Group Key Management in constrained IoT Settings

Alessandro Piccoli*, Marc-Oliver Pahl‡*, Lars Wüstrich*
*Technical University of Munich, ‡IMT Atlantique
*{alessandro.piccoli}@tum.de, {pahl,wuestrich}@s2o.net.in.tum.de; ‡ marc-oliver.pahl@imt-atlantique.fr

***Abstract*—The Internet of Things (IoT) enables software to orchestrate physical spaces. Due to the increased impact, IoT communication in factories, households, or critical infrastructures has to be highly secured. Besides point-to-point communication, group communication is frequently used in the IoT. Securing it typically requires the exchange of cryptographic keys. Several protocols have been proposed for such Group Key Management (GKM). They vary in their targeted settings, in their Key Distribution Model, Architecture Model, Reliability Properties, and Protocol Overhead. This paper surveys existing GKM mechanisms, analyzes their suitability for constrained IoT settings, and identifies open issues that require further research.**

***Index Terms*—IoT, security, group key management, autonomous management, constrained nodes, reliability**

I. Introduction

The Internet of Things (IoT) spreads into our lives. IoT systems can be characterized as distributed software processes communicating over a network [1], [2]. Critical applications such as Industry 4.0 increase the need for high resilience [3]. Many IoT devices are Cyber-Physical, interacting with their physical surrounding. Consequently security, safety, and privacy risks occur. A central requirement for IoT systems is therefore security [4].

A representative scenario for a distributed critical IoT-based infrastructure is the Smart Grid. It consists of connected substations. Orchestrating them is safety-critical. This paper uses a substation grid as guiding scenario.

IoT scenarios can have different topologies. Often they imply *group communication*. It is essential for different operations such as synchronizing information between distributed system parts, or for reducing latency, and using network bandwidth more efficiently. Securing the IoT requires *secure group communication*.

Secure group key communication and many other security mechanisms rely on *shared secret keys*. The management of secret keys on distributed entities is called *Group Key Management (GKM). This paper focuses on GKM.* Since IoT systems can be highly dynamic [2], with entities continuously joining and leaving, central operations of GKM are member join, leave, and re-key. *The central challenge of GKM is ensuring that all authorized group members always have an up-to-date key.*

Funded by the German Federal Ministry of Economic Affairs and Energy (BMWI) in DECENT (0350024A) and the GFA in SCHEIF.

Sensor networks, industrial automation networks, or even power distribution networks are usually managed by embedded systems with limited computational resources [5]. Typical limitations are CPU, power, and memory. Resource constraints result in unreliable and lossy communication channels, limited bandwidth, and dynamic network topology [6].

Resource-constraints make GKM in the IoT challenging. In addition, IoT field deployments vary in their architecture. Every installation is different, tailored to its application scenario and local setting. Consequently both, centralized and distributed architectures as well as synchronous or asynchronous key distribution modes are relevant for IoT GKM [7].

In centralized GKM schemes a group controller interacts with the group. Distributed architectures rely on an elected group member to manage specific sessions or require the group members to interact and agree on a group key. In both cases, the communication modes can be synchronous push or asynchronous pull.

This research

- surveys existing GKM mechanisms
- analyzes their suitability for constrained IoT settings
- identifies open issues that require further research

Section II presents related work and highlights common GKM protocol classifications. Section III introduces design fundamentals of GKM protocols and requirements of constrained IoT settings. Section IV assesses relevant GKM state of the art. Section V summarizes the identified shortcomings in existing group key management algorithms in constrained environments that require further research.

II. Related Work

Group Key Management algorithms have been studied for a long time. This paper follows the assessment criteria of existing surveys, extends them, and assesses more algorithms than before.

Though several surveys exist [7]–[13], compared to this work they only provide a partial view regarding both, the assessment criteria and the selection of GKM algorithms. In particular, the existing surveys do not highlight the distinction between synchronous and asynchronous GKM models. Moreover, the existing related work does not focus on constrained networks where providing reliability is a central challenge.

An assessment criterion common to [10]–[12] is the distinction between network-dependent and network-independent protocols. These studies investigate the context of wireless and sensor networks. Network-dependent protocols are further divided into cluster-based and tree-based designs [10]. Network-

independent protocols are classified in terms of the involvement of a Key Distribution Center (KDC) [10]. Compared to existing works, we further investigate and extend the context of network-independent protocols.

The survey [8] assesses GKM protocols according to this distinction and evaluates their performance. The symmetric key used by a group might be generated by a KDC in a centralized architecture, or collaboratively generated by the group members. According to this criterion, [8] distinguishes protocols between *Key Distribution* and *Key Agreement* schemes.

The authors of [9] further investigate Key Agreement Schemes. After an introduction to centralized schemes and their drawbacks, they provide an overview of existing distributed Group Key Management schemes. The mechanisms proposed in [9] are all variations of the Diffie-Hellman (DH) key agreement protocol [14]. Their scheme is extended from two to *n* parties. The Group Diffie-Hellman (GDH) key agreement enables multiple peers to agree on a shared secret as a function of the contributions of all the group members. The study in [9] also investigates the possibility of guaranteeing backward- and forward-secrecy in distributed environments.

A survey of Key Management Schemes for group communication can be found in [7]. The authors identify three categories. Each category presents different requirements. The paper proposes an additional class of GKM protocols, using the following classification:

1) Centralized group key management protocols that rely on a KDC,
2) Decentralized architectures, where group members are divided into subgroups with different group managers,
3) Distributed key-management protocols that do not require a KDC. The key generation is either performed by all the members collaboratively, or by an elected member and then distributed within the group.

The main contribution of [7] is a thorough protocol overview according to the three listed categories.

III. Foundations and Assessment Criteria

Managing security credentials in a group is the basis for secure communication between its members. A shared cryptographic secret can add confidentiality, integrity, and group-based authenticity to a communication channel [15]. Central operations of Group Key Management (GKM) are *member registration* and *re-keying*.

The member registration process enables joining a secured group. In this process a new member authenticates and retrieves the group-key either from a Key Distribution Center (KDC) or from a peer.

Group-keys typically have a lifetime for security reasons, such as replay protection. Thus, they need regular updating. Since members might be part of a group only for a limited duration, excluding them also requires a key change with the remaining parties. When a new member joins a group, the shared key is typically also updated to prevent the decryption of previously exchanged messages. Hence, group re-keying happens in regular intervals, or in the event of a membership change [15].

According to a group policy, a group key might be updated after every membership change. Typical implementations are individual re-keying or re-keying after a specific number of changes, i.e. batch re-keying [15], [16].

To illustrate commonalities and differences, the remainder of this section introduces the foundations of the different existing GKM models. The following classification serves as reference for the detailed protocol analysis in section IV.

This paper identifies two main criteria to classify GKM protocols: the *topological structure*, centralized and distributed, and the *key distribution mechanism* that can be asynchronous, i.e. pull-based, or synchronous, i.e. push-based.

Centralized and distributed models are two relevant and widely deployed GKM architectures. A centralized architecture relies on a central KDC to manage and coordinate the GKM operations. In a distributed architecture, no KDC is present. The group members self-organize to manage the security credentials for the group [7].

Low-latency and constrained networks often employ one or more central entities for ensuring flexibility and fault-tolerance [17]. However, they rely on a single entity, the KDC. This means that a failure of the KDC could affect the whole group.

In a centralized model keys are distributed and updated synchronously or asynchronously.

In the *asynchronous* model, after an initial registration, the group members are responsible for retrieving the group key from the KDC. Unreliable behavior of a member results in missing key updates. This leads to group inconsistency and results in an exclusion of these members from the communication [18].

In the synchronous model, after the initial registration, group members start listening for incoming push messages for updating their local group key. If a key update is missed, a member has no mechanism to recover from the failure. In [7], some specific mechanisms are presented to improve the efficiency of the push-re-key procedure in large groups.

In distributed environments, group members solve the GKM challenge in two possible ways [7], [9]. (1) A group member is elected to assume a controlling role within the group, or (2) group members collaboratively create security credentials. While centralized protocols are more resource-efficient, they rely on a single entity which should always be available and might become a single point of failure. In distributed architectures, the workload is more uniform and the system is efficient [17].

A reliable protocol execution is a priority for safety-critical applications. Reliability mechanisms include enabling a legitimate group member to receive, verify, and read messages from the KDC despite potential transmission errors. In addition, the key distribution and update procedures have to guarantee group-consistency, so that authorized group members always share the same secret key.

The described GKM models also need to satisfy the resource-constraints of energy automation networks. In par-

ticular, these limitations require the utilization of minimal resources in terms of bandwidth usage, and storage. Regarding the specific requirements in resource-constrained environments, this paper therefore assesses the existing solutions regarding:

1) *Architecture Model*: centralized or distributed
2) *Key Distribution Model*: synchronous or asynchronous
3) *Reliability*: protocol reliability including group consistency ensurance
4) *Protocol Overhead*: computational and message overhead

IV. Survey of GKM Algorithms

Section III introduced a classification for Group Key Management (GKM) protocols. This section assesses at least one protocol of each family. The assessment surveys each component separately to highlight differences between the protocols. Table I gives an overview on the assessment results.

Synchronous mechanisms make the start. The studies listed in this section represent the efforts of the Internet Engineering Task Force (IETF) to standardize Group Key Management protocols with RFCs.

Then, asynchronous mechanisms are presented. These protocols have not been widely studied and collected in the other referenced surveys. However, they promise higher reliability in the target constrained IoT setting.

The last part assesses the reliability of the chosen protocols, looking at packet loss-recovery and communication reliability. Another focus lays on group-consistency in asynchronous environments.

A. Synchronous Key Distribution Mechanisms

The centralized protocols presented in this section rely on a synchronous key distribution mechanism. This means that the group controller is responsible for initiating a push-based communication for distributing and updating the group key of the subscribed members.

1) Multimedia Internet Keying: Multimedia Internet KEYing (MIKEY) is specified in RFC3830 [19]. It is a lightweight authentication and key-management protocol for real-time applications [20]. The common setting for this protocol is multimedia applications in point-to-point or group-based communication.

MIKEY's core feature is a key exchange within only one round trip. MIKEY does not use a Key distribution Center (KDC). The protocol-initiating member becomes the group controller for the session. Group controller and members authenticate with pre-shared keys. Group-membership management is not supported.

MIKEY has a low message-overhead, making it well-suitable for constrained environments. However, MIKEY misses a key-update mechanism. Re-keying is performed by re-initiating the whole protocol. In addition, MIKEY addresses small and interactive group [19]. It has no packet loss recovery mechanism.

2) Group Domain of Interpretation: The Group Domain of Interpretation (GDOI) is specified in RFC 6407. It is a group security association and key management protocol (ISAKMP) [21]. GDOI distributes security associations, group-keys and policies. It ensures message authenticity, secrecy and freshness [21]. Like MIKEY, GDOI provides a low message-overhead. Inn addition, it provides higher security and flexibility.

GDOI uses a KDC. The KDC manages the group membership and generates the keys. Group members are authenticated to the KDC. The authorization mechanism is not standardized.

The initial authentication is performed with the Internet Key Exchange version 1 [22]. The group participants and the KDC can authenticate with a pre-shared key, or asymmetric cryptography via a Public Key Infrastructure (PKI) [22]. The initial key material secures the next messages.

After initial authentication, members request the current group key with a pull operation. The KDC can push re-key messages to the members in case of a key expiration, or membership changes. Once all members share a common key, they can securely communicate via multicast or broadcast.

With the KDC, GDOI is a centralized protocol. It specifies both, synchronous and asynchronous key-distribution mechanisms. The initial registration adds a significant message overhead: 6 messages for IKE and 4 messages for the pull operation. However, the push-re-key operation consists in just one message. Hence, after the initial expensive registration the system is very lightweight. No reliability measure is explicitly included in the protocol specification. The next paragraph covers a proposed protocol that is supposed to obsolete the standard GDOI specification.

3) GSAKMP: Unless the synchronous approaches presented so far, the following Group Secure Association Key Management Protocol (GSAKMP) can provide GKM for large and dynamic groups

GSAKMP provides distribution and enforcement of policies, key distribution, and key management for groups [23]. GSAKMP entities are the Group Owner, a Group Controller Key Server (GCKS), and the Group Members.

The Group Owner is responsible for creating security policies. The GCKS distributes and enforces policies. It manages the keying material and grants group access [23].

The GCKS can delegate subgroup-responsibilities to Group Members, making the protocol scale dynamically. GCKS assumes a dynamically-changing number of Group Members.

All messages exchanged during a join operation are signed by the sender. At the end of the exchange the new group member sends an acknowledgement to the server. Without an acknowledgement message within a specific time interval, the GCKS can send an error notification to the member.

GSAKMP is centralized and push-oriented. Several key management techniques can be used in order to make the re-keying efficient. Some examples are Logical Key Hierarchy (LKH) or One-way Function Trees (OFT) [7].

The high complexity and computational overhead of GSAKMP make this protocol not applicable to constrained environments. However, the acknowledgement message is a

well-suitable feature for constrained environments. It allows notifying the group controller in case of unsuccessful key distribution, supporting a reliable execution of GKM functionalities.

B. Asynchronous Key Distribution Mechanisms

The following part presents two asynchronous protocols.

1) ARF: The work of Tanaka and Sato [18] introduces an Asynchronous Rekeying Framework (ARF). It works on top of reliable and totally ordered multicast protocols (RTOMP). The authors focus on synchronizing all group members for re-keying.

An out-of-sync problem arises when a group member cannot decrypt a message because it was secured with a new key not received yet, or an old key that was already substituted. The designers of ARF claim that, especially for very dynamic groups, synchronizing all the members after every key update is costly and unnecessary. Instead, they propose a framework in which a group-member is responsible for verifying the validity of its group key before sending or receiving any message [24].

ARF Group Members are partitioned into subgroups, called *domains*, with a separate KDC each. Each domain shares an individual key. A KDC communicates via RTOMP, which provides reliability and ordering of messages [24].

When membership changes, a domain's KDC generates a new group key and distributes it to all other domains' KDCs. When a member decides to send a message, it sends a sequence number request to its KDC to check the recency of its key. When receiving a member-request using an old group key, the KDC (re-) sends the new credentials to the group. When a member receives a message that it cannot decrypt with any of its valid keys, it sends a request to the KDC and asks for new keys.

ARF enables the group members to check for the recency of their key. This mechanism ensures group-consistency at the cost of a high message overhead.

2) Bird et al.: The study in [25] considers synchronous and asynchronous key distribution models for constrained environments. This section focuses on the asynchronous pull-based mechanism.

They define a scenario described as A-B-K, where A and B are group members and K is a KDC. One member is responsible for initiating the PULL on behalf of the whole group. This member distributes the keys to all other members.

In the A-B-K pull scenario, it is assumed that A does not contact the KDC directly but is willing to let B contact the KDC on its behalf [25]. A contacts B and sends the payloads necessary for member-member authentication. Then, B starts an exchange with the server and receives the current group keys.

An advantage of this method is that no encryption or decryption operations are required. The exchanged messages are protected with Message Authentication Codes [25]. Tickets and nonces are used to ensure authenticity among group members, and between the group members and the KDC.

The approach presented in [25] does not include any reliability mechanism.

C. Packet Loss Recovery

An important component to ensure a secure execution of GKM protocols is a reliable communication channel. In particular, recovery from packet losses is an essential property in constrained networks, where the communication channel is lossy and unreliable [6]. On a lossy channel, without recovery mechanism, a lost key message results in exclusion of a member from the group. The mechanisms presented in this subsection provide error-recovery and channel-reliability in unreliable environments.

1) ELK: ELK [26] targets reliable key distribution for large groups. It uses a KDC. ELK is based on the idea of allowing a receiver to recover a lost one-session key with the support of hint messages from other group members [27].

An important concept in the ELK protocol is the usage of key trees. Key trees are data structures used to control and optimize the update of security credentials [26].

An ELK KDC maintains a key tree. Each node represents a symmetric key [26]. Every member is associated with a leaf node. The group key is at the root of the tree. Every member knows the keys from its leaf node to the root. All inner tree-nodes represent Key Encryption Keys, used to ensure that only legitimate members can retrieve the group key.

When a member joins the group, it receives all the keys from its leaf node to the root. When a member leaves the group, all the keys known by this member are updated [26]. Key trees allow a more efficient key distribution. In ELK, every key in the tree can be reconstructed with *hint* messages from the left and right children [26].

The authors of ELK consider just the recovery of the last session key. In case a member lost more than one session key, ELK reverts to a unicast mechanisms like Keystone [28] for re-initialization of the member. In [27], the authors extend this mechanism to reconstruct multiple session keys.

2) STORM: Structure-Oriented Resilient Multicast (STORM) [29] proposes an efficient error-recovery mechanism for real-time applications. It allows senders and receivers to collaboratively recover from packet losses [29].

STORM targets minimizing both, the overhead of control packets, and the delay in packet recovery. The authors argue that the loss-detection and the re-transmission strategy are two important aspects in the design of a secure multicast protocol [29].

The problem of loss-detection is addressed by allowing members suffering from packet losses to send negative acknowledgement (NACK) messages to the others. Other members can send repair packets to help the loss-suffering members to reconstruct the key update message.

Sending NACK and repair messages to the whole group can result in significant overhead on lossy channels. This can be solved by distributing the recovery messages according to a structure that is overlaid on the group end-points. Members

can self-organize into a tree-based structure to recover lost packets from adjacent nodes [29].

Moreover, the group structure is dynamic and it can be adapted to the membership changes and network traffic conditions. STORM solves the problem of reliable delivery. However, it suffers from high complexity, scalability issues, and is vulnerable to denial-of-service attacks.

3) Keystone: Keystone, [28] establishes a secure multicast channel by adopting forward error-correcting codes (FEC). Still, self-correcting codes cannot ensure a complete reliability [28].

In order to overcome this issue, the authors also propose a re-synchronization mechanism. This technique is supposed more efficient to recover a lost group key in comparison to inefficient re-transmission requests [28]. A re-transmission would require an additional signature and verification of a message - a cryptographically expensive operation.

Keystone sends a re-synchronization request to the KDC. The KDC responds by encrypting the security credentials with the client's individual key [28]. No expensive signature has to be generated, resulting in better performance.

D. Group Consistency in Asynchronous Environments

In asynchronous environments, the group members independently retrieve their own key from the KDC.

In asynchronous mechanisms group members are enabled to frequently issue KDC requests for checking the validity of the own key in order to ensure consistency [18]. The resulting overhead is a problem for constrained networks.

In another variant, group members update the group-key with a pull operation before the current key expiration. In this case, if one group member fails to download the new group key, the group state is inconsistent and the faulty members are excluded from the communication [18].

The following part therefore considers another group consistency ensuring mechanism.

1) Ong and Goh: The authors of [30] propose a protocol for an efficient, deadlock-free group key management. It ensures group consistency during the asynchronous key update procedure. Their protocol includes a new entity, an *arbitrator*. An arbitrator is a node that other nodes use to determine the current group key.

Some of the group members are statically or dynamically elected as arbitrators. Consequently, no single node is able to dictate to the other nodes which key to use [30].

The protocol consists in the following steps:

1) A node contacts an arbitrator and sends a *Multicast-Key Change Request* including a priority value.
2) Each arbitrator keeps a prioritized queue of key-change requests. An arbitrator replies with a *Permit* message to the highest priority request. The permit response contains a seed generated by the arbitrator.
3) The arbitrator responds with a *Fail* to all the other requests.
4) When a node receives a fail message, it sends a *Release* to all the arbitrators.
5) When a member receives the required number of permit messages and seeds, it can generate the new multicast-key.
6) At this point, this member can send a *Multicast-Key Change* to all other members, including the new key.

Regarding the key update mechanism, the authors realize that during the propagation of the seed among the group members, there is a time window when the group state is inconsistent. For ensuring that the new key is not used until everyone in the group has updated it, they use a *Key Switch* message [30]. It informs the members of the group that it is time to switch to the new key. This message is the last one to be protected with the old key.

E. Assessment

Table I summarizes the protocol assessment. As described before, the critical feature for the regarded constrained IoT environments is reliability. As shown, reliability is only partially provided.

Especially the centralized synchronous protocols provide reliability by making the KDC keep track of members with inconsistent keys, and resending messages in case of failures. Other approaches for reliability consist in using acknowledgement messages for notifying the successful completion of the protocol. However, the presented approaches do not scale well in the envisioned large and highly-dynamic settings. Moreover, this model generally presents a relevant message overhead for failure recovery in constrained networks, where the communication channels are unreliable.

As introduced before, synchronous mechanisms are not always applicable. The asynchronous approach allows the group members to independently initiate the distribution and update of the group key. However, in asynchronous models there is always a window of time in which the key is inconsistent. In fact, the group members do not initiate the GKM operations in a synchronized way.

This paper targets GKM for critical resource-constraint infrastructures. The previous discussion of synchronous and asynchronous GKM mechanism shows that the existing approaches either miss reliability mechanisms, or the mechanisms are unsuitable for the targeted dynamically-changing constrained setting.

For centralized synchronous protocols, suitable mechanisms exist but do not scale for the targeted dynamically-changing settings. None of the asynchronous protocols ensures reliability for constrained environments. In lightweight centralized-asynchronous protocols, reliability mechanisms are entirely missing so far.

V. Conclusion

Group Key Management is an essential for securing IoT systems. In particular, safety-critical resource-constrained infrastructures require a reliable execution of GKM operations.

This paper provided a structured overview of GKM protocols considering different key distribution models as classification criteria. Representative centralized and distributed archi-

Assessment	MIKEY	GDOI	GSAKMP	ARF	A-B-K	ELK	STORM	Keystone	Ong & Goh
Centr./Distr.	Distr.	Centr.	Centr.	Centr.	Centr.	Centr.	Centr.	Centr	Distr.
Synch./Asynch.	Synch.	Both	Synch.	Aynch.	Asynch.	Synch.	Synch.	Synch.	Asynch.
Reliability	Not specified	Not specified	Ack. for success or failure	Key Validity Check	Not specified	Hint msg.	Loss Recovery	Re-synch and FEC	Switch msg.
Lightweight	Yes, 1RTT	No, P1 high msg overhead	No, sign. every msg.	No, high msg. overhead	Yes, one PULL/-group	Yes, small hint msg.	No, high complexity	No, sign every msg.	No, high msg. overhead

TABLE I
ASSESSMENT OF GKM PROTOCOLS

tectures with asynchronous and synchronous key distribution models were assessed.

The assessment of the presented protocols reveals that further research is needed in the field of reliability for centralized-asynchronous GKM protocols for dynamically-changing constrained IoT settings.

REFERENCES

[1] M.-O. Pahl and S. Liebald, "Information-centric iot middleware overlay: Vsl," *2019 International Conference on Networked Systems (NetSys)*, pp. 1–8, 2019.

[2] M.-O. Pahl, "Multi-Tenant IoT Service Management towards an IOT App Economy," in *Hot Topics in Network and Service Management (HotNSM) at International Symposium on Integrated Network Management (IM)*, Washington DC, USA, Apr. 2019.

[3] P. K. Muhuri, A. K. Shukla, and A. Abraham, "Industry 4.0: A bibliometric analysis and detailed overview," *Engineering Applications of Artificial Intelligence*, vol. 78, no. November 2018, pp. 218–235, 2019.

[4] M.-O. Pahl and L. Donini, "Giving IoT Edge Services an Identity and Changeable Attributes," in *Int. Symposium on Integrated Network Management (IM)*, 2019.

[5] C. Bekara, "Security Issues and Challenges for the IoT-based Smart Grid," in *International Workshop on Communicating Objects and Machine to Machine for Mission Critical Applications*, 2014.

[6] C. Bormann, M. Ersue, and A. Keranen, "Terminology for Constrained-Node Networks," RFC 7228, 2014.

[7] S. Rafaeli and D. Hutchinson, "A Survey of Key Management for Secure Group Communication," in *ACM Computing Surveys*, vol. 35, no. 3, 2003.

[8] R. Aparna and B. B. Amberker, "Analysis of Key Management Schemes for Secure Group Communication and their Classification," in *Journal of Computing and Information Technology*, 2009.

[9] B. Daghighi, L. M. Kiah, and S. A. Rad, "Group Key Management Using Public Key Exchange," in *International Journal of Computer Theory and Engineering*, vol. 5, no. 5, 2013.

[10] R. Seetha and R. Saravan, "A Survey on Group Key Management Schemes," in *Cybernetics and Information Technologies*, vol. 15, no. 3, 2015.

[11] R. Barskar and M. Chawla, "A Survey on Efficient Group Key Management Schemes in Wireless Networks," in *Indian Journal of Science and Technology*, 2016.

[12] T. T. Mapoka, "Group Key Management Protocols for Secure Mobile Multicast Communication: A Comprehensive Survey," in *International Journal of Computer Applications*, 2013.

[13] J. Zhang and V. Varadharajan, "Wireless sensor network key management survey and taxonomy," in *Journal of Network and Computer Applications*, 2010.

[14] W. Diffie and M. E. Hellman, "New Directions in Cryptography," in *IEEE Transactions on Information Theory*, vol. 22, no. 6, 1976.

[15] T. Hardjono and L. R. Dondeti, *Multicast and Group Security*. Artech House Computer Security Series, 2003.

[16] Y. R. Yang, X. S. Li, X. B. Zhang, and S. S. Lam, "A Reliable Group Rekeying: A Performance Analysis," in *ACM SIGCOMM Computer Communication Review*. University of Texas at Austin, 2001.

[17] C. Li and U. Nguyen, T., "Group key management in wireless mesh networks," in *The Fourth International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, 2007.

[18] S.-Y. Tanaka and F. Sato, "A Key Distribution and Rekeying Framework with Totally Ordered Multicast Protocols," in *Proceedings 15th International Conference on Information Networking*, 2001.

[19] J. Arkko, E. Carrara, F. Lindholm, M. Naslud, and K. Norrman, "MIKEY: Multimedia Internet KEYing," RFC 3830, 2004.

[20] R. Falk and S. Fries, "Security Considerations for Multicast Communication in Power Systems," in *International Journal on Advances in Security*, vol. 6, no. 3,4, 2013.

[21] B. Weis, S. Rowles, and T. Hardjono, "The Group Domain of Interpretation," RFC 6407, 2011.

[22] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)," RFC 2409, 1998.

[23] H. Harney, U. Meth, A. Colegrove, and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol," RFC 4535, 2006.

[24] M. Taghdiri and D. Jackson, "A Lightweight Formal Analysis of a Multicast Key Management Scheme," in *Formal Techniques for Networked and Distributed Systems*, 2003.

[25] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung, "The KryptoKnight Family of Light-Weight Protocols for Authentication and Key Distribution," in *IEEE/ACM Transactions on Networking*, 1995.

[26] A. Penrig, D. Song, and D. Tygar, "ELK, a new protocol for efficient large-group key distribution," in *Proceedings 2001 IEEE Symposium on Security and Privacy*, 2000.

[27] J. Hur and Y. Lee, "A reliable group key management scheme for broadcast encryption," in *Journal of Communications and Networks*, vol. 18, no. 2, 2016.

[28] C. Wong and S. Lam, "Keystone: a key management service," in *Proceedings International Conference of Telecommunications*, 2000.

[29] X. Rex, X., M. C., Andrew, H. Zhang, and R. Ravatkar, "Resilient multicast support for continuous-media applications," in *Proceedings of 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, 1997.

[30] S. H. Ong and S. H. Goh, "A generic multicast-key determination protocol," in *Proceedings of IEEE Singapore International Conference on Networks/International Conference on Information Engineering '93*, 1993.