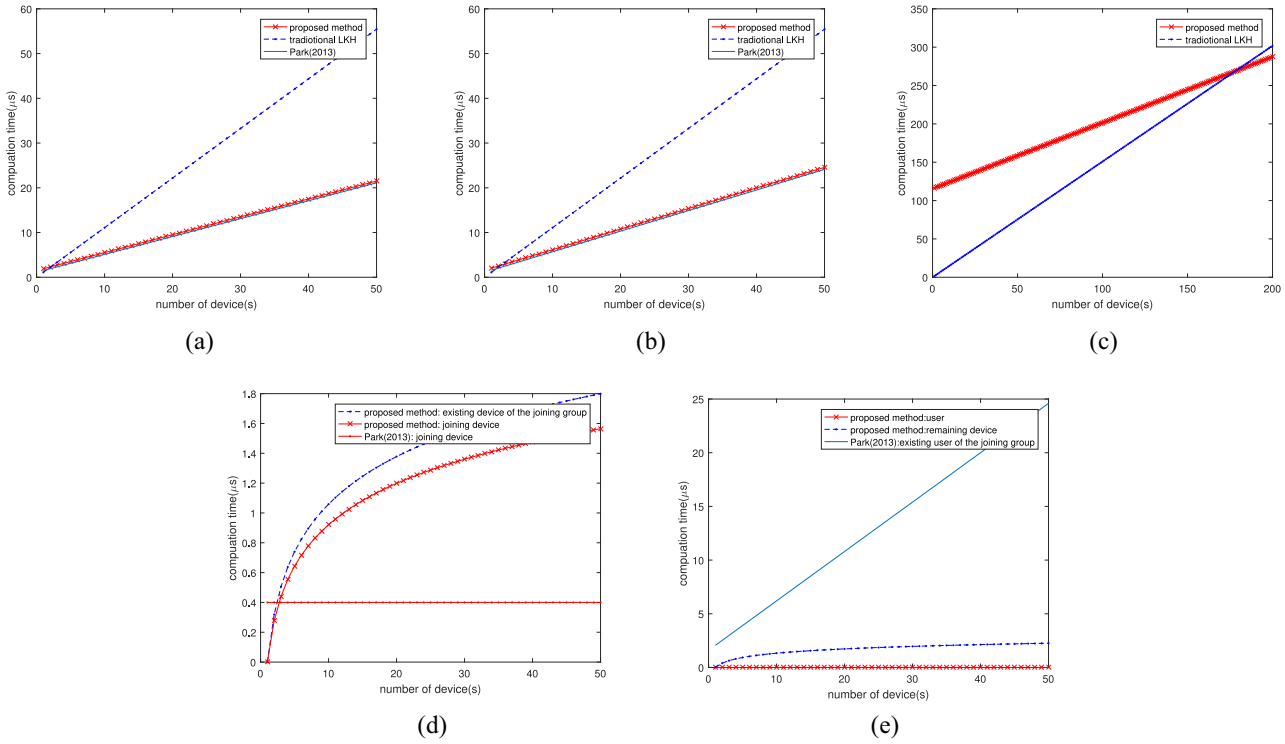

Fig. 7. Computation overhead comparision. User join: (a) overhead of the new user, (b) overhead of existing users, and (c) overhead of remaining users. (d) Device join: overhead of devices. (e) Device leave: overhead of users and devices.

will be surpassed by the need to update the group key of each device when the number of devices grows. As shown in Fig. 7(c), the computational overhead in our method can be represented as $\log(n_x) \times t_{\text{AES}} + t_{\text{ECC}} + (m_y) \times t_{\text{AES}} + m_y \times t_{\text{SHA}}$, where n_x is the number of users in the subgroup, m_y is the number of devices in the device group, t_{AES} is the computation time for an AES decryption for a 32-byte block of data, t_{ECC} is the computation time to perform a ECC-224 decryption, and t_{SHA} is the computation time for a SHA-256 computation. The computational overhead for the traditional LKH can be represented as $\log(n_x) * t_{\text{AES}} * m_y$, which is smaller than the overhead of our proposed method when the device number is small. However, its growth rate is much higher and the overhead surpasses our method's when the number of devices in a group is around 180. This shows that our method is more scalable to the growing IoT environment.

2) Master Key Encryption: Since our proposed method also leverages a similar MKE structure, the existing method proposed by Park *et al.* [16] has a similar storage overhead and update overhead when user membership changes. The overhead of our method will occasionally be larger since the devices or users need to perform a hash function to gain new keys. However, one of the main weakness of MKE is the huge overhead when the number of slave keys needs to be changed, as described in Section II. When a new device joins the system, 2^m new slave keys need to be generated in order to accommodate all possible subgroups generated by the new device, which is a huge overhead for the KDC. Furthermore, the users still need to perform joining procedures to subscribe to new devices, which is $t_{\text{AES}} \times \log(n_x) + t_{\text{AES}} \times m_y$. This also means the user needs to leave the existing group first, and the remaining users need to perform asymmetric decryption to gain new TEKs for the subscribed-to devices. When a user leaves the group to subscribe to new devices, the remaining users need to compute as many asymmetric decryptions as the number of devices they subscribe to, which is not comparable to our method. The existing users in the new group where the user joins, shown in Fig. 7(d), have to update their KEKs as well as their TEKs. Our proposed method does not require new slave keys to handle device joining. This is because new devices are assigned to existing device groups, where the existing devices need to update their group keys. Only one new device ID is needed to be acknowledged by users who do not have to change their group, so the update overhead will be constant. This will surely affect the flexibility of users' subscription preferences and induce overhead to the devices, since they have to maintain a key management structure themselves. However, it will greatly reduce update overhead on the user's side, since some of the users can be devices which do not have much computational power. In addition, if devices with similar attributes are automatically formed into groups, users that want to receive certain types of information can automatically obtain information from new devices without having to subscribe to them manually. When a device leaves, the remaining devices in our method have to receive and decrypt a message from KDC to obtain the new group key, while users simply revoke the leaving device from their lists, requiring no computational power. Users in Park's method will have to join another subgroup, which has the same overhead as other user-join scenarios shown in Fig. 7(e).

E. ID Update Method Collusion

In this section, we discuss the probability of a successful collusion attack performed by an evil device and a leaving client to gain device keys of other devices in the same group. We calculate the probability by considering the relative position between a victim and an evil device in the LKH structure.

For example, consider an evil device a and a target victim device b. Both devices share the same update method as long as neither of them are chosen as the pivot for update partition; therefore, the probability is $1-2/n$. If their shared parent level is the second level of the tree, they will share the same update method as long as none of the four nodes are chosen, as the probability is $1-4/n$.

Finally, we sum up the conditional probabilities as follows:

$$\sum_{i=1}^{\log_2 n} \frac{2^i-1}{n-1} * \left(1-\frac{2^i}{n}\right) \tag{2}$$

which is $9/28$ when $n=8$. The probability decreases every time a leaving event occurs as the pivot is randomly chosen and the tree structure may shuffle and the positions changes among users.

VII. Conclusion

In this paper, we explored the GKM problem in dynamic IoT environments, where user memberships and device numbers change frequently. To reduce overhead caused by said events, we proposed GroupIt, a two-tier GKM architecture that combines two existing GKM methods by using a device grouping technique. In GroupIt, users can subscribe to multiple devices at once, while each device can still encrypt its data using a unique device key.

Future Work: One interesting research challenge is how to handle collusion between devices (data senders) and users (data consumers). In our system model, the sender and the receiver of data are separated, meaning that a device is only in charge of sending information and will not directly subscribe to other devices. In practice, however, devices may be subscribers as well. A device can receive information from others and send its own data at the same time. In this case, the IDs of other devices in the same group can be known by the device if it subscribes to its own group, which will give it an opportunity to forge messages of other devices since it can obtain the device keys of others. This problem is challenging in our current architecture and can be viewed as an interesting future direction.

To put our proposed method into practice in the future, we plan to leverage our proof-of-concept implementation and construct an additional GKM module in Alljoyn. By adopting our architecture, services and applications that require multiple security-level access controls and handle dynamic membership changes can increase scalability and reduce overhead for resource-constrained devices. Such services and applications, including smart home surveillance, are already supported in Alljoyn and IoT-augmented hotels [2].

References

[1] *Breaking Down Mirai: An IoT DDoS Botnet Analysis*. Accessed: Oct. 30, 2017. [Online]. Available: https://www.incapsula.com/blog/malware-analysis-mirai-ddos-botnet.html

[2] *Marriott International Teams With Samsung and Legrand to Unveil Hospitality Industry's IoT Hotel Room of the Future, Enabling the Company to Deepen Personalized Guest Experience*. Accessed: Feb. 16, 2018. [Online]. Available: http://news.marriott.com/2017/11/marriott-international-teams-samsung-legrand-unveil-hospitality-industrys-iot-hotel-room-future-enabling-company-deepen-personalized-guest-experience/

[3] M. R. Abdmeziem, D. Tandjaoui, and I. Romdhani, "A decentralized batch-based group key management protocol for mobile Internet of Things (DBGK)," in *Proc. IEEE Int. Conf. Comput. Inf. Technol. Ubiquitous Comput. Commun. Depend. Auton. Secure Comput. Pervasive Intell. Comput. (CIT/IUCC/DASC/PICOM)*, IEEE, 2015, pp. 1109–1117.

[4] D. Balenson, D. McGrew, and A. Sherman, "Key management for large dynamic groups: One-way function trees and amortized initialization," *Adv. Security Res. J.*, vol. 1, no. 1, pp. 28–42, 1998.

[5] A. Banks and R. Gupta, *MQTT Version 3.1.1*, ISO/IEC Standard 20922:2016, 2014.

[6] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *Proc. IEEE Symp. Security Privacy (SP)*, IEEE, 2007, pp. 321–334.

[7] I. Damgård, H. Haagh, and C. Orlandi, "Access control encryption: Enforcing information flow with cryptography," in *Proc. Theory Cryptography Conf.*, Springer, 2016, pp. 547–576.

[8] *180-3, Federal Information Processing Standards Publication, Secure Hash Standard (SHS)*, Inf. Technol. Lab., Nat. Inst. Stand. Technol., Gaithersburg, MD, USA, 2008.

[9] H. Harney and E. Harder, "Logical key hierarchy protocol," Internet Eng. Task Force, Fremont, CA, USA, Internet Draft, Rep., Apr. 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-harney-sparta-lkhp-sec-00

[10] G. Horng, "Cryptanalysis of a key management scheme for secure multicast communications," *IEICE Trans. Commun.*, vol. E85-B, no. 5, pp. 1050–1051, 2002.

[11] International Organization for Standardization, *Information Technology—Telecommunications and Information Exchange Between Systems—Near Field Communication—Interface and Protocol (NFCIP-1)*, ISO/IEC Standard 18092:2013, Mar. 2013.

[12] S. Jahid, P. Mittal, and N. Borisov, "Easier: Encryption-based access control in social networks with efficient revocation," in *Proc. 6th ACM Symp. Inf. Comput. Commun. Security*, ACM, 2011, pp. 411–415.

[13] T. H.-J. Kim, L. Bauer, J. Newsome, A. Perrig, and J. Walker, "Challenges in access right assignment for secure home networks," in *Proc. HotSec*, 2010, Art. no. 1.

[14] Y. Kim, A. Perrig, and G. Tsudik, "Simple and fault-tolerant key agreement for dynamic collaborative groups," in *Proc. 7th ACM Conf. Comput. Commun. Security*, ACM, 2000, pp. 235–244.

[15] M. Y. Malik, "Efficient group key management schemes for multicast dynamic communication systems," *IACR Cryptol. ePrint Archive*, vol. 2012, p. 628, 2012.

[16] M.-H. Park, Y.-H. Park, H.-Y. Jeong, and S.-W. Seo, "Key management for multiple multicast groups in wireless networks," *IEEE Trans. Mobile Comput.*, vol. 12, no. 9, pp. 1712–1723, Sep. 2013.

[17] P. Porambage *et al.*, "Group key establishment for enabling secure multicast communication in wireless sensor networks deployed for IoT applications," *IEEE Access*, vol. 3, pp. 1503–1511, 2015.

[18] S. Rafaeli and D. Hutchison, "A survey of key management for secure group communication," *ACM Comput. Surveys*, vol. 35, no. 3, pp. 309–329, 2003.

[19] R. Roman, C. Alcaraz, J. Lopez, and N. Sklavos, "Key management systems for sensor networks in the context of the Internet of Things," *Comput. Elect. Eng.*, vol. 37, no. 2, pp. 147–159, 2011.

[20] S. Sciancalepore, A. Capossele, G. Piro, G. Boggia, and G. Bianchi, "Key management protocol with implicit certificates for IoT systems," in *Proc. Workshop IoT Challenges Mobile Ind. Syst.*, ACM, 2015, pp. 37–42.

[21] I.-C. Tsai, C.-M. Yu, H. Yokota, and S.-Y. Kuo, "Key management in Internet of Things via Kronecker product," in *Proc. IEEE 22nd Pac. Rim Int. Symp. Depend. Comput. (PRDC)*, IEEE, 2017, pp. 118–124.

[22] L. Veltri, S. Cirani, S. Busanelli, and G. Ferrari, "A novel batch-based group key management protocol applied to the Internet of Things," *Ad Hoc Netw.*, vol. 11, no. 8, pp. 2724–2737, 2013.

[23] P. Vijayakumar, S. Bose, and A. Kannan, "Chinese remainder theorem based centralised group key management for secure multicast communication," *IET Inf. Security*, vol. 8, no. 3, pp. 179–187, May 2014.
[24] X. Wang, J. Zhang, E. M. Schooler, and M. Ion, "Performance evaluation of attribute-based encryption: Toward data privacy in the IoT," in *Proc. IEEE ICC*, 2014, pp. 725–730.
[25] J. Zhang and V. Varadharajan, "Wireless sensor network key management survey and taxonomy," *J. Netw. Comput. Appl.*, vol. 33, no. 2, pp. 63–75, 2010.

Yi-Hsuan Kung received the B.S. and M.S. degrees in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 2015 and 2017, respectively.

His current research interests include network security, IoT security, and cryptography.

Hsu-Chun Hsiao (GS'10) received the B.S. and M.S. degrees from National Taiwan University, Taipei, Taiwan, in 2006 and 2008, respectively, and the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, USA, in 2014.

She is an Assistant Professor with the Department of Computer Science and Information Engineering, and the Graduate Institute of Networking and Multimedia, National Taiwan University. She also holds an Adjunct Assistant Researcher position with the Center of Information Technology and Innovation, Academia Sinica, Taipei. Her current research interests include network security, anonymity and privacy, and applied cryptography.

Dr. Hsiao was a recipient of the MOST Young Scholar Fellowship.