Pub/Sub meets MLS: End-to-End Encrypted Group Data Sharing over Publish-Subscribe

Kazuaki Ueda[*] Chikara Sasaki[*] and Atsushi Tagami[*]
[*]KDDI Research, Inc.
Email: {kz-ueda, ch-sasaki, tagami}@kddi-research.jp

***Abstract*—In Cyber-Physical Systems (CPS), numerous devices gather sensor data into the cyber domain, which is then consumed by various services, such as robots and analytic services, in a dynamic and real-time manner. The Publish-Subscribe (Pub/Sub) model is a promising communication paradigm that can easily support many-to-many data sharing. By introducing the intermediate message broker responsible for message forwarding, Pub/Sub can decouple the message publisher and its subscribers and achieve scalable data sharing. However, ensuring the confidentiality of messages against the broker presents a key challenge when distributing private information over Pub/Sub. End-to-end encrypted (E2EE) Pub/Sub protocol is a promising approach to support message confidentiality in Pub/Sub. Within CPS, sensor data is dynamically consumed by multiple services, and thus E2EE Pub/Sub should support dynamic changes in subscribers in a scalable manner. However, existing protocols have scalability issues due to the excessive number of messages on the order of N^2 for N subscribers during key updates. This paper presents a new E2EE Pub/Sub protocol designed to support dynamic data sharing in CPS, inspired by the state-of-the-art secure group messaging protocol, Messaging Layer Security (MLS). By carefully integrating MLS features into Pub/Sub, our protocol effectively handles dynamic subscriber changes in a scalable manner, updating keys with only $O(N)$ messages.**

***Index Terms*—Publish-Subscribe, Messaging Layer Security, End-to-End Encryption**

I. Introduction

With the rapid proliferation of the Internet of Things (IoT) and the increasing attention to Cyber-Physical Systems (CPS) [1], the role of communication is drastically changing. In CPS, numerous IoT sensors collect real-world data and transmit it to the cyber domain, where various services dynamically consume this data in real-time. For example, surveillance camera footage can be utilized for both safety monitoring services and facility pathway management.

The Publish-Subscribe (Pub/Sub) model is a promising communication paradigm that enables asynchronous sharing of a large number of data, which is different from the client-server model designed for connecting two endpoints. Pub/Sub decouples data publication and distribution into two distinct nodes: publishers and brokers. The publisher, such as an IoT sensor, is solely responsible for publishing the data within a designated *topic* and sends the publication message to the broker. Subsequently, the broker disseminates the received publication messages to subscribers of the corresponding topic. By scaling up the number of brokers, Pub/Sub can easily enhance the scalability of data distribution. Pub/Sub has been used in various domains, including industrial IoT [2], [3], big data processing [4], [5], and media distribution [6]. Additionally, several proposals have emerged for the Future Internet to support Pub/Sub functionality within the network [7], [8]. Pub/Sub will play an important role in future Internet communications.

ISBN 978-3-903176-63-8 © 2024 IFIP

Since Pub/Sub communication relies on a broker for data distribution, ensuring message confidentiality is extremely important for distributing sensitive information. End-to-end encryption (E2EE) is a promising method to ensure message confidentiality [9]. E2EE involves encrypting data from the sender to the receiver, preventing interception or eavesdropping by intermediaries. Since only the sender and receiver who share the cryptographic key can decrypt the encrypted data, data confidentiality is achieved. To ensure the message confidentiality in Pub/Sub, the End-to-End encrypted Pub/Sub [10]–[12] (E2EE Pub/Sub) have been proposed. E2EE Pub/Sub uses the topic's encryption and decryption keys to implement end-to-end encryption between publisher and subscribers. The publisher encrypts the message payload with the topic's encryption key and forwards the message to the broker. Only the message headers necessary for data delivery are accessible by the broker, and the message payload is kept secret by encryption. The legitimate subscriber obtains the topic's decryption key from the key server and uses it to decrypt the message payload, thereby consuming the data. We can distribute private information through the E2EE Pub/Sub even if brokers act as *semi-honest* nodes.

E2EE is an effective solution for ensuring message confidentiality in Pub/Sub, but it poses a significant challenge of decryption key compromise. Once the decryption key is compromised, an attacker can access the data unless the key is updated. Therefore, designing an E2EE Pub/Sub and its key management is important to support secure properties against key compromise, such as forward secrecy and post-compromise security. Additionally, CPS use cases involve dynamic data sharing by multiple services, resulting in frequent changes to subscriber groups, further complicating the satisfaction of these secure properties. When a subscriber leaves and loses access rights to subsequent data, the revocation of keys held by that subscriber must be processed. Therefore, E2EE Pub/Sub needs to handle the above revocation process efficiently. However, we have observed that existing E2EE Pub/Sub protocols encounter scalability issues due to the excessive number of messages required for key updates when subscriber groups change.

In this paper, we propose an extended E2EE Pub/Sub system and its scalable key update protocol, SKUA, to support dynamic secure group data sharing over Pub/Sub. Our contributions are as follows: First, we clarify the potential issue of key updates in existing E2EE Pub/Sub systems based on the inherent communication characteristics of Pub/Sub. Next, We design a new E2EE Pub/Sub and its scalable key management scheme inspired by state-of-the-art secure group messaging protocols, Messaging Layer Security (MLS) [13] and sender key [14], [15]. We evaluate the effectiveness of our proposed scheme through simulations.

II. End-to-End Encrypted Pub/Sub

To address the requirements of CPS, where many devices generate and consume data, a new communication paradigm is needed to support efficient, many-to-many communication between devices. Pub/Sub model is designed for asynchronous communication with the topic associated with the information as a communication identifier. Pub/Sub introduces a broker that relays the publication messages between publishers and topics' subscribers. By decoupling the publishers and subscribers and delegating the communication state to the brokers, Pub/Sub can provide simple communication, thus it can support many-to-many communication efficiently. While delegating message transmission to the brokers can reduce the complexity of the communication, it makes private data and critical control messages accessible to them. Currently, data confidentiality is not an issue in Pub/Sub because all Pub/Sub entities, such as subscribers, publishers, and brokers, are managed by the same organization. However, it is impractical for all CPS services to operate many brokers for data sharing. New service providers will emerge that offer the Pub/Sub platform itself as a service, similar to the current cloud services. Towards the future CPS services that distribute private information through the Pub/Sub platform, new security mechanisms will be required to ensure data confidentiality even for the *semi-honest* brokers.

A. End-to-End Encryption and its application to Pub/Sub

E2EE is a secure communication method that ensures the content of a message is only accessible to the sender and the intended recipients [9]. In the E2EE system, the message is encrypted by the senders and can only be decrypted by the recipients, with no intermediate entities accessing to the message. Even if the communication is intercepted during transmission or cached on intermediary servers, the content remains confidential and unreadable to anyone other than the sender and the recipients. E2EE provides a high level of privacy and security for sensitive communications, such as private messages, voice calls, or file transfers. It is commonly used in messaging apps, email services, and other communication platforms [9], [15].

Applying E2EE to Pub/Sub is a promising direction to provide message confidentiality against the semi-honest brokers. Several attempts have been conducted to construct E2EE Pub/Sub [2], [10], [11]. Fig. 1 shows the overview of E2EE Pub/Sub. Regarding message transmission, E2EE Pub/Sub is identical to the original Pub/Sub, and publication messages are delivered to subscribers via brokers according to their subscription status to the topic. On the other hand, the payload *encryption* at the publisher, the payload *decryption* at the subscriber, and key management at the *key store* are new functions of E2EE Pub/Sub. In terms of encryption and decryption keys, both asymmetric and symmetric keys can be used depending on the security requirements of the services. These keys are stored and managed in the key store. Once subscribers have been authenticated as legitimate subscribers of the topic, they can obtain the decryption key from the key store and access the content.

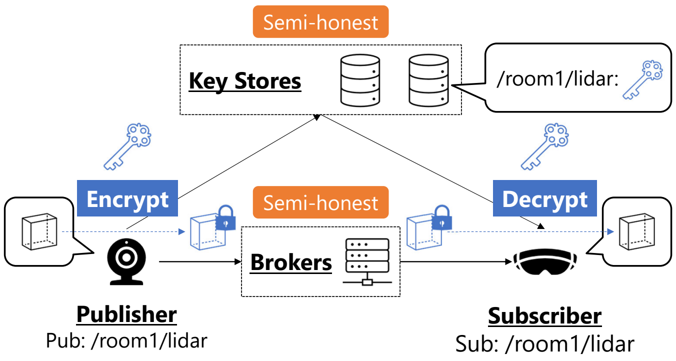

Fig. 1. E2EE Pub/Sub framework

Next, we explain the threat model of E2EE Pub/Sub. We assume that the system grants access to content only to legitimate subscribers through authentication. The key store distributes keys only to legitimate subscribers since anyone can access the data by obtaining its decryption key. We also assume that the broker and key store are operated by different organizations and cannot collude. Furthermore, we assume that legitimate subscribers are semi-honest nodes and do not intentionally leak their decryption keys to other users. However, since subscribers attempt payload decryption based on their retained keys, key diversion or misuse of revoked keys may lead to unintended content access. Moreover, a proper key revocation scheme is required to reflect the status of subscribers' access rights.

There are multiple approaches in designing key management in E2EE Pub/Sub. The authors of [10] propose ENTRUST, which utilizes topic keys, key management with a key server, and key exchange protocol over Pub/Sub. ENTRUST assigns a symmetric key to each topic, referred to as a *topic key*, which is managed and updated by the central key server. Both publishers and subscribers encrypt and decrypt the message payload by retrieving the topic key from the key server. In this system, the key server has keys and associated topics. Therefore, the key server can access the publication message and must be an *honest* node. In addition, since the key server updates and stores keys, publishers and subscribers must periodically obtain topic keys from the key server for each key lifetime.

The authors of [11] present the SEEMQTT, which is designed to decouple the key server's functions, i.e., key update and key storage, into distinct nodes. Within this framework, key storage is delegated to multiple *key stores*, and by introducing secret sharing, each key store only holds a fragment of a topic key. Therefore, each key store does not possess a complete topic key and cannot decrypt a publication message. Furthermore, the key update function can be delegated to the publishers, enabling decentralized key updates by the publishers.

B. Problem statement: limitation of existing approaches

As described above, the existing E2EE Pub/Sub can achieve message confidentiality for brokers by applying E2EE to publishers and subscribers However, the existing methods fail to address the following three challenges against the key compromise;

- Ensuring three secure properties against key compromise: Forward Secrecy (FS), Backward Secrecy (BS), and Post-Compromise Secrecy (PCS),
- Addressing frequent key compromise events stemming from the unique communication pattern of Pub/Sub,
- Addressing simultaneous key compromise events arising from the large-scale many-to-many data sharing in CPS.

First, we outline three essential secure properties that E2EE Pub/Sub systems must adhere to in the event of key compromise. FS protects past publication messages against future key compromises [16]. FS is achieved by the short-term ephemeral keys derived irreversibly through key updates rather than relying on long-term keys. An extension of FS, Perfect Forward Secrecy (PFS), enhances security by changing the ephemeral key for each message. BS ensures that newly joined subscribers cannot access past publication messages. BS is also accomplished by updating ephemeral keys upon adding new subscribers, as new subscribers should not have access to past keys. PCS guarantees that evicted subscribers no longer have access to publication messages and topic keys. PCS assumes compromise of the key derivation function of ephemeral keys, as in the case of evicted subscribers, necessitating an update to the master secret for key derivation.

Next, we will explore the relationship between the Pub/Sub communication model and the frequency of key compromise events. In Pub/Sub, a single publication message is distributed to numerous subscribers through a broker. Therefore, in E2EE Pub/Sub, all subscribers need to share the decryption key. As keys are shared among subscriber groups, any changes in group membership result in key compromise events. Consequently, in E2EE Pub/Sub, key compromise frequently occurs not only from traditional key leaks due to hacking but also from events such as subscribers joining and leaving the group. As mentioned in the above three secure properties, key updates in the event of a key compromise is a critical issue that significantly impacts message confidentiality. Given the high frequency of key compromise, establishing a scalable mechanism to update and share decryption keys with appropriate subscribers is an important challenge in E2EE Pub/Sub.

Finally, we discuss the issue of simultaneous key compromise events in future CPS use cases. In the current Pub/Sub-based service, a single service provider distributes its own data through Pub/Sub to realize a service. As a service consumes multiple topic data, a one-to-many relationship exists between services and topics. In contrast, since each topic is solely utilized by a specific service, the relationship between topics and services remains one-to-one. However, in future CPS use cases, IoT data managed by various organizations is expected to be shared through the data lake and consumed by multiple services. The authors of [17] propose a privacy-aware space-sharing application. It shares multiple 3D objects in the user's room, such as persons and furniture, scanned by 3D sensors with many participants. 3D conferences can be conducted as if everyone were in the same 3D space. There are two challenges to such an application: a large number of 3D object sharing among participants and appropriate access control concerning the sensitivity of objects. To address such future use cases, it is essential to support access control when the relationship between topics and services is many-to-many. When multiple services consume a single topic, the impact of key updates becomes significant. When there are changes in the subscriber group within a service, such as join or leave, updating the keys for all topics consumed by that service is necessary. Since multiple services consume each topic, a change in the subscriber group of a service results in a ripple effect of updating numerous topic keys and sending them to many subscribers across multiple services.

We will clarify the issues of existing schemes in terms of the three key challenges stated above. In ENTRUST, the centralized key server performs key updates and distribution. FS and BS can be achieved by setting shorter lifetimes for topic keys, thereby realizing them as ephemeral keys. However, ephemeral keys are distributed from the key server to subscribers and publishers, resulting in a large amount of key distribution traffic proportional to the number of nodes occurring periodically at intervals corresponding to the short lifetime. Moreover, since the management of topic keys is performed by the centralized key server rather than the message publisher, it is hard to achieve PFS, where keys are changed for each message. On the other hand, SEEMQTT achieves a decentralized key update by the publisher. Although not discussed in detail in the paper, this scheme can be extended to realize PFS and BS by updating keys on a message-by-message basis. The amount of traffic for ephemeral key distribution can also be suppressed by generating ephemeral keys locally with a Key Derivation Function (KDF) and master secret. Thus, from the FS and BS perspectives, a design based on a decentralized publisher-driven key update is considered a promising approach.

PCS is not adequately addressed in all existing schemes. Both centralized and decentralized approaches require functionality to manage the state of topic subscribers and issue new master secrets and keys when subscriber groups change. In addition, distributing newly derived secrets and keys in a scalable manner is an essential challenge to realize E2EE Pub/Sub. In the existing methods, changes in subscriber groups

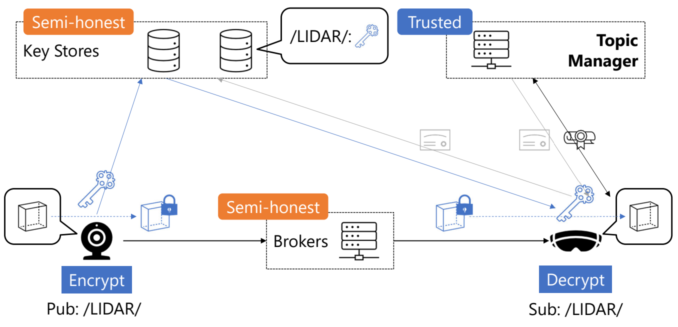

Fig. 2. Overall of our proposed E2EE Pub/Sub

result in numerous key update messages. The number of key update messages can be estimated as follows: If a service has N users and consumes M topics, it is required to send M new keys to $N-1$ users during key update upon the user's leave. As a result, at least $M \times (N-1)$ messages are generated, with an order of $O(N^2)$. Thus, existing schemes are insufficient to support FS, BS, and PCS. It is necessary to design a method that satisfies these requirements and addresses the new issue of message explosion caused by key updates due to the characteristics of Pub/Sub.

III. SKUA: Extended E2EE Pub/Sub

In this section, we propose a new E2EE Pub/Sub and its scalable key update protocol, Service-based Key Update Aggregation (SKUA), which addresses the challenges of existing schemes: the excessive number of messages required for key updates while maintaining the secure properties of E2EE. We integrate the functionalities of state-of-the-art secure group messaging protocols, MLS [13] and sender key [14], [15] to efficiently handle key updates in response to changes in the subscriber group. We mitigate the message traffic for key updates with two new features: offline-based key updates and aggregation of key update events.

Fig. 2 shows the overview of our scheme. In order to satisfy FS, BS, and PCS, topic keys must be updated correctly to meet the subscriber group status. We introduce the topic manager, which authenticates legitimate subscribers and manages subscriber groups for the topic. It handles a variety of group events that lead to key updates, including group member join, leave, and key revocation. We also use key stores, which securely distribute topic keys and secrets to the subscribers.

We design topic key updates in E2EE Pub/Sub by following the design of the sender keys used in E2EE messaging apps [15]. We assign a master secret and a symmetric key for a topic, the topic secret ts and the ephemeral topic encryption key tk. ts is used to derive tk, and separation of the secret and the encryption key makes it difficult to obtain the next one, even if an encryption key is leaked. After the authentication with the topic manager, the legitimate subscribers retrieve the ts from the key stores and derive the tk to decrypt messages. Fig. 3 shows the message structure of our scheme. Both information on keys and encrypted payload are stored in the private message field in a publication message to avoid access from intermediaries.

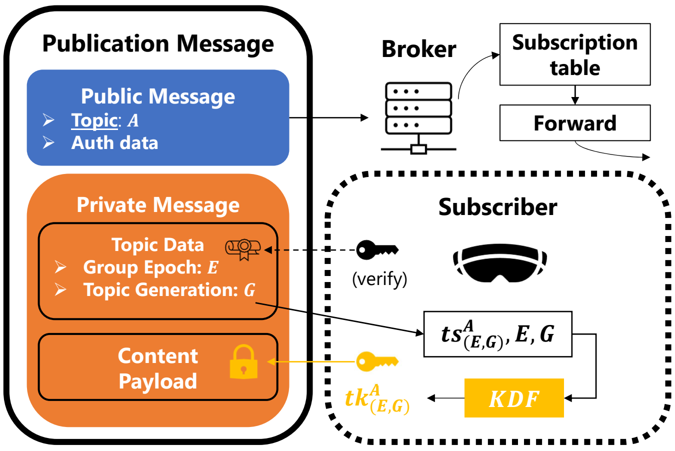

Fig. 3. Detailed structure of the encrypted publication message

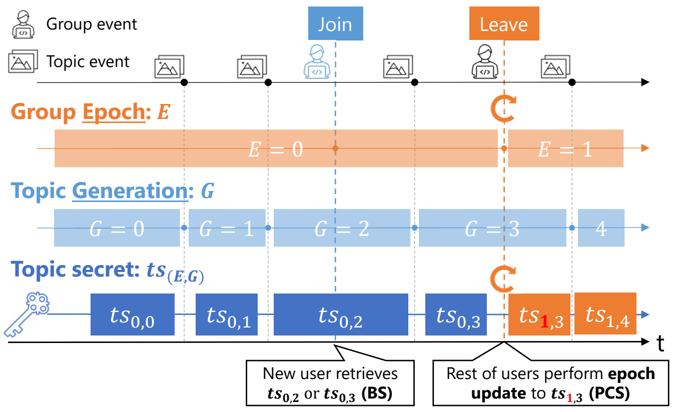

Fig. 4. Two types of key updates: topic generation and group epoch

A. Key Updates with Topic Generation and Group Epoch

Fig. 4 shows the overview of our key update procedures. We use two types of key updates to achieve FS, BS, and PCS: topic generation and group epoch. The topic generation-based key update is used to support FS and BS. A topic generation represents the ephemeral identifier of a topic message that is defined by the publisher. This update is performed periodically offline by the publisher and subscribers and thus does not require control traffic. The group epoch-based key update is used to support PCS. A group epoch represents the subscriber group status which is changed in response to the users' leave. This update is performed upon the topic manager's notification and requires control messages.

1) Topic generation-based key update: In the topic generation-based key update, we derive an ephemeral topic encryption key of the corresponding topic generation to support FS and BS. A topic secret ts and a topic encryption key tk are periodically updated with a Key Derivation Function (KDF). Following the design of the double ratchet algorithm [18], we use the HMAC-based Extract-and-Expand Key Derivation

Function (HKDF) SHA256 as a KDF. ts and tk are derived as follows:

$$ts^{A}_{i,j} \leftarrow \mathrm{HKDF}(\mathrm{ts^{A}_{i,j-1}, j-1, "secret", L_{sec}}), \quad (1)$$

$$tk^{A}_{i,j} \leftarrow \mathrm{HKDF}(\mathrm{ts^{A}_{i,j}, j, "key", L_{key}}). \quad (2)$$

The $ts^{A}_{i,j}$ denotes a topic A's secret in group epoch i and generation j, $tk^{A}_{i,j}$ denotes a topic A's encryption key in group epoch i and generation j, L_{sec} denotes the length of topic secret, and L_{key} denotes the length of topic encryption key, respectively. As we can see from the above equations, we can derive the next topic secret from the previous topic secret. The topic encryption key is derived from the topic secret of the corresponding generation. The KDF derives the topic secret of generation j with input of a topic secret of previous generation $j-1$, the value of generation j, the label, and the key length. We use HKDF-Expand-Label as a Key Derivation Function. This key update can be performed offline by the publisher and subscriber.

Regarding the BS, our ephemeral key can limit the effect of the key leakage. In Fig. 4, a new user subscribes to the topic at generation $G = 2$. In this case, the new user can access only the published data for $G = 2$. By updating the topic generation frequently, the accessible data is limited, and thus, BS can be achieved. If the service wants to support strict BS, we have an option to invoke a generation update upon the user's join. In this case, a group manager notifies a publisher to perform a generation update for $G = 3$. Moreover, a new user retrieves $ts_{0,3}$ from the key store. Since our scheme is based on decentralized, publisher-driven key updates, strict BS can be easily supported.

2) Group epoch-based key update: Next, we update the topic secret with the group epoch to support *post-compromise security*. The newer version of ts must be kept secret to the compromised user who knows the previous ts. ts in the next epoch can be derived with a previous ts and a fresh random nonce as follows:

$$ts^{A}_{i+1,j} \leftarrow \mathrm{HKDF}(\mathrm{ts^{A}_{i,j}, n, L_{sec}}) \quad (3)$$

where $ts^{A}_{i,j}$ denotes the topic A's secret in the i-th group epoch, n denotes the random nonce, respectively. When the group state changes, the topic manager notifies the publisher and legitimate subscribers of a random nonce n through a secure channel. Both publisher and subscriber perform epoch updates offline to derive the new ts. Compromised users cannot derive the new topic secret because they do not receive the random nonce from the topic manager.

B. SKUA: Service-based Key Update Aggregation

Although our key update scheme can be performed offline by the subscriber, a notification message is required for the epoch updates of each topic. Therefore, the user leaving the service yields $O(N^2)$ messages, as explained in Sec.II. We propose a Service-based Key Update Aggregation method (*SKUA*) to improve the scalability of key management by suppressing message traffic. In the epoch update, a newer topic secret is derived with a random nonce and the previous topic secret. Therefore, if we use a fresh random nonce as an input for epoch updates for multiple keys, we can derive multiple topic secrets from a single notification message. The topic manager can track changes in the subscription groups, such as users joining or leaving. Therefore, we can reduce the control message by aggregating the epoch updates across multiple topics.

Algorithm 1 Service-based key update aggregation

```
Require: $srcNode, groupEvent$
 1: $targetNodes, isolatedNodes \leftarrow \emptyset$
 2: $topics = getRelatedTopics(groupEvent)$
 3: for $topic \in topics$ do
 4:     $subscribers \leftarrow getSubscribers(topic)$
 5:     if $srcNode \in subscribers$ then
 6:         $targetNodes = targetNodes \cup subscribers$
 7:     else
 8:         $isolatedNodes = isolatedNodes \cup subscribers$
 9:     end if
10: end for
11: $nonce1, nonce2 \leftarrow random()$
12: for $node \in targetNodes$ do
13:     $sendNotification(nonce1, node)$
14: end for
15: for $node \in isolatedNodes$ do
16:     $sendNotification(nonce2, node)$
17: end for
```

In SKUA, a topic manager manages service subscriptions. When the group state changes for services such as join and leave, the topic manager identifies groups of topics related to services and forwards a single *key update notification* to the subscribers of those groups of topics. Subscribers receive the key update notifications and perform epoch updates for the corresponding keys of each topic. Each subscriber can perform generation updates based on a nonce n and the previous topic key $ts_{i,j}$. Thus, a subscriber can update multiple topics at once from a single notification message. If a service has N users and consumes M topics, SKUA's key generation update process only needs one message to $N-1$ users. As a result, the number of messages is in the order of $O(N)$, which is $1/N$ to the conventional cost. The SKUA algorithm is shown in algorithm 1. This algorithm is executed within the topic manager to determine the group of key updates.

In our scheme, the key is dynamically updated and the message cannot be decrypted unless the publisher and the subscriber are in the same key state. In order to synchronize the key state, we store group epoch and topic generation in the topic data field of the publication message as shown in Fig. 3. Both epoch and generation are monotonically increasing with each update. When the publisher publishes a message, it stores the generation and epoch of the topic key as the topic

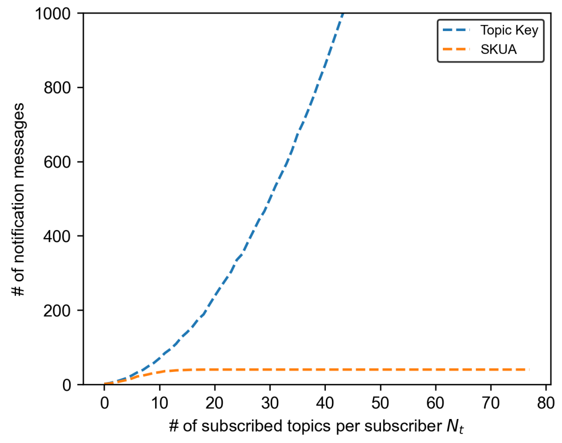

Fig. 5. The number of notification messages for an epoch-based update

data. Upon reception of the publication message, a subscriber decrypts it with a topic key corresponding to the stored key state. A subscriber compares the received generation value with its latest value stored locally. When there is a difference in the generation, the corresponding ts and tk are derived by performing generation updates by the number of differences.

IV. Evaluation

In this section, we evaluate the benefits of SKUA through simulation. We estimate the number of messages required for an epoch-based key update. We assume a scenario with 40 publishers and 40 subscribers, and each publisher publishes two topic data. We evaluated the average number of notification messages in an epoch update when subscribers subscribe randomly to N_t topics. Fig. 5 shows the number of notification messages for subscribed topics N_t. In $N_t = 10$, the existing method requires 58 messages, while SKUA requires 30 messages, and it can reduce the number of messages by 47%. On the other hand, for $N_t = 20$, the existing method requires 213 messages, while SKUA requires 40 messages. By aggregating the notification target, SKUA can limit the number of messages to the total number of subscribers. The greater the degree of overlap in the topics to which subscribers subscribe, the more effective SKUA provides

V. Conclusion

E2EE Pub/Sub is a promising approach to distribute private information through Pub/Sub. E2EE Pub/Sub needs to address frequent key compromises in a scalable manner while ensuring FS, BS, and PCS. This paper clarifies the scalability challenge posed by existing schemes, particularly in generating a large number of messages during key updates. We propose SKUA, which can achieve scalable key updates through distributed key derivation and aggregating key update events. As a future work, we plan to implement SKUA in the Pub/Sub messaging platform to evaluate performance overhead.

References

[1] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[2] "OPC 10000-14: UA Part 14: PubSub." [Online]. Available: https://reference.opcfoundation.org/Core/Part14/v105/docs/

[3] R. A. Light, "Mosquitto: server and client implementation of the mqtt protocol," *Journal of Open Source Software*, vol. 2, no. 13, p. 265, 2017.

[4] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, no. 2011. Athens, Greece, 2011, pp. 1–7.

[5] G. Baldoni, J. Loudet, L. Cominardi, A. Corsaro, and Y. He, "Facilitating distributed data-flow programming with eclipse zenoh: The erdos case," in *Proceedings of the 1st Workshop on Serverless mobile networking for 6G Communications*, 2021, pp. 13–18.

[6] L. Curley, K. Pugin, S. Nandakumar, V. Vasiliev, and I. Swett, "Media over QUIC Transport," Internet Engineering Task Force, Internet-Draft draft-ietf-moq-transport-03, Mar. 2024, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-moq-transport/03/

[7] H. Balakrishnan, S. Banerjee, I. Cidon, D. Culler, D. Estrin, E. Katz-Bassett, A. Krishnamurthy, M. McCauley, N. McKeown, A. Panda, S. Ratnasamy, J. Rexford, M. Schapira, S. Shenker, I. Stoica, D. Tennenhouse, A. Vahdat, and E. Zegura, "Revitalizing the public internet by making it extensible," *SIGCOMM Comput. Commun. Rev.*, vol. 51, no. 2, p. 18–24, May 2021.

[8] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From PSIRP to PURSUIT," in *Broadband Communications, Networks, and Systems: 7th International ICST Conference, BROADNETS 2010, Athens, Greece, October 25–27, 2010, Revised Selected Papers 7*. Springer, 2012, pp. 1–13.

[9] K. Ermoshina, F. Musiani, and H. Halpin, "End-to-end encrypted messaging protocols: An overview," in *Internet Science: Third International Conference, INSCI 2016, Florence, Italy, September 12-14, 2016, Proceedings 3*. Springer, 2016, pp. 244–254.

[10] M. Dahlmanns, J. Pennekamp, I. B. Fink, B. Schoolmann, K. Wehrle, and M. Henze, "Transparent end-to-end security for publish/subscribe communication in cyber-physical systems," in *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, ser. SAT-CPS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 78–87.

[11] M. Hamad, A. Finkenzeller, H. Liu, J. Lauinger, V. Prevelakis, and S. Steinhorst, "SEEMQTT: Secure End-to-End MQTT-Based Communication for Mobile IoT Systems Using Secret Sharing and Trust Delegation," *IEEE Internet of Things Journal*, 2022.

[12] L. Malina, G. Srivastava, P. Dzurenda, J. Hajny, and R. Fujdiak, "A secure publish/subscribe protocol for internet of things," in *Proceedings of the 14th international conference on availability, reliability and security*, 2019, pp. 1–10.

[13] R. Barnes, B. Beurdouche, R. Robert, J. Millican, E. Omara, and K. Cohn-Gordon, "The Messaging Layer Security (MLS) Protocol," RFC 9420, Jul. 2023. [Online]. Available: https://www.rfc-editor.org/info/rfc9420

[14] K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila, "A formal security analysis of the signal messaging protocol," *Journal of Cryptology*, vol. 33, pp. 1914–1983, 2020.

[15] D. Balbás, D. Collins, and P. Gajland, "Whatsupp with sender keys? analysis, improvements and security proofs," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2023, pp. 307–341.

[16] R. Anderson, "Two remarks on public key cryptology," University of Cambridge, Computer Laboratory, Tech. Rep., 2002.

[17] T. Amano, T. Mizumoto, S. M. Kala, H. Yamaguchi, T. Matsui, and K. Yasumoto, "Visual privacy control for metaverse and the beyond," *IEEE Pervasive Computing*, 2024.

[18] J. Alwen, S. Coretti, and Y. Dodis, "The double ratchet: security notions, proofs, and modularization for the signal protocol," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2019, pp. 129–158.