



A Stochastic Framework for Evaluating Key Management Schemes in Dynamic Linear-Hierarchy Internet of Things Systems

Abdessamed Seddiki¹, Mourad Benmalek^{2*}

¹ CRAN Laboratory, Université de Lorraine, CNRS, UMR 7039, Campus Sciences, F-54000 Nancy, France

² Computer Engineering Department, College of Engineering, Al Yamamah University, Riyadh 11512, Saudi Arabia

Corresponding Author Email: m_benmalek@yu.edu.sa

Copyright: ©2026 The authors. This article is published by IETA and is licensed under the CC BY 4.0 license (<http://creativecommons.org/licenses/by/4.0/>).

<https://doi.org/10.18280/ijssse.160806>

ABSTRACT

Received: 6 May 2026

Revised: 10 July 2026

Accepted: 14 July 2026

Available online: 31 August 2026

Keywords:

Internet of Things, security, Key Management Schemes, linear hierarchy, Markov chains, Poisson processes, scalability

Dynamic Internet of Things (IoT) systems increasingly rely on hierarchical access-control structures to regulate secure data sharing among entities with different privilege levels. In such environments, Key Management Schemes (KMSs) play a fundamental role in preserving confidentiality while supporting frequent membership changes caused by mobility, maintenance, or operational dynamics. However, the evaluation of hierarchical KMSs is often limited to static analyses or short-term simulations, which do not adequately capture the long-term impact of continuous joins and leaves on system overhead. To address this limitation, this paper proposes a stochastic framework for evaluating KMSs in dynamic linear-hierarchy IoT systems. The proposed framework combines discrete-time Markov chains with Poisson-driven membership events in order to model both inter-class key continuity and intra-class key distribution. Based on the stationary behavior of these processes, the framework derives steady-state insights into the storage, computation, and communication overheads induced by the considered KMS. To demonstrate its applicability, the framework is instantiated on the Secure and Efficient Key Management Scheme for Internet of Things (SEKM-IoT) as a representative case study. The obtained results show that the analyzed KMS achieves logarithmic scalability with respect to class size, while preserving the secure hierarchical accessibility required in dynamic IoT environments. The proposed framework provides a rigorous basis for moving beyond static configurations and for assessing the long-term scalability of hierarchical KMSs under realistic operating conditions.

1. INTRODUCTION

The Internet of Things (IoT) has significantly transformed the design and operation of modern infrastructures by enabling a tight integration between the physical and digital worlds [1]. Through the interconnection of billions of sensors, devices, and actuators, IoT systems now support a wide range of critical applications, including smart cities [2], intelligent transportation [3], precision agriculture [4], and energy networks [5]. This pervasive deployment has created new opportunities for automation, monitoring, and data-driven decision making. However, it has also introduced a broad and complex threat landscape. As IoT systems continue to grow in scale and heterogeneity, they generate and exchange large volumes of sensitive data that must be protected against unauthorized access and malicious manipulation. Therefore, ensuring strong security guarantees while preserving data privacy has become a major concern for both researchers and practitioners [6]. If these issues are not properly addressed, IoT deployments may suffer from serious consequences, including privacy leakage, large-scale data breaches, and even disruptions of critical infrastructures [7].

Among the different security mechanisms required to

protect IoT environments, Key Management Schemes (KMSs) play a fundamental role [8]. A KMS is responsible for generating, distributing, updating, and revoking cryptographic keys so that only authorized entities can access protected data and services. In many practical IoT applications, especially in industrial and military environments, access control is not flat, but rather hierarchical [9]. In such settings, entities are organized into ordered security classes according to their access privileges. In particular, linear hierarchical structures represent an important class of access control models in which each upper-level class is allowed to access its own data as well as the data of lower-level classes, whereas the reverse direction must be strictly prohibited. In these systems, two requirements are generally imposed. The first one is intra-class accessibility, where all members of the same class share a common group key to communicate securely within that class. The second one is inter-class derivation, where a user belonging to a higher security class can derive the keys of lower classes without requiring additional credentials. Consequently, a well-designed KMS for such environments must satisfy important security requirements [10], including confidentiality, forward secrecy, backward secrecy, and resistance against collusion.

Designing efficient KMSs for hierarchical IoT systems remains a challenging task due to both device limitations and the dynamic nature of these environments. Indeed, IoT nodes may frequently join or leave the system because of mobility, maintenance operations, battery depletion, connectivity variations, or role changes. Such membership dynamics require repeated key update operations in order to maintain the desired security properties. In recent years, several KMSs have been proposed to address these issues, including the Secure and Efficient Key Management Scheme for Internet of Things (SEKM-IoT) [9]. Although these schemes provide interesting solutions, their evaluation is often conducted under simplified assumptions. More precisely, many existing studies rely on static configurations or short-duration simulations, which do not adequately capture the cumulative and stochastic effects of continuous membership changes over long-term system operation. As a result, there is still a need for rigorous analytical frameworks capable of evaluating the long-term behavior, scalability, and overhead of hierarchical KMSs in dynamic IoT environments.

In this paper, we address this gap by proposing a stochastic framework for evaluating KMSs in dynamic linear-hierarchy IoT systems. Unlike conventional deterministic analyses, the proposed framework models the system evolution as a probabilistic process, which makes it possible to characterize the impact of user joins and leaves on the long-term behavior of the considered scheme. More specifically, the framework relies on Markov chains and Poisson processes to capture the stochastic dynamics of class membership and their effect on key management operations. In this way, the proposed approach provides a more realistic and rigorous basis for analyzing how security overheads evolve over time in steady-state conditions. To demonstrate its applicability, we instantiate the framework using SEKM-IoT as a representative case study, and we show how the obtained analytical results can support deployment decisions in resource-constrained IoT environments.

The main contributions of this paper can be summarized as follows:

- We develop a reusable stochastic evaluation workflow based on Markov chains and Poisson processes for dynamic linear-hierarchy KMSs. The workflow separates generic modeling steps from protocol-specific state-transition and overhead definitions.
- We define dedicated probabilistic models for the Key Continuity Process (KCP) and the Class Key Distribution Process (CKDP), thereby enabling a fine-grained analysis of both inter-class and intra-class key management behavior over time.
- We instantiate the proposed framework on the SEKM-IoT protocol and derive asymptotic expressions for its storage, computation, and communication overheads, thus demonstrating the ability of the framework to assess the scalability of hierarchical KMSs.
- We present a comparative evaluation with representative state-of-the-art schemes and show that the analyzed approach provides good scalability and efficiency, which makes it suitable for large-scale and dynamic IoT deployments.

Although SEKM-IoT is used as the case study in this paper, the proposed stochastic evaluation approach is not restricted to this scheme. It can be adapted to other dynamic hierarchical KMSs by defining the corresponding membership-update behavior and overhead metrics.

The remainder of this paper is organized as follows. Section

2 reviews the related work on KMSs for IoT systems and on the use of stochastic analysis in security. Section 3 presents the necessary background on SEKM-IoT, Markov chains, and Poisson processes. Section 4 details the proposed stochastic framework. Section 5 presents the analytical results and discusses the main findings. Section 6 compares the obtained results with those of previous works. Finally, Section 7 concludes the paper and outlines future research directions.

2. RELATED WORK

Ensuring secure communications is a fundamental requirement in IoT systems, since such systems support a wide range of critical applications, including smart grids, healthcare monitoring, industrial automation, and large-scale cyber-physical infrastructures. However, the design of efficient KMSs for IoT remains a challenging problem because of the limited resources of devices, the heterogeneity of the network, and the dynamic behavior of participating nodes. In particular, frequent membership changes caused by mobility, connectivity variations, maintenance operations, or device failures make secure and efficient rekeying a critical issue. Moreover, the evaluation of KMSs should not be limited to static performance snapshots, but should also capture their long-term behavior under dynamic operating conditions. For this reason, both efficient KMS design and rigorous analytical evaluation have attracted considerable attention in the literature.

2.1 Key Management Schemes

A large number of KMSs have been proposed in the literature with the objective of providing strong security guarantees while maintaining acceptable storage, communication, and computation overheads. SBSA [11] was proposed as a scheme in which nodes rely on per-session broadcast keys. Although this design helps avoid key collisions and preserve user privacy, it also induces significant storage and computation overhead at the device level. To alleviate the burden on a single Key Server (KS), DLGKM-AC [12] distributes the rekeying workload across multiple sub-key distribution centers, thereby improving scalability from the key distribution point of view.

Recent works have further investigated the design of KMSs and lightweight security mechanisms that are better adapted to dynamic and resource-constrained IoT environments. An asynchronous group KMS for IoT [13] was proposed to handle situations where devices may become temporarily unavailable because of energy-saving modes, battery depletion, or connectivity disruption. This approach improves the practicality of group key establishment in asynchronous IoT settings and provides important security properties. However, it does not address linear hierarchical access control and does not provide a stochastic evaluation of long-term rekeying behavior.

Other works have explored the use of physical characteristics as security primitives. SCBA [14] was proposed for Wireless Body Area Networks, where certificateless biometric authentication based on ECG characteristics is used to provide anonymity and message integrity. Similarly, Group-Key PHEMAP [15] is a lightweight scheme that exploits Physically Unclonable Functions (PUF) to support dynamic group management

without storing permanent keys, thus reducing the attack surface. In a different direction, another approach [16] optimizes key distribution by taking into account node energy levels and geographic location, with the objective of minimizing the global cryptographic overhead.

Several recent studies have also focused on lightweight authentication and access-control architectures for IoT systems. A blockchain-based lightweight authentication scheme using lattice encryption [17] was proposed to improve decentralization and provide post-quantum-oriented protection for IoT devices. Although this approach is relevant to secure IoT deployment, it is mainly authentication-oriented and does not study hierarchical KMS behavior. An IoT access-control system based on blockchain and message queuing [18] was introduced to manage access transactions through a decentralized and auditable architecture. This work addresses dynamic access-control management, but it does not focus on cryptographic key hierarchy or rekeying overhead. In the same direction, a decentralized authentication and data access-control scheme using decentralized identifiers [19] was proposed for fog-enabled Industrial IoT. This approach strengthens access control in industrial environments, but it does not provide a KMS-level stochastic analysis under membership churn.

At the implementation level, lightweight cryptographic algorithms for resource-constrained IoT devices, including AES-128, SPECK, and ASCON, were evaluated in the study [20]. The results confirm the importance of selecting suitable lightweight primitives according to execution time, memory usage, latency, throughput, and security robustness. An end-to-end encryption enabled lightweight mutual authentication scheme for resource-constrained IoT networks was proposed in the study [21]. These works [20, 21] are important for lightweight IoT security, but their scope remains focused on primitive-level evaluation or authentication protocols rather than dynamic hierarchical KMS evaluation.

In the domain of critical infrastructure, eSKAMI [22] and SAMI [23] were proposed for smart grid Advanced Metering Infrastructure (AMI), utilizing a key graph structure to optimize storage and rekeying for unicast and multicast traffic. This concept was further extended in iVerSAMI [24] to secure broadcast traffic by introducing multi-group graphs for dynamic service subscription. While effective for smart grid environments, these approaches are less suitable for highly dynamic and mobile IoT systems.

Linear hierarchies, which reflect the organization of many industrial and military IoT systems, have received particular attention. KTLH [25] uses hash-based relations for downward key derivation in combination with key tables to handle dynamic membership changes. This idea was refined through LHSC [26], in which key tables are removed and replaced by public class parameters at the cost of additional computation overhead. DKM [27] employs a dependent-keys approach that confines hierarchy updates to the affected subtrees, thereby

improving update locality. More recently, lattice-based cryptography has been incorporated into lightweight KMSs [28] to provide post-quantum protection for IoT data.

Building on these previous efforts, SEKM-IoT [9] combines the Logical Key Hierarchy (LKH) [29] with secure hash-based derivation to support both intra-class accessibility and inter-class key derivation in dynamic linear-hierarchy IoT systems. The scheme provides an interesting trade-off between security and efficiency and is particularly suitable for hierarchical environments in which upper-level classes must be able to access the keys of lower-level classes. However, although SEKM-IoT has shown good performance under the considered evaluation settings, its long-term behavior under stochastic membership churn has not yet been rigorously analyzed.

2.2 Markov chains in security evaluation

Probabilistic modeling provides a useful perspective for studying systems that evolve under uncertainty, and Markov chains constitute one of the most widely used tools for representing such dynamic behavior. In the security domain, these models have been used to capture random transitions between system states and to derive quantitative measures related to performance, reliability, and resilience.

For example, Hidden Markov Models (HMMs) were employed to detect adversarial traffic patterns in Network Intrusion Detection Systems [30]. Model-based security evaluation has also been shown to provide deeper insights into system dependability than simulation alone through the use of quantitative analytical models [31]. In the same direction, tools such as PRISM [32] have contributed to the development of probabilistic verification techniques for security protocols based on stochastic automata.

In the specific context of key management, Markov chains were applied to analyze the KTLH scheme [25], and analytical expressions for its storage and bandwidth overheads were derived [33]. The results showed that mathematical modeling can provide more precise insights than discrete-event simulation when evaluating the performance of hierarchical KMSs.

Despite the efforts summarized in Table 1, the existing literature remains mainly focused on the design of specific KMSs, lightweight authentication protocols, access-control architectures, or isolated performance evaluations. However, they do not provide a general stochastic framework dedicated to the evaluation of KMSs in dynamic linear-hierarchy IoT systems. To the best of our knowledge, such a framework is still lacking. This gap motivates the present work. More precisely, in this paper, we propose a stochastic framework for evaluating linear-hierarchy KMSs and instantiate it on SEKM-IoT as a representative case study. In this way, the proposed framework provides new quantitative insights into the storage, computation, and communication overheads of hierarchical KMSs under dynamic operating conditions.

Table 1. Comparison of representative Key Management Schemes (KMSs) and recent security mechanisms for Internet of Things (IoT) and linear-hierarchy environments

Scheme/Work	Target Environment	Supports Linear Hierarchy	Supports Dynamic Membership	Main Technique	Evaluation Method	Main Limitation
SEKM-IoT [9]	Dynamic linear-hierarchy IoT	Yes	Yes	Key graph structure and hash-based derivation	Complexity analysis	No rigorous long-term stochastic evaluation

SBSA [11]	Sensor networks/IoT	No	Limited	Broadcast session keys	Performance analysis	High storage and computation overhead
DLGKM-AC [12]	Distributed IoT	No	Yes	Distributed key distribution centers	Performance analysis	Focuses mainly on scalability of distribution
Asynchronous GKM [13]	Asynchronous IoT groups	No	Yes	Asynchronous group credential establishment	Security and performance analysis	Not designed for linear hierarchical access control
SCBA [14]	Wireless Body Area Networks	No	Limited	ECG-based biometric authentication	Functional and security analysis	Domain-specific applicability
Group-Key PHEMAP [15]	IoT groups	No	Yes	PUF-based lightweight group key management	Security and performance analysis	Not designed for hierarchical key derivation
Blockchain-lattice authentication [17]	Large-scale IoT	No	Limited	Blockchain identity management and lattice-based authentication	Security and performance analysis	Authentication-focused and not a hierarchical KMS
Blockchain-MQ access control [18]	IoT access control	No	Yes	Blockchain and message queuing for access management	Modeling and performance evaluation	Does not address KMS overhead or linear hierarchy
DID-based IIoT access control [19]	Fog-enabled Industrial IoT	No	Yes	Decentralized identifiers and data access control	Security and performance analysis	Access-control focused and not a KMS-level analysis
LWC evaluation [20]	Resource-constrained IoT devices	No	No	Evaluation of AES-128, SPECK, and ASCON	Experimental evaluation	Primitive-level evaluation rather than KMS analysis
E2E lightweight authentication [21]	Resource-constrained IoT	No	Limited	End-to-end encryption and lightweight mutual authentication	Security and performance analysis	Authentication-oriented and not designed for hierarchical KMSs
eSKAMI [22]	Smart grid AMI	No	Yes	Key graph structure	Analysis and simulation	Designed for AMI settings
SAMI [23]	Smart grid AMI	No	Yes	Key graph structure	Analysis and simulation	Designed for AMI settings
iVerSAMI [24]	Smart grid AMI	No	Yes	Multi-group key graphs	Analysis and simulation	Less suitable for highly dynamic IoT systems
KTLH [25]	Linear hierarchies	Yes	Yes	Hash chains and key tables	Analytical and complexity analysis	Key table management overhead
LHSC [26]	Linear hierarchies	Yes	Yes	Public class parameters	Analytical and complexity analysis	Higher computation and communication overheads
DKM [27]	Linear hierarchies	Yes	Yes	Dependent keys and local subtree updates	Analytical analysis	Limited evaluation under IoT churn
Lattice-based KMS [28]	IoT data management	No	Limited	Lattice-based lightweight key management	Security and performance analysis	Not dedicated to dynamic linear hierarchies

Note: Secure and Efficient Key Management Scheme for Internet of Things (SEKM-IoT), Physically Unclonable Functions (PUF).

3. BACKGROUND

This section presents the background required for the development of the proposed framework. First, we describe the dynamic linear-hierarchy IoT model considered in this paper. Then, we present the LKH mechanism used to support efficient intra-class rekeying, followed by the SEKM-IoT scheme adopted as a representative KMS for dynamic linear-hierarchy IoT systems. Finally, we review the main concepts related to Markov chains and Poisson processes that will be used in the stochastic modeling.

3.1 Dynamic linear-hierarchy Internet of Things model

In this paper, we consider an IoT system organized according to a dynamic linear hierarchy of security classes. Let

$(C_1, C_2, \dots, C_N)$ denote the set of classes, where C_1 is the highest class and C_N is the lowest one. Each class C_j contains a set of members sharing the same access privilege level, and is associated with a class key CK_j used to secure intra-class communications.

The considered hierarchy follows a linear access-control structure. More precisely, a member belonging to a higher class is authorized to access the data and communications of lower classes, whereas a member of a lower class is not allowed to access the information of upper classes. Therefore, the key management process must preserve both local secure communication inside each class and hierarchical accessibility across classes.

The system is assumed to be dynamic. Indeed, members may join or leave their classes over time because of mobility,

operational changes, maintenance activities, or connectivity conditions. These membership changes require the corresponding class keys to be updated in order to preserve forward secrecy and backward secrecy. In addition, when a class key is updated, the key accessibility relation between the affected class and the lower classes must remain consistent with the hierarchy.

A KS is responsible for managing the cryptographic material of the system. More precisely, the KS generates the individual keys and class keys, distributes them securely to the legitimate members, and performs the required rekeying operations whenever a membership change occurs. In this way, the KS ensures that the hierarchical access-control policy remains correctly enforced over time.

Figure 1 illustrates the dynamic linear-hierarchy IoT model considered in this paper.

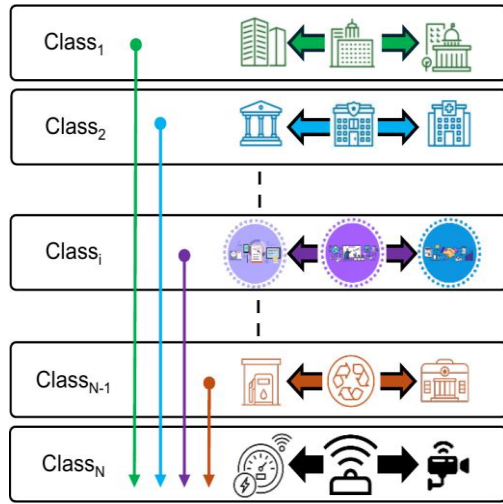


Figure 1. Linear-hierarchy Internet of Things (IoT) model

3.2 Logical Key Hierarchy

In hierarchical IoT systems, secure intra-class communication is a fundamental requirement. Since members of the same class need to exchange data securely, they must share a common class key. This class key must be updated whenever a member joins or leaves the class in order to preserve important security properties such as forward secrecy and backward secrecy. One of the most relevant and widely used mechanisms for handling such updates efficiently is the LKH protocol.

LKH is a tree-based group key management mechanism managed by a KS. In this structure, the root node represents the class key, while the leaf nodes correspond to the individual keys of the members. The internal nodes store intermediate

cryptographic keys that are used as key encryption keys in the rekeying process. This structure enables efficient multicast-based key updates and avoids the need for unicast transmission to all group members.

To explain how LKH works, let us consider a key tree composed of seven members, $\{m_1, m_2, \dots, m_7\}$, as shown in Figure 2(a). In this tree, the member m_2 stores the set of keys located on the path from its leaf to the root, namely $\{k_2, k_{12}, k_{14}, CK\}$. When a membership change occurs, only the keys located on the affected path need to be updated. Therefore, the communication and computation overhead of the rekeying process can be significantly reduced compared with flat group key management approaches.

3.2.1 Member join scenario

Let us consider the scenario shown in Figure 2(b), where a new member m_8 joins the group. First, the KS establishes a secure channel with m_8 and shares with it an individual key k_8 . Then, in order to preserve backward secrecy, the KS updates the keys located on the affected path and generates the new keys $\{k_{78}, k_{58}, CK'\}$. These keys are then distributed securely as follows:

$$\begin{aligned} KS &\rightarrow \{m_7\}: \text{Enc}(k_{78}, k_7), \text{Enc}(k_{58}, k_{78}), \\ KS &\rightarrow \{m_5, m_6\}: \text{Enc}(k_{58}, k_{56}), \\ KS &\rightarrow \{m_5, m_6, m_7\}: \text{Enc}(CK', k_{58}), \\ KS &\rightarrow \{m_1, m_2, m_3, m_4\}: \text{Enc}(CK', k_{14}). \end{aligned}$$

3.2.2 Member leave scenario

Now, let us consider the leave scenario depicted in Figure 2(c), where the member m_3 leaves the group. In this case, the key k_{34} is removed, and the keys $\{k_{14}, CK'\}$ are updated to $\{k'_{14}, CK''\}$ in order to preserve forward secrecy. The updated keys are distributed as follows:

$$\begin{aligned} KS &\rightarrow \{m_4\}: \text{Enc}(k'_{14}, k_4), \\ KS &\rightarrow \{m_1, m_2\}: \text{Enc}(k'_{14}, k_{12}), \\ KS &\rightarrow \{m_1, m_2, m_4\}: \text{Enc}(CK'', k'_{14}), \\ KS &\rightarrow \{m_5, m_6, m_7, m_8\}: \text{Enc}(CK'', k_{58}). \end{aligned}$$

A major advantage of LKH is that it naturally supports multicast rekeying. Instead of sending the new class key separately to each member, KS can encrypt it with an intermediate key shared by a subgroup of members and transmit it once to all of them. In this way, LKH reduces the communication overhead from a linear cost to a logarithmic one. More precisely, for a balanced binary tree containing n members, the communication and computation overheads of rekeying are of order $O(\log n)$, and each member stores at most $1 + \log_2(n)$ keys.

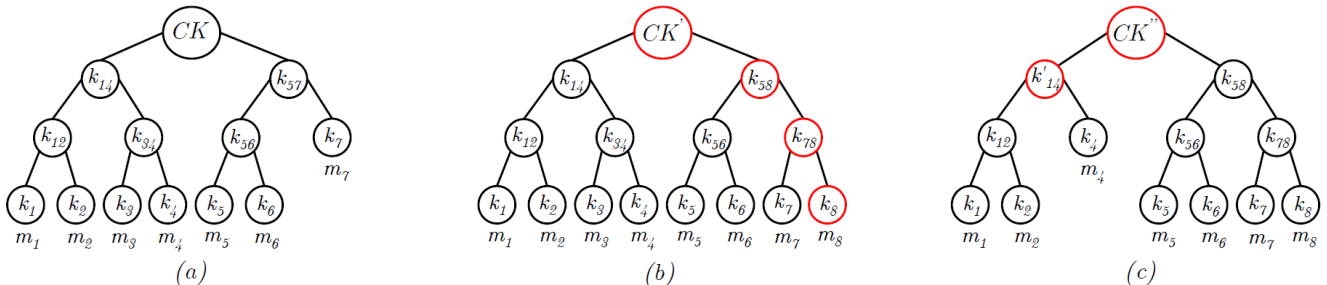


Figure 2. Logical Key Hierarchy (LKH) scheme: (a) LKH key tree with seven members, (b) LKH key tree when m_8 joins, and (c) LKH key tree when m_3 leaves

3.3 Secure and Efficient Key Management Scheme for Internet of Things scheme

SEKM-IoT [9] was proposed to support secure and efficient key management in dynamic linear-hierarchy IoT systems. The scheme considers a set of ordered security classes $(C_1, C_2, \dots, C_N)$, where C_1 is the highest class and C_N is the lowest one. In such a hierarchy, each class must be able to communicate securely within itself, while higher classes must also be able to access the keys of lower classes. For this reason, SEKM-IoT combines two complementary mechanisms: an LKH-based structure for intra-class key management and a hash-based derivation process for inter-class accessibility.

More precisely, each class maintains its own LKH key tree. The root of this tree is the class key, while the leaves correspond to the individual keys of the class members. Therefore, if a member belongs to class C_j , it stores its own individual key together with the keys located on the path from its leaf to the root. In this way, when a member joins or leaves class C_j , the LKH rekeying process is applied locally inside this class in order to generate a new version of the class key efficiently.

At the inter-class level, SEKM-IoT relates the class keys through a one-way hash chain. Starting from the class key of an upper class, the key of the next lower class is obtained by applying the hash function once. Thus, the class keys satisfy the relation.

$$CK_{i+1} = \mathcal{H}(CK_i).$$

Consequently, a member of class C_j can derive the keys of lower classes by applying the hash function repeatedly, whereas a member of a lower class cannot derive the key of an upper class because of the one-way property of the hash function.

Initially, the KS establishes an individual key with each member through a secure key exchange process. Then, it builds the LKH tree of each class and assigns the corresponding class key to its root. Once a member receives its class key, it can derive the keys of lower classes whenever authorized. Since class keys are updated after membership changes, we denote by CK_j^p the class key of class C_j after $(p - 1)$ updates, where CK_j^1 is the initial version.

When a member m_i^j joins or leaves class C_j , a rekeying process is triggered. This process affects not only class C_j , but also all lower classes C_l , where $C_l \leq C_j$. Otherwise, a joining member could access old traffic of lower classes, or a leaving member could retain access to future traffic. Therefore, SEKM-IoT updates the key of the affected class and propagates the update to the lower classes accordingly. The rekeying process of SEKM-IoT can be summarized as follows:

Step 1: The member m_i^j is inserted into or removed from the LKH key tree of class C_j , and the LKH rekeying process generates a new class key CK_j^{p+1} .

Step 2: The KS computes the new keys of all lower classes C_l , where $C_M \leq C_l < C_j$, by applying the hash function recursively: $CK_l^{p+1} = \mathcal{H}(CK_{l-1}^{p+1})$.

Step 3: The KS securely distributes each newly generated class key to the corresponding members of the lower classes by using the appropriate shared keys of their LKH trees. In

practice, the keys associated with the direct children of the root are used to ensure efficient and secure multicast distribution.

Step 4: The KS sends an update message to each upper class C_k , where $C_j < C_k$, in order to notify them of the new version of CK_j . More precisely, the pair $P_j = (j, CK_j^{p+1})$ is encrypted with the class key of the corresponding upper class and sent securely to its members.

Step 5: Upon receiving the update message, the members of the upper classes update their class-keys table so that subsequent key derivations remain synchronized with the new hierarchy state.

The role of the update message is particularly important. Indeed, when the key of class C_j is modified, the members of higher classes cannot derive the new version of CK_j directly from their previously stored information, because the new key is generated through the local LKH rekeying process and not directly from the previous class key. For this reason, each member maintains a class-keys table that is updated whenever such a notification is received. When the member needs to derive the key of a lower class, it consults this table and starts the derivation from the most recent available class key.

3.3.1 Example of rekeying in Secure and Efficient Key Management Scheme for Internet of Things

Let us consider the example of a smart city IoT infrastructure composed of five classes ($N = 5$). Assume that the member m_8^3 leaves class C_3 . As shown in Figure 3, the KS first applies the LKH rekeying process to the key tree of class C_3 . As a result, the key k_{78}^3 is deleted and the keys $\{k_{58}^3, CK_3^1\}$ are updated to $\{k_{58}^{13}, CK_3^2\}$. The updated keys are then distributed securely as follows:

$$\begin{aligned} \text{KS} &\rightarrow \{m_7^3\}: \text{Enc}(k_{58}^{13}, k_7^3), \\ \text{KS} &\rightarrow \{m_5^3, m_6^3\}: \text{Enc}(k_{58}^{13}, k_{56}^3), \\ \text{KS} &\rightarrow \{m_5^2, m_6^2, m_7^2\}: \text{Enc}(CK_3^2, k_{58}^{13}), \\ \text{KS} &\rightarrow \{m_1^3, m_2^3, m_3^3, m_4^3\}: \text{Enc}(CK_3^2, k_{14}^3). \end{aligned}$$

Then, the keys of the lower classes C_4 and C_5 are recomputed from the updated key CK_3^2 as follows:

$$CK_4^2 = \mathcal{H}(CK_3^2), CK_5^2 = \mathcal{H}(CK_4^2).$$

These new keys are then securely transmitted to the corresponding members of classes C_4 and C_5 by using the appropriate shared keys of their LKH trees.

In addition, the pair $P_3 = (3, CK_3^2)$ is securely sent to the members of the upper classes C_1 and C_2 , so that they can update their class-keys tables accordingly. At the end of the process, the members of upper classes can still derive the keys of lower classes correctly by using their updated tables. For instance, a member of C_1 can use the updated entry associated with class C_3 , and then apply the hash function repeatedly in order to derive the keys of classes C_4 and C_5 .

From the above, it can be observed that SEKM-IoT provides an interesting trade-off between security and efficiency. On the one hand, it uses LKH to support efficient intra-class rekeying. On the other hand, it preserves hierarchical access control by linking the class keys through a one-way hash-based derivation process. For this reason, SEKM-IoT constitutes a representative case study for the stochastic framework proposed in this paper.

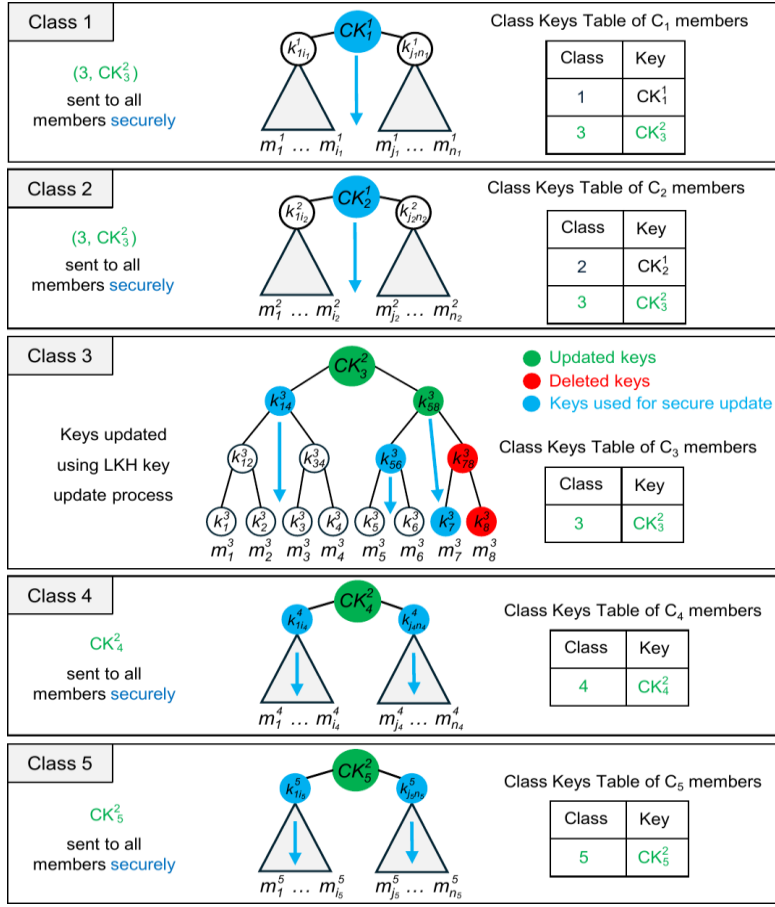


Figure 3. Example of rekeying process in Secure and Efficient Key Management Scheme for Internet of Things (SEKM-IoT) [9]

3.4 Markov chains

Stochastic processes provide the mathematical framework for modeling systems whose behavior evolves over time in a probabilistic manner. Among these processes, Markov chains constitute an important class of models in which the future evolution of the system depends only on its current state and not on the past history. This characteristic is known as the Markov property [34].

Let X_t denote the state of the system at time t , and let $S = \{s_1, s_2, \dots, s_n\}$ be the state space. The Markov property is expressed as follows:

$$\begin{aligned} P(X_{t+1} = s_j | X_t = s_i, X_{t-1}, \dots, X_0) \\ = P(X_{t+1} = s_j | X_t = s_i) \end{aligned} \quad (1)$$

In the discrete-time setting, the evolution of the system is characterized by a transition probability matrix $P = [p_{i,j}]$, where each entry $p_{i,j}$ denotes the probability of moving from state s_i to state s_j during one observation interval. Therefore,

$$p_{i,j} = P(X_{t+1} = s_j | X_t = s_i) \quad (2)$$

with

$$\sum_{j \in S} p_{i,j} = 1, \forall i \in S \quad (3)$$

Let $\pi_t = (\pi_t(1), \dots, \pi_t(n))$ denote the state probability distribution at time t , where $\pi_t(i) = P(X_t = s_i)$. Then, the evolution of the distribution over time is governed by the

transition matrix P .

A Markov chain is said to be homogeneous when its transition probabilities do not depend explicitly on time. It is said to be irreducible when every state can be reached from every other state through a sequence of transitions with non-zero probability. These properties are important because they determine the existence and uniqueness of the long-term equilibrium distribution.

The long-term behavior of the system is described by the stationary distribution π , which satisfies

$$\pi P = \pi, \sum_{i \in S} \pi(i) = 1 \quad (4)$$

If the chain is irreducible and aperiodic and the state space is finite, then the stationary distribution exists and is unique. In that case, it characterizes the asymptotic proportion of time spent in each state. This property is particularly useful in the present work, since the proposed framework aims at evaluating the long-term behavior of KMSs under stochastic membership dynamics.

3.5 Poisson processes

To model the occurrence of random events over time, such as member joins and leaves, we rely on the Poisson process [35]. A Poisson process is a stochastic process used to represent the number of discrete events occurring during a given time interval, under the assumption that these events occur independently and at a constant average rate.

Let $N(t)$ denote the number of events occurring during the interval $[0, t]$, and let $\lambda > 0$ be the average event rate. Then,

the probability of observing exactly k events during the time interval t is given by

$$P(N(t) = k) = \frac{(\lambda t)^k}{k!} e^{-\lambda t} \quad (5)$$

A fundamental property of the Poisson process is that the numbers of events occurring in disjoint time intervals are independent. In addition, the inter-arrival times between consecutive events are exponentially distributed with parameter λ . This memoryless property makes the Poisson process particularly suitable for modeling random and uncorrelated membership changes in dynamic IoT systems.

In the context of this paper, the Poisson process is used to model the stochastic occurrence of join and leave events during a given observation interval. These event counts are then used to define the transition probabilities of the Markov chains representing the behavior of the considered KMS. Combined with the Markov-chain formulation, this process provides the basis for analyzing the long-term impact of membership dynamics on the storage, communication, and computation overheads.

4. MARKOV MODELING AND STEADY-STATE ANALYSIS

In this section, we model the behavior of SEKM-IoT by

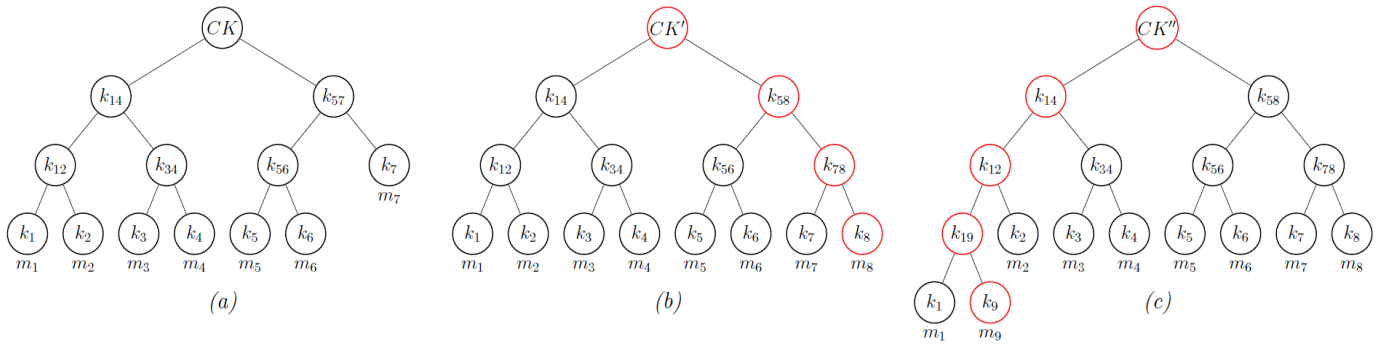


Figure 4. Non-Markovian behavior of tree depth in a growing Logical Key Hierarchy (LKH) tree

We first examine whether the process $D(C_j)$ can itself be modeled as a Markov chain. For this purpose, we recall that a process satisfies the Markov property if the knowledge of the current state is sufficient to determine the probabilistic evolution of the next state, that is,

$$P(X_{t+1} | X_t, X_{t-1}, \dots, X_0) = P(X_{t+1} | X_t) \quad (6)$$

However, in the present case, the depth of the LKH tree does not evolve solely according to its current value. It also depends on the detailed structure of the tree, which is itself determined by the history of previous join and leave events. To illustrate this point, let us consider the example shown in Figure 4(a), where the depth of the tree is $D(C_j) = 3$.

When a new member m_8 joins the class, the system moves to the state shown in Figure 4(b), and the depth remains equal to $D(C_j) = 3$. However, if another new member m_9 joins, the tree evolves to the state shown in Figure 4(c), and the depth becomes $D(C_j) = 4$.

This example shows that the current value $D(C_j) = 3$ is not

means of Markov processes in order to evaluate its main performance metrics. More precisely, we construct stochastic models for the evolution of the hierarchical continuity between classes and for the key distribution behavior inside each class. Then, by using the stationary distribution of the corresponding Markov chains, we derive the expected long-term values of the considered metrics. As highlighted previously in Section 3.4, the stationary distribution characterizes the long-term behavior of the system when it evolves over time and reaches equilibrium.

The proposed stochastic analysis follows three steps. First, a state variable is selected to capture the relevant key-management condition. Second, membership events are mapped to state transitions and their rates. Third, the stationary distribution of the resulting process is used to compute the long-term overhead metric. This procedure makes the modeling approach explicit and can be reused when studying related dynamic hierarchical KMSs.

4.1 A non-Markovian description based on tree depth

Let us consider a hierarchy composed of N classes, and let C_j denote a given class in this hierarchy. Let $D(C_j)$ represent the depth of the LKH tree associated with class C_j . If a member m_i^j belongs to this class, then it stores its individual key k_i^j together with the keys corresponding to the nodes located on the path from its leaf to the class key.

sufficient by itself to determine the probabilistic behavior of the next state. Indeed, two different tree configurations may have the same depth but may evolve differently after the next membership event. Therefore, the process based only on tree depth does not satisfy the Markov property, and it cannot be used directly as a Markov chain state variable.

4.2 Markov processes for Key Management Scheme evaluation

To evaluate the performance of the considered KMS, we introduce two Markov processes. The first one models the inter-class continuity relation induced by the hash-based derivation mechanism. The second one models the intra-class key distribution behavior induced by the LKH structure. These two processes are used to evaluate the following metrics:

Storage overhead (stor_i^j): this metric represents the number of keys stored by a member m_i^j . Each member stores both the class-related keys and the keys assigned through the LKH

structure. If $\mathcal{K}_{C_j}$ denotes the set of class-related keys available to m_i^j , and $\mathcal{K}_{m_i^j}$ denotes the set of local LKH keys stored by this member, then the total storage overhead is given by

$$|\mathcal{K}_{m_i^j}^{\text{total}}| = |\mathcal{K}_{C_j}| + |\mathcal{K}_{m_i^j}| \quad (7)$$

Computation overhead (comp_i^j): this metric is divided into two parts. At the user side, it represents the number of decryption and hash operations carried out by the member m_i^j . At the server side, it represents the number of encryption and key update operations performed by the KS during rekeying. The value of this metric depends on both the continuity process and the position of the member in the LKH tree.

Communication overhead (comm_i^j): this metric quantifies the number of key update messages exchanged during rekeying. It depends on the number of keys sent inside the class and on the number of update messages exchanged across classes.

To evaluate these metrics, we define the KCP for inter-class behavior and the CKDP for intra-class behavior.

4.2.1 Key Continuity Process

Let us consider a hierarchy composed of N classes ($C_1, C_2, \dots, C_N$), ordered from the highest privilege level to the lowest one. In SEKM-IoT, class keys are related through the one-way hash function $\mathcal{H}$, so that a higher class can derive the keys of lower classes.

For two classes C_i and C_j , with $i < j$, the continuity relation between them is defined by the possibility of deriving CK_j from CK_i by successive applications of $\mathcal{H}$, namely,

$$CK_j = \mathcal{H}^{j-i}(CK_i) \quad (8)$$

For a given class C_j , we define the KCP $\text{Ct}(C_j)$ as the number of consecutive classes, starting from C_j , that remain linked through the hash-based derivation relation. In other words, $\text{Ct}(C_j)$ is the largest integer r such that

$$CK_{j+\ell} = \mathcal{H}^\ell(CK_j), \forall \ell \in \{1, \dots, r-1\}, \quad (9)$$

whenever $j + \ell \leq N$. Therefore, $\text{Ct}(C_j)$ represents the length of the continuity chain starting from class C_j . Its possible values are

$$\text{Ct}(C_j) \in \{1, 2, \dots, N - j + 1\}. \quad (10)$$

The minimum value is $\text{Ct}(C_j) = 1$, which means that the continuity chain is broken immediately after C_j . The maximum value is $\text{Ct}(C_j) = N - j + 1$, which means that the continuity extends from C_j down to the lowest class C_N .

Figure 5 illustrates the continuity process. Unlike the tree depth process discussed previously, $\text{Ct}(C_j)$ can be modeled as a Markov chain because its future evolution depends only on its current continuity length and on the next membership event that affects the hierarchy.

For each class C_j , we define a discrete-time Markov chain whose state space is

$$\mathcal{S}_j = \{1, 2, \dots, N - j + 1\}.$$

The transition probability matrix associated with this chain is denoted by $P^{(j)} = [p_{\alpha, \beta}^{(j)}]$, where $p_{\alpha, \beta}^{(j)}$ is the probability of moving from continuity state α to continuity state β during one observation interval Δt .

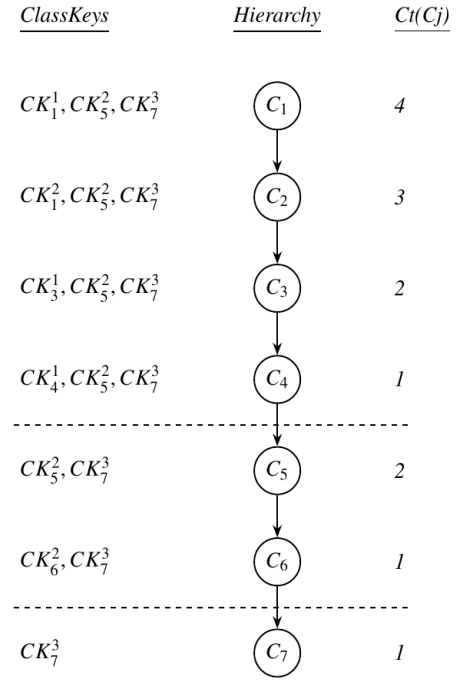


Figure 5. Key Continuity Process (KCP)

We assume that join and leave events affecting class C_i follow independent Poisson processes with rates $\lambda_{j,i}$ and $\lambda_{L,i}$, respectively. The setting $\lambda_{j,i} = \lambda_j$ and $\lambda_{L,i} = \lambda_L$ for all i is a particular case of this formulation. Thus, the aggregate update rate of class C_i is

$$\lambda_i^{\text{upd}} = \lambda_{j,i} + \lambda_{L,i} \quad (11)$$

For a KCP starting from class C_j , let

$$L_j = N - j + 1 \quad (12)$$

denote the maximum continuity length. Consider that the current KCP state is $\alpha \in \{1, \dots, L_j\}$.

A membership update in a class C_i with $i \leq j$ refreshes the key of C_i and propagates new keys to all lower classes. Consequently, the continuity chain starting from C_j is restored to its maximum length L_j . The total rate of such restoring events is

$$\Lambda_{1:j}^{\text{upd}} = \sum_{i=1}^j \lambda_i^{\text{upd}} \quad (13)$$

In contrast, an update in a lower class C_{j+r} , with $1 \leq r < \alpha$, breaks the current continuity chain after class C_{j+r-1} . Therefore, the KCP moves from state α to state r . Updates in classes below the first already-broken position do not modify the current KCP state.

These transition rules define a continuous-time Markov

chain with the matrix $\mathbf{Q}^{(j)} = [q_{\alpha,\beta}^{(j)}]$. Its off-diagonal entries are given by

$$q_{\alpha,\beta}^{(j)} = \begin{cases} \lambda_{j+\beta}^{\text{upd}}, & 1 \leq \beta < \alpha \\ \lambda_{1:j}^{\text{upd}}, & \beta = L_j \text{ and } \alpha < L_j \\ 0, & \text{otherwise} \end{cases} \quad (14)$$

The diagonal entries are computed as

$$q_{\alpha,\alpha}^{(j)} = - \sum_{\beta \neq \alpha} q_{\alpha,\beta}^{(j)} \quad (15)$$

The discrete-time transition matrix over an observation interval Δt is then obtained as

$$\mathbf{P}^{(j)} = \exp(\mathbf{Q}^{(j)} \Delta t) \quad (16)$$

This expression accounts for all possible event sequences during Δt , including multiple join and leave events occurring in different classes.

By construction, $\mathbf{P}^{(j)}$ is a stochastic matrix. If the chain is irreducible and aperiodic, it admits a unique stationary distribution

$$\pi^{(j)} = (\pi_1^{(j)}, \pi_2^{(j)}, \dots, \pi_{L_j}^{(j)}),$$

which satisfies

$$\pi^{(j)} \mathbf{P}^{(j)} = \pi^{(j)}, \sum_{\alpha=1}^{L_j} \pi_{\alpha}^{(j)} = 1 \quad (17)$$

Accordingly, the expected continuity value of class C_j in steady state is given by

$$\mathbb{E}[\text{Ct}(C_j)] = \sum_{\alpha=1}^{L_j} \alpha \pi_{\alpha}^{(j)} \quad (18)$$

This expected value provides a quantitative measure of the long-term inter-class continuity maintained by the KMS.

4.2.2 Class Key Distribution Process

The CKDP models the intra-class evolution of the LKH structure. To ensure that the Markov state fully captures the effect of membership changes, we use the current population of class C_j as the state variable. The number of local keys stored by a representative active member is then derived from the corresponding LKH-tree depth.

Let $X_t^{(j)} \in \{1, 2, \dots, M\}$ denote the number of members in class C_j at time t , where M is the maximum class size. The process is considered from the perspective of a representative member that remains active in the class. Hence, leave events correspond to the departure of other members of the same class.

The process $\{X_t^{(j)}\}$ is modeled as a finite birth-death Markov chain. A join event increases the class population by one, whereas a leave event decreases it by one. The infinitesimal generator matrix $\mathbf{B}^{(j)} = [b_{n,m}^{(j)}]$ is defined by

$$b_{n,m}^{(j)} = \begin{cases} \lambda_{j,j}, & m = n + 1, 1 \leq n < M \\ \lambda_{L,j}, & m = n - 1, 1 < n \leq M \\ - \sum_{r \neq n} b_{n,r}^{(j)}, & m = n \\ 0, & \text{otherwise} \end{cases} \quad (19)$$

At the lower boundary $n = 1$, no further leave transition is possible because the representative member remains active. At the upper boundary $n = M$, no additional join transition is admitted. The discrete-time transition matrix over the observation interval Δt is

$$\mathbf{R}^{(j)} = \exp(\mathbf{B}^{(j)} \Delta t) \quad (20)$$

Let $\nu^{(j)} = (\nu_1^{(j)}, \nu_2^{(j)}, \dots, \nu_M^{(j)})$ denote the stationary distribution of the class population. It satisfies

$$\nu^{(j)} \mathbf{R}^{(j)} = \nu^{(j)}, \sum_{n=1}^M \nu_n^{(j)} = 1 \quad (21)$$

For a balanced d -ary LKH tree containing n members, the number of local keys stored by a representative member is

$$g_d(n) = 1 + \lceil \log_d(n) \rceil \quad (22)$$

where, the first term represents the individual key and the remaining terms represent the keys located on the path from the leaf node to the class-key root.

Therefore, the long-term expected number of local LKH keys stored by a member of class C_j is

$$\mathbb{E}[K^{(j)}] = \sum_{n=1}^M g_d(n) \nu_n^{(j)} \quad (23)$$

Finally, by combining (7), (18), and (23), the long-term storage overhead of a member can be derived. The same modeling principle can also be used to evaluate the long-term computation and communication overheads induced by the KMS.

4.2.3 Illustrative construction of transition matrices

This example is provided only to illustrate the transition-matrix construction. The numerical values used here are not the parameters of the evaluation presented in Section 5.

Consider a hierarchy with $N = 3$ classes and assume that

$$\lambda_{j,i} = \lambda_{L,i} = 0.05 \text{ s}^{-1}, \forall i \in \{1, 2, 3\} \quad (24)$$

Hence, the aggregate update rate is $\lambda_i^{\text{upd}} = 0.10 \text{ s}^{-1}$ for every class. Let the observation interval be $\Delta t = 1 \text{ s}$.

Consider the KCP associated with C_1 . Its state space is $\mathcal{S}_1 = \{1, 2, 3\}$. State 1 indicates that the continuity is broken immediately after C_1 , state 2 indicates that C_1 and C_2 remain continuous, and state 3 indicates that the continuity extends from C_1 to C_3 .

Using Eq. (14), the generator matrix is

$$\mathbf{Q}^{(1)} = \begin{bmatrix} -0.10 & 0 & 0.10 \\ 0.10 & -0.20 & 0.10 \\ 0.10 & 0.10 & -0.20 \end{bmatrix} \quad (25)$$

The corresponding discrete-time transition matrix is

$$\begin{aligned} \mathbf{P}^{(1)} &= \exp(\mathbf{Q}^{(1)}\Delta t) \\ &= \begin{bmatrix} 0.9094 & 0.0042 & 0.0864 \\ 0.0906 & 0.8230 & 0.0864 \\ 0.0906 & 0.0822 & 0.8272 \end{bmatrix} \end{aligned} \quad (26)$$

The stationary distribution is

$$\pi^{(1)} = [0.5000, 0.1667, 0.3333] \quad (27)$$

which yields

$$\begin{aligned} \mathbb{E}[\text{Ct}(C_1)] &= 1(0.5000) + 2(0.1667) + 3(0.3333) \\ &= 1.8333 \end{aligned} \quad (28)$$

Consider a binary LKH tree with $d = 2$ and a maximum class size of $M = 4$. The corresponding CKDP generator matrix is

$$\mathbf{B}^{(j)} = \begin{bmatrix} -0.05 & 0.05 & 0 & 0 \\ 0.05 & -0.10 & 0.05 & 0 \\ 0 & 0.05 & -0.10 & 0.05 \\ 0 & 0 & 0.05 & -0.05 \end{bmatrix} \quad (29)$$

Using Eq. (20), the corresponding transition matrix over $\Delta t = 1$ s is

$$\mathbf{R}^{(j)} = \begin{bmatrix} 0.9524 & 0.0464 & 0.0012 & 0.0000 \\ 0.0464 & 0.9071 & 0.0453 & 0.0012 \\ 0.0012 & 0.0453 & 0.9071 & 0.0464 \\ 0.0000 & 0.0012 & 0.0464 & 0.9524 \end{bmatrix} \quad (30)$$

For this symmetric illustrative configuration, the stationary distribution is

$$\nu^{(j)} = [0.25, 0.25, 0.25, 0.25] \quad (31)$$

Since $g_2(1) = 1$, $g_2(2) = 2$, $g_2(3) = 3$, and $g_2(4) = 3$, the expected number of locally stored keys is

$$\begin{aligned} \mathbb{E}[K^{(j)}] &= \\ 0.25(1) + 0.25(2) + 0.25(3) + 0.25(3) &= 2.25 \end{aligned} \quad (32)$$

5. RESULTS AND DISCUSSION

In this section, we present the stationary results obtained from the KCP and CKDP models introduced in Section 4. All numerical settings used in the evaluation are summarized in Table 2. The baseline configuration considers homogeneous membership dynamics across classes in order to isolate the effect of the hierarchy size and the class size on the considered overheads.

Table 2. Baseline parameter settings used for the stationary evaluation

Parameter	Baseline Value	Description
Membership model	Independent Poisson processes	Join and leave events are independent across classes.
Join rate λ_j	0.05 s^{-1} per class	Homogeneous for all classes
Leave rate λ_L	0.05 s^{-1} per class	Homogeneous for all classes
Aggregate update rate λ^{upd}	0.10 s^{-1} per class	$\lambda^{\text{upd}} = \lambda_j + \lambda_L$
Observation interval Δt	1s	Used to construct the discrete-time transition matrices
Hierarchy sizes N	{10, 50, 100, 150, 200, 250, 300, 350, 400}	Number of security classes in the KCP evaluation
Starting KCP classes C_j	$j = 1, \dots, N$	All starting classes are included in the reported mean
Maximum class size M	1, ..., 400	Range considered in the CKDP evaluation
LKH-tree degree d	2	Balanced binary LKH tree
CKDP boundary condition	Reflecting boundaries	Population states are bounded between 1 and M
Stationary computation	Balance equations	$\pi \mathbf{Q} = 0$ and $\nu \mathbf{B} = 0$ with normalization

Note: Logical Key Hierarchy (LKH), Key Continuity Process (KCP), Class Key Distribution Process (CKDP).

5.1 Evaluation settings

Each class is assumed to experience independent Poisson join and leave processes with identical rates. The baseline choice $\lambda_j = \lambda_L = 0.05 \text{ s}^{-1}$ corresponds to an aggregate membership update rate of 0.10 s^{-1} per class. This balanced configuration avoids systematic population growth or shrinkage and provides a reference setting for the stationary analysis.

For the KCP, the stationary continuity is evaluated for all starting classes C_j , with $j \in \{1, \dots, N\}$, and the reported hierarchy-level result is the mean continuity over all possible starting classes. For the CKDP, the maximum class size varies from $M = 1$ to $M = 400$, and a binary LKH tree is considered. All reported values correspond to steady-state quantities; therefore, they do not depend on an arbitrary initial state.

5.2 Event-driven simulation validation

To validate the stationary analysis, we developed an

independent event-driven Monte Carlo simulator. The simulator directly generates the Poisson-driven membership-update process and applies the corresponding update rules in continuous time. It does not use the transition matrices or the stationary solvers of the analytical framework. Therefore, this validation verifies that the analytical Markov construction reproduces the long-run behavior of the underlying event dynamics under the considered modeling assumptions.

For the KCP, we used 30 independent replications, with a warm-up period of 200 s and an observation period of 1000 s. For the CKDP, 50 independent replications were performed with a warm-up period and an observation period of 5×10^5 s, respectively. Table 3 compares the analytical stationary values with the simulation estimates. All analytical values lie within the corresponding 95% confidence intervals, and the maximum absolute relative deviation is below 1%.

All numerical results were implemented in Python using the parameter settings reported in Table 2. For the event-driven Monte Carlo validation, a fixed random seed of 20260630 was used.

Table 3. Validation of the stationary analytical results using independent event-driven Monte Carlo simulations. The simulation values are reported as the mean $\pm$ 95% confidence-interval half-width across independent replications

Process	Configuration	Analytical	Simulation	Absolute Deviation
KCP	$N = 10$	3.2500	3.2504 ± 0.0147	0.01%
KCP	$N = 50$	13.2500	13.2452 ± 0.0122	0.04%
KCP	$N = 100$	25.7500	25.7471 ± 0.0123	0.01%
CKDP	$M = 25$	4.4023	4.3940 ± 0.0232	0.19%
CKDP	$M = 50$	5.3418	5.3754 ± 0.0391	0.63%
CKDP	$M = 100$	6.3066	6.3643 ± 0.0880	0.91%

Note: Key Continuity Process (KCP), Class Key Distribution Process (CKDP).

5.3 Storage overhead

According to Eq. (7), the storage overhead of a member is the sum of the keys induced by the inter-class continuity and the local keys induced by the LKH structure. Therefore, the evaluation of the storage overhead requires the analysis of both $|\mathcal{K}_{C_j}|$ and $|\mathcal{K}_{m_i}|$.

To evaluate the class-related storage component $|\mathcal{K}_{C_j}|$, we solve the stationary KCP for each hierarchy size N and for every starting class C_j . To summarize the behavior of an entire hierarchy, we define the system-wide mean continuity as

$$\bar{C}(N) = \frac{1}{N} \sum_{j=1}^N \mathbb{E}[C_t(C_j)] \quad (33)$$

The mean continuity is evaluated for the nine hierarchy sizes reported in Table 2. To quantify its dependence on the hierarchy size, we fit the linear regression model using ordinary least squares. The fitting quality is assessed using the coefficient of determination R^2 , the mean absolute error (MAE), and the root mean squared error (RMSE).

$$\hat{\bar{C}}(N) = aN + b \quad (34)$$

The resulting regression is

$$\hat{\bar{C}}(N) = 0.2500N + 0.7500, \quad R^2 = 1.0000 \quad (35)$$

The corresponding fitting errors are $\text{MAE} = 1.41 \times 10^{-14}$ and $\text{RMSE} = 1.41 \times 10^{-14}$. These residual values are at the numerical round-off level. This exact agreement results from the homogeneous and symmetric membership-update configuration considered in the baseline analysis. Therefore, under this configuration, the system-wide mean continuity grows linearly with the hierarchy size:

$$\bar{C}(N) \sim \mathcal{O}(N) \quad (36)$$

Figure 6 illustrates this behavior by showing the system-wide steady-state mean continuity as a function of the number of security classes. The results confirm the linear increase predicted by Eq. (36) under the homogeneous baseline configuration.

The continuity metric reported above characterizes the average residual continuity observed from a starting class. However, the class-related storage overhead is determined by the number of continuity segments rather than by the continuity length alone.

Let $\mathcal{H}_j = (C_j, C_{j+1}, \dots, C_N)$ denote the suffix of the

hierarchy accessible from class C_j , and let S_j denote the number of maximal continuity segments in this suffix. Each segment requires one anchor key. Starting from this anchor, all keys inside the segment can be derived successively through the hash function. Therefore, each continuity break requires one additional anchor key in the class-keys table.

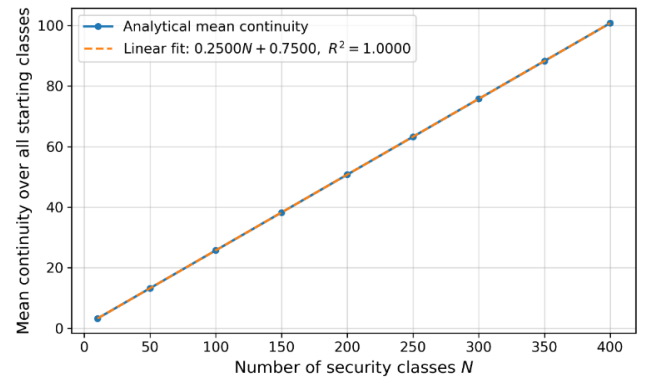


Figure 6. System-wide steady-state mean continuity $\bar{C}(N)$ versus the number of security classes under the baseline configuration $\lambda_j = \lambda_L = 0.05 \text{ s}^{-1}$ and $\Delta t = 1 \text{ s}$

For example, if the hierarchy contains continuity breaks between C_3 and C_4 , and between C_6 and C_7 , then a member of C_1 stores anchor keys for the segments (C_1, C_2, C_3) , (C_4, C_5, C_6) , and $(C_7, \dots, C_N)$. The remaining class keys are obtained through repeated hash applications. Consequently, up to an additive constant associated with the class key already stored in the local LKH tree,

$$|\mathcal{K}_{C_j}| = \theta(S_j) \quad (37)$$

To derive the expected number of segments, let I_k be the indicator of a continuity break before class C_k , for $k > j$. Such a break is present when the most recent membership update among classes $C_1, \dots, C_k$ occurred in C_k . Under the homogeneous baseline configuration, the update processes are independent and have identical rates. Therefore, each of the k classes is equally likely to contain the most recent update, which gives

$$\Pr(I_k = 1) = \frac{1}{k} \quad (38)$$

The number of continuity segments accessible from class C_j is thus

$$S_j = 1 + \sum_{k=j+1}^N I_k \quad (39)$$

Taking the expectation yields

$$\mathbb{E}[S_j] = 1 + \sum_{k=j+1}^N \frac{1}{k} = 1 + H_N - H_j \quad (40)$$

where, $H_n = \sum_{r=1}^n 1/r$ is the n th harmonic number. Since $H_n = \theta(\log n)$, the expected class-related storage overhead is

$$\mathbb{E}[\|\mathcal{K}_{C_j}\|] = \mathcal{O}\left(\log\left(\frac{N}{j}\right) + 1\right) \quad (41)$$

Thus, the inter-class storage contribution grows logarithmically with the hierarchy size. In the practical IoT regime considered in this work, the number of security classes is substantially smaller than the maximum number of members in a class, i.e., $N \ll M$. Therefore, this inter-class component does not change the dominant storage order when combined with the local LKH component.

To evaluate the local storage component $|\mathcal{K}_{m_i^j}|$, we solve the stationary class-population process for each maximum class size M . The resulting distribution characterizes the long-term probability of each population state. For every population state, the number of locally stored keys is obtained from the corresponding balanced binary LKH-tree depth.

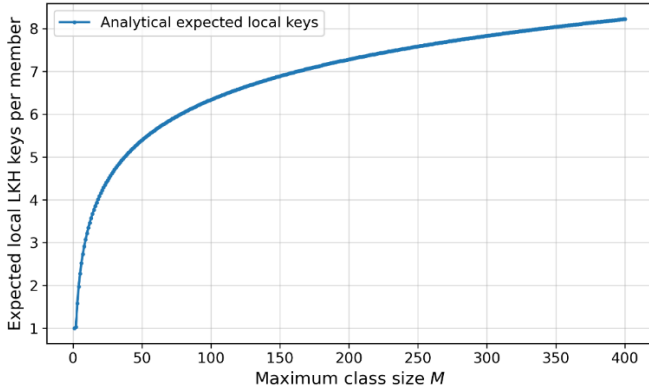


Figure 7. Expected number of local Logical Key Hierarchy (LKH) keys per member versus the maximum class size M for a balanced binary LKH tree

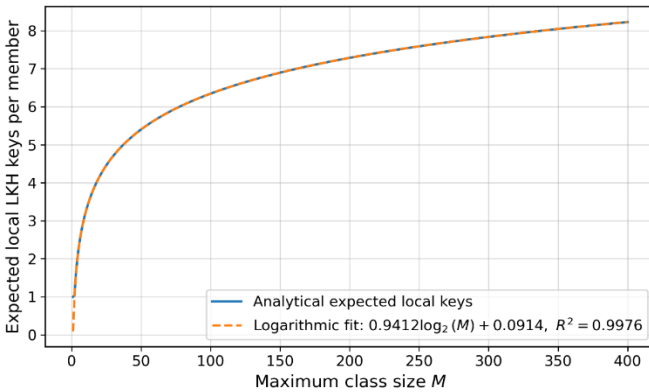


Figure 8. Analytical local-key storage values and their logarithmic fit as a function of the maximum class size M

Figure 7 shows that the expected number of local LKH keys increases slowly with the maximum class size. Figure 8

compares the analytical values with a least-squares logarithmic fit. The resulting approximation is

$$\mathbb{E}[K^{(j)}] = 0.9412 \log_2(M) + 0.0914, R^2 = 0.9976 \quad (42)$$

The high goodness-of-fit value confirms that the local key-storage component grows logarithmically with the class size. Therefore,

$$|\mathcal{K}_{m_i^j}| \sim \mathcal{O}(\log M) \quad (43)$$

Finally, by combining the inter-class bound in Eq. (41) with the local LKH bound in Eq. (43), the general per-member storage overhead is

$$\text{stor}_i^j = \mathcal{O}\left(\log\left(\frac{N}{j}\right) + 1 + \log M\right) \quad (44)$$

Since $j \geq 1$ and $N \leq M$ in the practical regime considered in this work, we have

$$\log\left(\frac{N}{j}\right) + 1 \leq \log N + 1 = \mathcal{O}(\log M) \quad (45)$$

Consequently, the overall per-member storage overhead simplifies to

$$\text{stor}_i^j = \mathcal{O}(\log M) \quad (46)$$

This result shows that the storage overhead remains logarithmic with respect to the maximum class size in the considered regime.

5.4 Scaling evaluation

To assess scalability beyond the moderate ranges illustrated in the previous figures, we extended the stationary evaluation up to $N = 10\,000$ security classes and $M = 10\,000$ members per class. Table 4 shows that the class-related anchor key requirement at the highest class grows slowly, from 6.5699 keys for $N = 400$ to 9.7876 keys for $N = 10\,000$. Similarly, the expected local LKH storage increases from 8.2753 keys for $M = 400$ to 12.9000 keys for $M = 10\,000$. These results provide a numerical illustration of the logarithmic storage growth at larger scales.

Table 4. Extended-scale stationary results under the baseline configuration. The class-anchor value corresponds to the upper-level class C_1

N	$\bar{C}(N)$	$\mathbb{E}[\ \mathcal{K}_{C_1}\]$	M	$\mathbb{E}[K^{(j)}]$
400	100.7500	6.5699	400	8.2753
1000	250.7500	7.4855	1000	9.5869
5000	1250.7500	9.0945	5000	11.9007
10000	2500.7500	9.7876	10000	12.9000

Note: Logical Key Hierarchy (LKH).

5.5 Sensitivity to membership dynamics

To assess the effect of membership dynamics, we consider five rate configurations: low balanced churn, baseline balanced churn, high balanced churn, join-dominated dynamics, and leave-dominated dynamics. The evaluated rate pairs are summarized in Table 5.

Table 5. Sensitivity of the stationary results to join and leave rate combinations. The reported local-key values correspond to $M = 100$, while the Key Continuity Process (KCP) continuity value corresponds to $N = 100$

Scenario	$\lambda_J(\text{s}^{-1})$	$\lambda_L(\text{s}^{-1})$	$\rho = \lambda_J/\lambda_L$	$\bar{C}(100)$	$\mathbb{E}[K^{(J)}]$
Low balanced	0.01	0.01	1.0	25.75	6.3066
Baseline balanced	0.05	0.05	1.0	25.75	6.3066
High balanced	0.10	0.10	1.0	25.75	6.3066
Join-dominated	0.10	0.01	10.0	25.75	7.7186
Leave-dominated	0.01	0.10	0.1	25.75	1.1070

For the KCP, both join and leave events trigger a class-key update and therefore have the same structural effect on the continuity relation. Under homogeneous rates across classes, the stationary KCP depends on the relative update rates between classes rather than on the common update-rate scale. Consequently, the stationary system-wide mean continuity remains unchanged across the considered scenarios. For the reference hierarchy size $N = 100$, we obtain for all five rate configurations. The absolute update rates affect the transient speed at which the stationary regime is reached, but not the stationary continuity value under the homogeneous assumptions adopted here.

$$\bar{C}(100) = 25.75 \quad (47)$$

In contrast, the CKDP is directly affected by the join-to-leave rate ratio

$$\rho = \frac{\lambda_J}{\lambda_L} \quad (48)$$

When $\rho = 1$, the stationary population distribution is uniform over the admissible population states, and changing the common scale of λ_J and λ_L does not modify the steady-state local storage. When $\rho > 1$, the population distribution shifts toward the maximum class size, which increases the expected number of local LKH keys. Conversely, when $\rho < 1$, the stationary distribution concentrates near the minimum class size and reduces the local storage requirement.

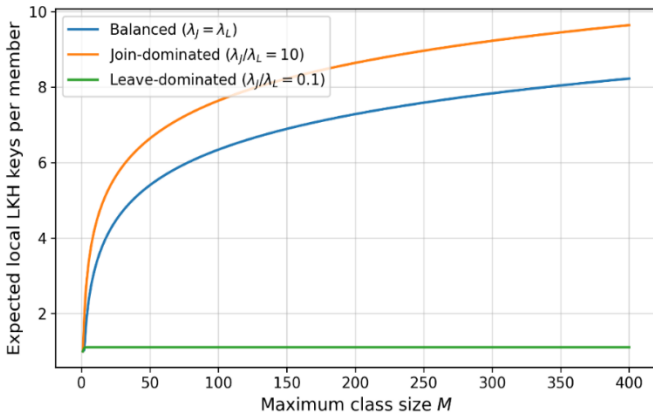


Figure 9. Sensitivity of the expected local Logical Key Hierarchy (LKH) keys to membership-rate combinations. The balanced curve represents all configurations with $\lambda_J = \lambda_L$, which have identical stationary local-storage values

Figure 9 illustrates the effect of these dynamics on the expected local LKH storage over a range of maximum class sizes. The balanced curve represents all balanced configurations, since they lead to identical stationary values.

The join-dominated configuration produces higher local storage because the stationary population is concentrated near the upper boundary, whereas the leave-dominated configuration remains close to one local key per member because the population is concentrated near the lower boundary.

5.6 Computation overhead

To evaluate the computation overhead, we rely on the results of both the KCP and the CKDP. Let us consider a hierarchy of N classes, where each class C_j contains at most M members.

5.6.1 Server side

When a member joins or leaves a class C_j , the KS must update the affected local keys and distribute the corresponding rekeying messages. In the LKH tree, only the keys located on the path from the affected leaf to the root are updated. Therefore, if the tree has depth h , the number of updated nodes is proportional to h . Since each updated key must be securely distributed, the server-side computation overhead is proportional to the tree depth.

From Eq. (43), the depth of the LKH tree grows logarithmically with the class size. Hence,

$$h \sim \mathcal{O}(\log M)$$

Therefore, the server-side computation overhead can be expressed as

$$\text{comp}_i^{j,\text{server}} \sim \mathcal{O}(\log M) \quad (49)$$

5.6.2 User side

At the user side, the computation overhead includes two components: the overhead induced by the inter-class continuity and the overhead induced by the local LKH rekeying process.

For the inter-class part, the worst-case number of hash derivations or decryptions associated with the continuity chain is proportional to the length of the continuity segment. From Eq. (36), this contribution is

$$\mathcal{O}(N) \quad (50)$$

For the intra-class part, the number of decryptions performed by a member depends on its position in the LKH tree.

Joining Event. When a new member joins a class C_j , the affected path is updated. In the worst case, the existing member that is closest to the inserted member in the tree may need to process one updated key per level. Therefore, the worst-case number of decryptions is proportional to the tree

depth:

$$\text{Dec}_{\text{join}} \sim \mathcal{O}(h) \quad (51)$$

Since $h \sim \mathcal{O}(\log M)$, we obtain

$$\text{Dec}_{\text{join}} \sim \mathcal{O}(\log M) \quad (52)$$

Leaving Event. When a member leaves the class, the keys along the affected path are refreshed. In the worst case, the member most closely related to the leaving member in the tree performs a number of decryptions proportional to the tree depth. Hence,

$$\text{Dec}_{\text{leave}} \sim \mathcal{O}(h) \sim \mathcal{O}(\log M) \quad (53)$$

By combining the inter-class and intra-class contributions, the user-side computation overhead is

$$\text{comp}_i^{j,\text{user}} \sim \mathcal{O}(N + \log M) \quad (54)$$

If the number of classes is small compared with the class size, that is, if $N \ll M$, then the logarithmic term dominates in practical large-scale classes. In that case, the complexity can be approximated by

$$\text{comp}_i^{j,\text{user}} \sim \mathcal{O}(\log M) \quad (55)$$

5.7 Communication overhead

The communication overhead represents the number of rekeying messages sent by the KS after a join or leave event. As in the previous analysis, this overhead is evaluated by combining the inter-class behavior captured by the KCP and the intra-class behavior captured by the CKDP.

For the inter-class part, when a membership change affects a class C_j , the continuity structure may be split or updated, and the affected upper classes must receive the corresponding update information. In the considered KMS, this inter-class overhead remains bounded, and therefore it is constant asymptotically:

$$\mathcal{O}(1) \quad (56)$$

For the intra-class part, let us consider the two events separately.

Joining Event. When a new member joins class C_j , the KS

updates the keys located on the affected path of the LKH tree. Since the number of updated levels is proportional to the tree depth h , the number of rekeying messages generated inside the class is also proportional to h . Therefore,

$$\text{comm}_{\text{join}} \sim \mathcal{O}(h) \quad (57)$$

Since $h \sim \mathcal{O}(\log M)$, it follows that

$$\text{comm}_{\text{join}} \sim \mathcal{O}(\log M) \quad (58)$$

Leaving Event. Similarly, when a member leaves class C_j , the KS refreshes the keys on the affected path and distributes the corresponding rekeying messages. The resulting number of messages is again proportional to the depth of the tree. Hence,

$$\text{comm}_{\text{leave}} \sim \mathcal{O}(h) \sim \mathcal{O}(\log M) \quad (59)$$

By combining the inter-class and intra-class contributions, the overall communication overhead becomes

$$\text{comm}_i^j \sim \mathcal{O}(1 + \log M) \quad (60)$$

Therefore,

$$\text{comm}_i^j \sim \mathcal{O}(\log M) \quad (61)$$

This result shows that the communication overhead of the considered KMS increases logarithmically with the maximum number of members in a class.

6. COMPARISON WITH PREVIOUS WORK

Table 6 provides an asymptotic comparison with representative dynamic linear-hierarchy KMSs, namely KTLH [25], LHSC [26], and DKM [27]. The comparison is based on the complexity expressions reported in the original studies and on the analytical results derived for SEKM-IoT in this work. For consistency, all metrics are interpreted at the per-rekeying-event level, using M to denote the number of members in the affected class. This comparison is intended to highlight asymptotic scalability trade-offs; it is not a head-to-head implementation benchmark, since the schemes were originally evaluated under different assumptions and environments.

Table 6. Comparison of storage, computation, and communication overheads under the practical regime $N \ll M$

KMS	Storage Overhead	Computation Overhead (User)	Computation Overhead (Server)	Communication Overhead
SEKM-IoT [9]	$\mathcal{O}(\log M)$	$\mathcal{O}(\log M)$	$\mathcal{O}(\log M)$	$\mathcal{O}(\log M)$
KTLH [25]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(M)$	$\mathcal{O}(M)$
LHSC [26]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(M)$	$\mathcal{O}(M)$
DKM [27]	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(M)$	$\mathcal{O}(M)$

Note: Internet of Things (IoT), Key Management Schemes (KMSs), Secure and Efficient Key Management Scheme for Internet of Things (SEKM-IoT).

6.1 Storage overhead

As shown in Table 6, SEKM-IoT requires a storage overhead of order $\mathcal{O}(\log M)$ per member. This logarithmic growth remains moderate even when the class size becomes

large, and therefore preserves the practicality of the KMS in dynamic IoT environments. In contrast, KTLH, LHSC, and DKM require only $\mathcal{O}(1)$ storage. Although this constant storage overhead appears attractive, it should not be evaluated in isolation. Indeed, as indicated by the table, these schemes

incur significantly higher overheads at the server and communication levels, both of which grow linearly with the class size. Therefore, the apparent storage advantage of these KMSs is achieved at the expense of scalability in other performance dimensions.

6.2 Computation overhead

At the user side, SEKM-IoT incurs a computation overhead of order $\mathcal{O}(\log M)$, whereas KTLH, LHSC, and DKM achieve $\mathcal{O}(1)$. This means that a member in SEKM-IoT may perform slightly more computations during rekeying operations. However, from a scalability point of view, the most critical component is generally the server-side computation overhead, since the KS is responsible for updating and distributing keys to all affected members.

In KTLH, LHSC, and DKM, the rekeying process relies essentially on individual transmissions, which causes the server-side computation overhead to grow linearly with the class size, namely $\mathcal{O}(M)$. By contrast, SEKM-IoT exploits the LKH structure in order to update keys through a multicast-oriented mechanism. As a result, only the keys located on the affected path of the local tree need to be processed and distributed, which reduces the server-side computation overhead to $\mathcal{O}(\log M)$. This represents a significant improvement in terms of scalability, especially when the number of members becomes large.

6.3 Communication overhead

The same observation applies to the communication overhead. Table 7 shows that KTLH, LHSC, and DKM require $\mathcal{O}(M)$ communication overhead, because each rekeying operation involves a number of update messages proportional to the class size. Such an overhead may become prohibitive in large and dynamic IoT environments, where bandwidth and energy are often limited.

In contrast, SEKM-IoT reduces the communication overhead to $\mathcal{O}(\log M)$ by relying on the LKH tree to distribute updated keys efficiently to groups of members. Instead of sending separate rekeying messages to all members, the KS exploits shared intermediate keys to cover multiple users simultaneously. Consequently, the communication overhead remains low even when the class size increases significantly.

6.4 Summary

The above comparison shows that evaluating a KMS according to a single metric may lead to misleading conclusions. Although KTLH, LHSC, and DKM achieve constant storage overhead and constant user-side computation overhead, they scale poorly at the server and communication levels. SEKM-IoT, on the other hand, provides a more balanced behavior by achieving logarithmic overhead for storage, computation, and communication. This improvement is mainly due to the integration of the LKH structure for intra-class rekeying and the efficient management of hierarchical accessibility. Therefore, SEKM-IoT appears to be better suited for large-scale and dynamic linear-hierarchy IoT environments.

7. CONCLUSION

In this paper, we proposed a stochastic framework for

evaluating the performance of linear-hierarchy KMSs in dynamic IoT environments. More precisely, the proposed framework relies on discrete-time Markov chains and Poisson-driven membership events in order to model the stochastic behavior of both the inter-class KCP and the intra-class CKDP. Based on these probabilistic models, steady-state analytical expressions were derived for the main performance metrics, namely storage, computation, and communication overheads. In this way, the proposed framework makes it possible to move beyond static or short-term evaluations and to characterize the long-term behavior of KMSs under dynamic operating conditions.

To illustrate the applicability of the framework, we instantiated it on SEKM-IoT as a representative case study. The obtained results show that this KMS achieves logarithmic scalability for the main overhead metrics. More precisely, the analysis indicates that the inter-class component induces a bounded overhead, while the intra-class component governed by the LKH structure leads to logarithmic growth with respect to the maximum class size. These results confirm that SEKM-IoT provides an interesting trade-off between security support and scalability in dynamic linear-hierarchy IoT systems.

The proposed framework provides a rigorous basis for evaluating the long-term behavior of linear-hierarchy KMSs under dynamic membership conditions. At the same time, some extensions can further broaden its applicability. In the present work, membership changes are modeled through independent Poisson processes, which provide a tractable and suitable representation of random join and leave events. However, future work may consider more general stochastic models in order to capture bursty, correlated, or time-varying membership dynamics that may appear in some IoT deployments.

In addition, the current framework focuses on the performance evaluation of storage, computation, and communication overheads. This choice is consistent with the main objective of the paper, which is to quantify the scalability of hierarchical KMSs under dynamic conditions. As a future direction, the proposed stochastic analysis could be combined with formal verification techniques in order to jointly assess performance behavior and security properties under more advanced adversarial models. Such extensions would provide a more comprehensive methodology for the analysis and design of secure and scalable KMSs for dynamic IoT environments.

REFERENCES

- [1] Alioanei, C., Popescu, N. (2025). AI-based solutions for security and resource optimization in IoT environments: A systematic review. *Information*, 16(10): 841. <https://doi.org/10.3390/info16100841>
- [2] Zeng, F., Pang, C., Tang, H.J. (2024). Sensors on Internet of Things systems for the sustainable development of smart cities: A systematic literature review. *Sensors*, 24(7): 2074. <https://doi.org/10.3390/s24072074>
- [3] Oladimeji, D., Gupta, K., Kose, N.A., Gundogan, K., Ge, L.Q., Liang, F. (2023). Smart transportation: An overview of technologies and applications. *Sensors*, 23(8): 3880. <https://doi.org/10.3390/s23083880>
- [4] Quy, V.K., Hau, N.V., Anh, D.V., et al. (2022). IoT-enabled smart agriculture: Architecture, applications, and challenges. *Applied Sciences*, 12(7): 3396.

- <https://doi.org/10.3390/app12073396>
- [5] Benmalek, M., Harkat, K., Haouam, K.D., Gheid, Z. (2023). SE-CDR: Enhancing security and efficiency of key management in internet of energy consumer demand-response communications. *International Journal of Safety and Security Engineering*, 13(4): 611-623. <https://doi.org/10.18280/ijssse.130403>
 - [6] Sun, P.J., Wan, Y., Wu, Z.D., Fang, Z.X., Li, Q. (2025). A survey on privacy and security issues in IoT-based environments: Technologies, protection measures and future directions. *Computers & Security*, 148: 104097. <https://doi.org/10.1016/j.cose.2024.104097>
 - [7] Sebestyen, H., Popescu, D.E., Zmaranda, R.D. (2025). A literature review on security in the Internet of Things: Identifying and analysing critical categories. *Computers*, 14(2): 61. <https://doi.org/10.3390/computers14020061>
 - [8] Samiullah, F., Gan, M.L., Akleylek, S., Aun, Y. (2023). Group key management in Internet of Things: A systematic literature review. *IEEE Access*, 11: 77464-77491. <https://doi.org/10.1109/ACCESS.2023.3298024>
 - [9] Benmalek, M. (2024). SEKM-IoT: Secure and efficient key management for dynamic access control in smart IoT systems. In 2024 21st International Conference on High Capacity Optical Networks and Enabling Technologies (HONET), Doha, Qatar, pp. 97-102. <https://doi.org/10.1109/HONET63146.2024.10822894>
 - [10] Kokila, M., Reddy, K.S. (2025). Authentication, access control and scalability models in Internet of Things security: A review. *Cyber Security and Applications*, 3: 100057. <https://doi.org/10.1016/j.csa.2024.100057>
 - [11] Song, W.J., Liu, M.Q., Baker, T., Zhang, Q.K., Tan, Y.A. (2023). A group key exchange and secure data sharing based on privacy protection for federated learning in edge-cloud collaborative computing environment. *International Journal of Network Management*, 33(5): e2225. <https://doi.org/10.1002/nem.2225>
 - [12] Dammak, M., Senouci, S.M., Messous, M.A., Elhdhili, M.H., Gransart, C. (2020). Decentralized lightweight group key management for dynamic access control in IoT environments. *IEEE Transactions on Network and Service Management*, 17(3): 1742-1757. <https://doi.org/10.1109/TNSM.2020.3002957>
 - [13] Abdmeziem, M.R., Nacer, A.A., Deroues, N.M. (2024). Group key management in the Internet of Things: Handling asynchronicity. *Future Generation Computer Systems*, 152: 273-287. <https://doi.org/10.1016/j.future.2023.10.023>
 - [14] Tan, H.W., Chung, I.Y. (2019). Secure authentication and group key distribution scheme for WBANs based on smartphone ECG sensor. *IEEE Access*, 7: 151459-151474. <https://doi.org/10.1109/ACCESS.2019.2948207>
 - [15] Barbareschi, M., Casola, V., Emmanuele, A., Lombardi, D. (2024). A lightweight PUF-based protocol for dynamic and secure group key management in IoT. *IEEE Internet of Things Journal*, 11(20): 32969-32984. <https://doi.org/10.1109/JIOT.2024.3418207>
 - [16] Najafi, Z., Babaie, S. (2023). A lightweight hierarchical key management approach for Internet of Things. *Journal of Information Security and Applications*, 75: 103485. <https://doi.org/10.1016/j.jisa.2023.103485>
 - [17] Kuang, Y.P., Wu, Q.W., Chen, R.Q., Liu, X.L. (2025). Blockchain based lightweight authentication scheme for Internet of Things using lattice encryption algorithm. *Computer Standards & Interfaces*, 93: 103981. <https://doi.org/10.1016/j.csi.2025.103981>
 - [18] Karankar, N., Seth, A. (2025). An IoT system for access control using blockchain and message queuing system. *EURASIP Journal on Information Security*, 2025(1): 31. <https://doi.org/10.1186/s13635-025-00208-4>
 - [19] Park, K. (2025). Decentralized authentication and data access control scheme using DID for fog-enabled industrial Internet of Things. *Mathematics*, 13(22): 3686. <https://doi.org/10.3390/math13223686>
 - [20] Radhakrishnan, I., Jadon, S., Honnavalli, P.B. (2024). Efficiency and security evaluation of lightweight cryptographic algorithms for resource-constrained IoT devices. *Sensors*, 24(12): 4008. <https://doi.org/10.3390/s24124008>
 - [21] Ullah, S., Nasir, H.M., Kadir, K., et al. (2025). End-to-end encryption enabled lightweight mutual authentication scheme for resource constrained IoT network. *Computers, Materials & Continua*, 82(2): 3223-3249. <https://doi.org/10.32604/cmc.2024.054676>
 - [22] Benmalek, M., Challal, Y. (2015). eSKAMI: Efficient and scalable multi-group key management for advanced metering infrastructure in smart grid. In 2015 IEEE Trustcom/BigDataSE/ISPA, Helsinki, Finland, pp. 782-789. <https://doi.org/10.1109/Trustcom.2015.447>
 - [23] Benmalek, M., Challal, Y., Bouabdallah, A. (2015). Scalable multi-group key management for advanced metering infrastructure. In 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM), Liverpool, UK, pp. 183-190. <https://doi.org/10.1109/CIT/IUCC/DASC/PICOM.2015.27>
 - [24] Benmalek, M., Challal, Y., Derhab, A. (2019). An improved key graph-based key management scheme for smart grid AMI systems. In 2019 IEEE Wireless Communications and Networking Conference (WCNC), Marrakesh, Morocco, pp. 1-6. <https://doi.org/10.1109/WCNC.2019.8885646>
 - [25] Hassen, H.R., Bettahar, H., Bouabdallah, A., Challal, Y. (2012). An efficient key management scheme for content access control for linear hierarchies. *Computer Networks*, 56(8): 2107-2118. <https://doi.org/10.1016/j.comnet.2012.02.006>
 - [26] Odelu, V., Das, A.K., Goswami, A. (2013). LHSC: An effective dynamic key management scheme for linear hierarchical access control. In 2013 Fifth International Conference on Communication Systems and Networks (COMSNETS), Bangalore, India, pp. 1-9. <https://doi.org/10.1109/COMSNETS.2013.6465571>
 - [27] Lopriore, L. (2018). Key management in tree-shaped hierarchies. *Information Security Journal: A Global Perspective*, 27(4): 205-213. <https://doi.org/10.1080/19393555.2018.1516835>
 - [28] More, S.S., More, P.S. (2025). Lightweight key management mechanism using lattice-based encryption for IoT data management systems. *Transactions on Emerging Telecommunications Technologies*, 36(12): e70317. <https://doi.org/10.1002/ett.70317>
 - [29] Wong, C.K., Gouda, M., Lam, S.S. (2000). Secure group communications using key graphs. *IEEE/ACM*

Transactions on Networking, 8(1): 16-30.
<https://doi.org/10.1109/90.836475>

- [30] Wang, Y.T., Wu, G.L. (2024). Intrusion detection for Internet of Things security: A hidden Markov model based on fuzzy rough set. Proceedings of SPIE, 13397: 133970V. <https://doi.org/10.1117/12.3052573>
- [31] Nicol, D.M., Sanders, W.H., Trivedi, K.S. (2004). Model-based evaluation: From dependability to security. IEEE Transactions on Dependable and Secure Computing, 1(1): 48-65. <https://doi.org/10.1109/TDSC.2004.11>
- [32] Kwiatkowska, M., Norman, G., Parker, D. (2011). PRISM 4.0: Verification of probabilistic real-time systems. In Computer Aided Verification, Springer, Berlin, Heidelberg, pp. 585-591. https://doi.org/10.1007/978-3-642-22110-1_47
- [33] Ragab-Hassen, H., Lounes, E. (2017). A key management scheme evaluation using Markov processes. International Journal of Information Security, 16(3): 271-280. <https://doi.org/10.1007/s10207-016-0323-3>
- [34] Norris, J.R. (1997). Markov Chains. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge, UK. <https://doi.org/10.1017/CBO9780511810633>
- [35] Kingman, J.F.C. (1993). Poisson Processes. Oxford Studies in Probability. Clarendon Press, Oxford, UK.

NOMENCLATURE

C_i	The i^{th} class
CK_i^p	The i^{th} class key after $(p - 1)$ updates
KS	The Key Server
M	Maximum number of members in a class
d	Degree of the LKH tree
$C_i \leq C_j$	C_i is a lower class than or equal to C_j
$\mathcal{H}$	Hash function
$\text{Ct}(C_j)$	Continuity of class C_j
Q	Infinitesimal generator matrix
stor_i^j	Storage overhead for member m_i^j
$\text{comp}_i^{j,\text{server}}$	Computation overhead at the server side
CK_i	The i^{th} class key
m_i^j	The i^{th} member of class C_j
N	Total number of security classes
h	Depth of the LKH tree
$C_i < C_j$	C_i is a lower class than C_j
$\text{Enc}(\text{Msg}, k)$	Message Msg encrypted with key k
$\mathcal{H}^k$	k -fold application of the hash function
λ	Rate of the Poisson process
π	Stationary distribution vector
$\text{comp}_i^{j,\text{usr}}$	Computation overhead at the user side