Hindawi
Security and Communication Networks
Volume 2022, Article ID 2644716, 18 pages
https://doi.org/10.1155/2022/2644716

Research Article

FSEE: A Forward Secure End-to-End Encrypted Message Transmission System for IoT

Li Cui,[1,2] Xing Qianqian,[1] Wang Yi,[1] Wang Baosheng,[1] Tao Jing,[1] and Liu Liu[3]

[1]*College of Computer, National University of Defense Technology, Changsha 410073, China*
[2]*College of Information and Communication, National University of Defense Technology, Xi'an 710106, China*
[3]*Sixty-third Research Institute, National University of Defense Technology, Nanjing 210012, China*

Correspondence should be addressed to Tao Jing; ellen5702@aliyun.com

Received 12 April 2022; Revised 14 May 2022; Accepted 25 May 2022; Published 8 August 2022

Academic Editor: Jie Cui

Leakage of long-term secrets is a major concern when cryptographic schemes are implemented on devices with weak protection capability, especially for resource-constrained IoT devices. Forward secrecy is a means to minimize the damage when such an event takes place. For pub-/sub-based IoT systems, several end-to-end (from publisher to subscriber) encrypted message transmission schemes have been proposed to tackle the confidentiality problems brought by malicious message brokers. But none of them provide forward secrecy. This article presents FSEE, a forward secure end-to-end encrypted message transmission system for pub-/sub-based IoT. To support FSEE, we design a novel group key exchange protocol BA-GKE, which relies on a semi-trusted key exchange server to provide forward secrecy and support asynchronous communication between group members. We prove its forward secrecy by ProVerif. The core idea of FSEE is to establish a forward secure symmetric key per device using BA-GKE asynchronously, and this device-specific key is shared with the device and its authorized subscribers for encrypting messages securely. By adding a semi-trusted key exchange server to realize BA-GKE in the current IoT architecture, FSEE does not need to change the existing message broker and could be deployed incrementally. The experimental results show that FSEE has comparable performance to existing prominent research and provides higher security.

1. Introduction

In order to realize large-scale communication between multiple entities in the Internet of Things (IoT), many IoT systems distribute messages based on the publish-subscribe (pub/sub) paradigm. A typical pub-/sub-based IoT system contains devices with sensors, user's mobile applications, and the message broker (generally deployed on the cloud platform) [1], and the network architecture is shown in Figure 1. Devices and applications perform as publishers and subscribers in the systems. Devices publish the sensor collected data to a specific topic (such as topic_stat), and authorized users' applications subscribe to the topic. With the help of the message broker's subscription collection and message forwarding, all the authorized users could get their interested messages. At the same time, the devices also subscribe to a topic with regard to control (such as topic_com), and authorized users control the devices by issuing control commands to the specific topic through the application. As device sharing is ubiquitous in IoT systems, a device is generally accessed by multiple authorized users (such as office members or maintenance workers et al.). The communication model in pub-/sub-based IoT systems is not one-to-one, but one-to-many (one device is accessed by multiple users), not direct communication between them, but through a middle message broker. Users' access rights to the device and revocation of the authorization are both managed by the device owner.

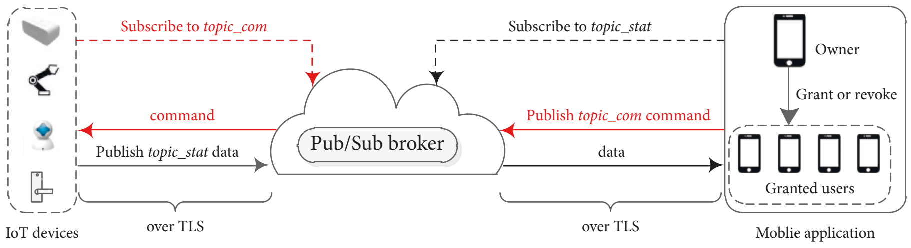


Figure 1: Typical architecture of pub-/sub-based IoT system.

Currently, the data transmitted between clients (publishers and subscribers) and the message broker in IoT systems are generally protected by TLS [1–3]. The publisher encrypts and sends the packets to message broker through the secure channel established between the two parties via TLS. The message broker gets the plaintext by decrypting the packets, then encrypts for each authorized subscriber, and sends the corresponding packet to the subscriber again via TLS. TLS is designed for one-to-one communication between two entities, and the message broker could get all the plaintext of each client. This approach forces us to trust the broker for data processing [4]. Since the IoT device manufacturers or IoT cloud platform vendors usually establish and maintain the broker, we cannot take its trustworthiness for granted.

Under normal circumstances, the data generated by IoT devices are usually closely related to users' living habits, whereas the message broker is not completely reliable. The centralized services provided by the message brokers perform like a black box for IoT users, where users do not have control over how brokers use their data [5]. Many Internet-accessible message brokers even do not deploy any security policies, and anyone can inject data or receive messages [6]. Some researchers show that the MQTT message brokers of a specific traffic monitoring system were disclosing the traffic flow of Mexico City [7]. Servers infected by spyware may leak secret credentials to an adversary, which may hamper the security of mission-critical data [8]. What's more, most of the current IoT systems are constructed based on the MQTT messaging protocol, and the MQTT protocol is not designed for hostile environments. Yan Jia et al. found that the MQTT protocol has serious flaws. Platforms using MQTT protocol can allow adversaries to access devices, launch large-scale denial of service (DoS) attacks, steal users' private information, and forge users' device status [9].

To address the privacy and confidentiality concerns brought by malicious message brokers, several end-to-end (from publisher to subscriber) encrypted message distribution methods have been proposed. Essentially, the existing methods can roughly be divided into three categories. The first category is that the publisher and subscriber coordinate to establish an encryption key [10–14]. The broker in the middle does not know the key and therefore cannot get the plaintext. This approach ensures that the messages are encrypted from the publisher to the subscriber, and only the communication endpoints own the cryptographic keys, which are necessary to decrypt the messages. There is no need to trust the underlying message broker. The second category is the re-encryption method [15–17]. The broker with some specific information (such as re-encryption keys) can transform the ciphertext of a publisher into the ciphertexts that can be decrypted by authorized subscribers. The third category is based on some trusted entities, such as trusted broker (as [3, 9]) or trusted hardware [18]. However, none of the existing end-to-end encryption schemes provide forward secrecy.

1.1. *The Need of Forward Secrecy in IoT.*

Leakage of long-term secret keys is a major concern when cryptographic schemes are implemented on devices with weak security protection capability (especially for resource-constrained IoT devices). Forward secrecy is used as a means to minimize the damage when such an event takes place. Forward secrecy ensures that the compromise of entities at this moment does not impact the security of the entities' previous completed communication. Most current designs of secure communication protocols consider forward secrecy an indispensable design goal [19]. Because adversaries with massive storage abilities and powerful penetration capabilities are increasing [20], long-term secrets leakage is often more likely to happen than expected: the keys may be stolen by system intruders or extracted from the stolen smartphones, law enforcement agencies may force users to reveal their key according to law, backup software may inadvertently upload a copy of the key to network storage, and so on [21].

1.2. *The Challenges for Achieving Forward Secure Key Exchange Asynchronously.*

At present, there are generally three methods to achieve forward secrecy in key establishment [20]. One is to extend the traditional DH key exchange protocol [22, 23], it requires entities to be always online to interact with each other, and asynchronous communication is not supported. This method is not suitable for IoT systems, as multiple authorized users of a device are not necessarily online at the same time, and the offline users will affect the communication of online users. The second method is to use precomputed key, which is also a common method to achieve forward secrecy without interaction. But this method is prone to message suppression attacks [20]. The third method is to convert a puncturable public key encryption scheme [24] into a puncturable key exchange scheme to provide forward secrecy [25]. However, because of the high computational complexity of puncturable encryption, this method is not suitable for low-power IoT devices. Naturally, we are interested in the following question:

1.3. Motivation Question. How to establish a forward secure session key asynchronously among a group while keeping low overhead for each member?

1.4. Our Scheme. Identity-based public key cryptography can greatly simplify the management of certificates by taking user's identity as its public key, whereas key exchange based on implicit authentication (such as HMQV [26]) can greatly reduce the communication overhead. In our work, combining BKEM [27] proposed by Colin *et al.* with identity-based implicit authentication [28], we propose an efficient forward secure asynchronous group key exchange protocol BA-GKE. Based on BA-GKE, a forward secure end-to-end encrypted message transmission system FSEE is designed for IoT. More details are given below.

1.5. The New Protocol: BA-GKE. BA-GKE is essentially an extension of DH-based key exchange in groups, which relies on a semi-trusted server to provide forward secrecy and support asynchronous communication of group members. We present a formal security validation of BA-GKE using ProVerif. The results show that it can provide mutual authentication for group members, guarantee the confidentiality of generated session keys, and provide forward secrecy at the same time.

1.6. The New Solution: FSEE. Based on the group key exchange protocol BA-GKE, we design FSEE, which utilizes BA-GKE to update the symmetric key between the device and its multiple authorized users in IoT. The core idea of FSEE is to establish a symmetric key per device that can be used to protect sensitive messages. This device-specific key is shared with the device and its authorized users. Due to the use of BA-GKE, FSEE adds a semi-trusted key exchange server to the current IoT architecture, does not need to change the existing message brokers, and can seamlessly integrate into existing ones. FSEE empowers the device owner the ability to control their data by controlling with whom to share the encryption key. Each device owner is responsible for authorizing and revoking access rights to its devices, without relying on a fully trusted third party.

1.7. Our Contributions. Specifically, our main contributions are as follows:

(1) Combining BKEM and identity-based implicit authentication, an efficient group key exchange protocol BA-GKE is proposed, which can establish a forward secure session key asynchronously among a group while keeping low overhead for each member.

(2) Based on the protocol BA-GKE, we propose FSEE, an end-to-end encrypted message transmission system for IoT to address the confidentiality concerns of malicious brokers and compromised clients. FSEE achieves confidential and forward secure end-to-end communication in IoT, and supports asynchronous communication and decentralized authorization at the same time.

(3) Based on the open-source MQTT broker HiveMQ, the forward secure end-to-end encrypted message transmission system is implemented and tested. The experimental results show that FSEE not only is easy to implement on the existing commercial brokers, but also has comparable performance to recent prominent research while providing higher security. The proposed system can be used for real-time IoT applications.

The manuscript is organized as follows. Section 2 presents the related work. In Section 3, assumptions and the requirements for end-to-end encryption system are discussed. In Section 4, we detail some basic components of our scheme, including BKEM, identity-based implicit authenticated key agreement protocol, identity-based public encryption and message authentication code. The design of BA-GKE is presented in Section 5. Based on BA-GKE, Section 6 designs the forward secure end-to-end encryption system FSEE. Then, Section 7 implements the system and evaluates the performance of our scheme. Finally, we conclude this article in Section 8.

2. Related Work

Security of IoT systems has been studied extensively, such as physical security of IoT devices [29–31], secure authentication [32, 33], access control [34, 35], data integrity [36], and outsourcing of computations [37]. Realizing end-to-end security for pub/sub communication has only been studied by few approaches to date.

2.1. Schemes Based on Trusted Broker. Currently, most IoT systems use TLS to protect the packets between brokers and clients [3]. The message broker could get all the plaintext produced by the devices and users. This approach forces users to trust the broker for data processing. Due to the access control problem in MQTT [9], Yan Jia et al. propose MOUCON, in which the access rights to a message for every client are checked based on specific policies. However, full trust in the broker is also required, which does not solve the confidentiality concern caused by malicious brokers.

2.2. Schemes Based on Trusted Key Server. A transparent end-to-end encryption scheme for pub-/sub-based cyber-physical systems (CPSs) is provided by Markus et al. [13], in which a trusted key server is required to distribute the topic-specific key to all the authorized clients. This approach forces users to trust the key server, whose compromise can lead to the breach of millions of client accounts and permissions.

2.3. Schemes Based on Proxy Re-Encryption. PICADOR uses a lattice-based proxy re-encryption scheme to realize a pub/subsystem with end-to-end encryption [15]. However, based on the publisher's private key and each subscriber's public key, PICADOR requires a trusted authority to generate a re-encryption key for each subscriber. Compromise of the trusted authority may lead to the exposure of all the publishers' private keys. Furthermore, PICADOR does not satisfy forward secrecy. References [16, 17] are also proxy re-encryption-based schemes, which have the same problem as PICADOR.

2.4. Schemes Based on Secret Sharing. Sana belguith et al. [38] propose an efficient revocable secure pub/sub system based on the idea of secret sharing. A broker is divided into three parts: topic matching, routing, and message sending. The adversary cannot compromise all the brokers. Three brokers re-encrypt the publisher's message in turn and then send it to the subscriber. However, the scheme requires the development of a custom message broker, which is difficult to deploy and does not achieve forward secrecy too.

2.5. Schemes Based on Hardware. Segarra et al. restrict the broker to run only in a trusted execution environment (TEE) [18] to tackle the problem of malicious or compromised brokers, thus ensuring that the broker software is executed as intended. However, TEE is not available on every server, which is difficult to deploy. Additionally, several attacks against the current mainstream TEE were shown [39], rendering its benefit questionable.

2.6. Schemes Based on IBE or ABE. JEDI [11] uses WKD-IBE algorithm to implement end-to-end encryption between devices and users in IoT. Sieve [12]and Yu at el. Reference [40] use ABE to control which principals have access to encrypted data in the cloud. All of these schemes support asynchronous communication and decentralized authorization but do not provide forward secrecy as well.

2.7. Schemes Based on Chaos Cryptography. Some schemes [41–45] implement real-time image encryption for IoT using chaos-based cryptography. Essentially, chaos-based cryptography utilizes chaotic signals to generate pseudorandom sequences, which is more suitable for video and image encryption compared to traditional cryptography techniques (DES, IDEA, and AES) [46]. These efforts focus on solving the problems of large data capacity, strong adjacent pixels correlation, high real-time performance, and high redundancy among raw pixels, and they do not provide forward secrecy as well.

3. Assumption and Requirements for End-to-End Encryption System

3.1. Assumption. As is shown in Figure 1, the device owner is a special user of its devices, who can grant their device's access rights to others (such as guests, office members, and maintenance workers) and also revoke their authorization. We make the following assumptions:

(1) Each device corresponds to a unique device owner and is completely controlled by its owner. A device trusts its owner completely. If the device owner is compromised, the adversary can gain full control over its device.

(2) When a user initially purchases an IoT device from the manufacturer and binds it to the message broker, we call the user the owner of the device. The device owner operates his mobile APP and establishes a local connection with the device. We assume that the device and the device owner are mutually authenticated through the local connection, and some initial secret keys are reliably exchanged between the two.

(3) By distributing authorization tokens to other users through secure channels (such as the local connection or some out-of-band channels), the device owner can grant their device's access rights to others.

3.2. Requirements for End-to-End Encryption Scheme. Based on the consideration of security and availability, we summarize five requirements that end-to-end encryption methods should meet.

3.2.1. Confidentiality. Confidentiality ensures that the interpretation of a message is impossible for anyone except the target authorized users. As the authorized user set is dynamically changing (a user is authorized or revoked), there are two meanings of confidentiality here: first, newly joined users should not be able to interpret encrypted data before their joining time; second, revoked users that previously had a key should not be able to interpret future encrypted data using their previous revoked key.

3.2.2. Support Asynchronous Communication. In the Internet of Things, data generated by a device may be obtained by multiple authorized users. Generally, these users may not always be online. Users who are offline cannot affect the communication between the device and other users. Therefore, an IoT end-to-end encryption solution also needs to support asynchronous communication. Even if a user is offline at a certain time, it will not affect other users to get the updated symmetric session key. Once the user is online, the latest session key can be obtained.

3.2.3. Forward Secrecy. Forward secrecy means that compromise of long-term secrets does not lead to compromise of session keys of previously completed sessions. Namely, the confidentiality of previously encrypted messages can be guaranteed when an entity is compromised later.

3.2.4. Deployability. To ensure wide deployment of the end-to-end encryption system, the security scheme should be

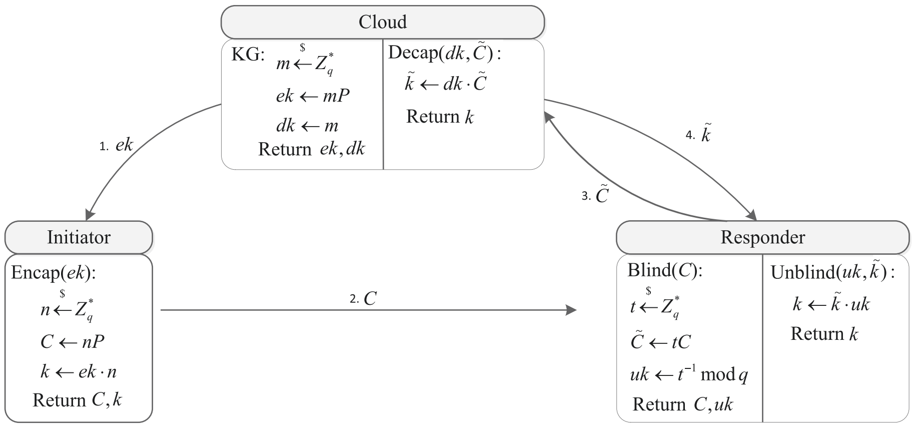

Cloud
KG: $m \overset{\$}{\leftarrow} Z_q^*$
$ek \leftarrow mP$
$dk \leftarrow m$
Return ek, dk
Decap$(dk, \tilde{C})$:
$\tilde{k} \leftarrow dk \cdot \tilde{C}$
Return k
1. ek
4. $\tilde{k}$
3. $\tilde{C}$
Initiator
Encap(ek):
$n \overset{\$}{\leftarrow} Z_q^*$
$C \leftarrow nP$
$k \leftarrow ek \cdot n$
Return C, k
2. C
Responder
Blind(C):
$t \overset{\$}{\leftarrow} Z_q^*$
$\tilde{C} \leftarrow tC$
$uk \leftarrow t^{-1} \bmod q$
Return C, uk
Unblind$(uk, \tilde{k})$:
$k \leftarrow \tilde{k} \cdot uk$
Return k

Figure 2: Diffie–Hellman-based blinded KEM in the case of one responder.

designated in a deployable manner; that is, the security scheme should make as few modifications as possible to the current IoT message broker, preferably easy to implement on the existing ones, to minimize the cost of implementing the security scheme. Otherwise, the redevelopment of a customized message broker will not only take too long but also is incompatible with existing systems and cannot be interconnected.

Furthermore, most IoT devices are powered by batteries and with limited resources. Therefore, operations of IoT devices should be as simple as possible.

3.2.5. Decentralized Authorization. Relying on a centralized authority to achieve fine-grained authorization of all devices has poor scalability, and this authority may become the target of adversaries. Therefore, the end-to-end encryption scheme should not rely on a centralized trusted third party, and the authorization of each device should be fully controlled by its owner. If an adversary compromises a device owner, it only threatens the safety of the owner and its devices, and the safety of other users and devices will not be affected.

4. Preliminaries

4.1. Blinded Key Encapsulation Mechanism (BKEM). The concept of key encapsulation mechanism (KEM) is to use public key encryption algorithm to transport keys for use in symmetric encryption and does not provide any forward secrecy. Blinded key encapsulation mechanism (BKEM) [27] is proposed in the scenario with cloud participation. The semi-trusted cloud is used to provide forward secrecy for communication between entities. BKEM consists of three entities: a cloud server, an initiator, and multiple responders. A BKEM has two additional algorithms: blinding algorithm and unblinding algorithm.

A blinded KEM (BKEM) consists of five algorithms (**KG**, **Encap**, **Blin d**, **De cap**, **Unblin d**). Figure 2 is a DH based BKEM. Let $\mathbb{G}$ be a cyclic group of prime order q with generator P.

KG $(\lambda) \longrightarrow (ek, dk)$: The key generation algorithm is executed by the cloud, and on input security parameter, output an encapsulation key ek and a decapsulation key dk.

Encap $(ek) \longrightarrow (C, k)$: The encapsulation algorithm is executed by the initiator, takes as input an encapsulation key ek, and outputs an encapsulation C and a group session key k. The initiator will transfer the encapsulation C to whoever he wants to establish the shared session key. Note that encapsulation C must be transmitted to the intended responders secretly; otherwise, anyone who knows the encapsulation C can interact with the cloud to get the final session key k.

Blin d $(t, C) \longrightarrow (\tilde{C}, uk)$: The blinding algorithm is executed by multiple responders. Each responder chooses a random blind value t, inputs the encapsulation C from the initiator, outputs a blinded encapsulation $\tilde{C}$ and an unblinding value uk. uk is closely related to the selected random blind value t. Then, responders send the blinded encapsulation $\tilde{C}$ to the cloud.

De cap $(dk\ , \tilde{C}) \longrightarrow \tilde{k}$: The decapsulation algorithm is executed by the cloud. It takes decapsulation key dk and a blinded encapsulation $\tilde{C}$ as input, outputs a blinded key $\tilde{k}$ to the responders.

Unblin d $(uk, \tilde{k}) \longrightarrow k$: The unblinding algorithm is executed by the responders. It takes as input an unblinding value uk and a blinded key $\tilde{k}$ from the cloud and outputs the final session key k.

In BKEM, the cloud is semi-trusted. The initiator uses the cloud to negotiate the same group session key with multiple responders noninteractively, and multiple responders do not need to be online at the same time, thus supporting asynchronous communication. BKEM is essentially a DH key exchange between the initiator and the cloud. The encapsulation C generated by the initiator is actually its DH public key. The initiator secretly transmits the encapsulation C to the responders, and the cloud cannot obtain the encapsulation C, which ensures that only the initiator and the responders can negotiate the same session key, whereas the cloud cannot get any useful information of the session key. In addition, based on the property of DH key exchange, compromise of user's long-term secrets does not lead to compromise of session keys of previously completed sessions. So forward secure communication between the initiator and the responders is realized.

4.2. Identity-Based Authenticated Key Exchange (ID-AKE). Sherman et al. proposed an identity-based key exchange protocol (ID-AKE) [28]. While completing the key exchange, the identity of both sides can be implicitly authenticated. The scheme includes the following three stages: system setup, key extraction, and interaction.

AKE.Setup $(\lambda_{\mathrm{AKE}}) \longrightarrow (\mathrm{msk}_{\mathrm{AKE}}, \mathrm{params}_{\mathrm{AKE}})$: On input a security parameter λ_{AKE}, The Key Generation Center $\mathrm{KGC}_{\mathrm{AKE}}$ generates $(\mathbb{G}, \mathbb{G}_T, e)$ where $\mathbb{G}$ and $\mathbb{G}_T$ are groups of prime order q and e: $\mathbb{G} \times \mathbb{G} \longrightarrow \mathbb{G}_T$ is the pairing function. KGC chooses three cryptographic hash functions $\mathscr{H}$: $\{0,1\}^n \longrightarrow \mathbb{G}$, $\mathscr{H}_0$: $\mathbb{G} \times \{0,1\}^* \longrightarrow \mathbb{Z}_q^*$ and a key derivation function $\mathscr{K}$. All three of these are modeled as random oracles. KGC also randomly chooses an arbitrary generator P of $\mathbb{G}$. An element s is randomly chosen from $\mathbb{Z}_q^*$ as the master secret key $\mathrm{msk}_{\mathrm{AKE}}$, and the corresponding public key is $P_{\mathrm{pub}} = sP$. Finally, KGC keeps $\mathrm{msk}_{\mathrm{AKE}}$ secret and makes public parameters $\mathrm{params}_{\mathrm{AKE}} = \langle \mathbb{G}, \mathbb{G}_T, q, e, P, P_{\mathrm{pub}}, \mathscr{H}, \mathscr{H}_0, \mathscr{K} \rangle$ public.

AKE.Extract $(\mathrm{msk}_{\mathrm{AKE}}, \mathrm{ID}_i) \longrightarrow S_{ID_i}$: On input an identity ID_i and master secret key $\mathrm{msk}_{\mathrm{AKE}}$, the public key Q_{ID_i} of ID_i is set as $\mathscr{H}\,(ID_i)$. KGC uses the master secret key s to generate user's private key $S_{ID_i} = s\mathscr{H}\,(ID_i)$.

AKE.Interact $(S_A, ID_A, S_B, ID_B) \longrightarrow \mathrm{sk}$: If user A and user B want to authenticate each other and finally exchange a session key, they first interact with each other to exchange some information and then use their private key, the received information and the identity of the other party to authenticate each other, and negotiate the same session key. The interaction process is shown in Figure 3. User A randomly chooses a from $\mathbb{Z}_q^*$, computes $(W_A, T_A) = (aQ_A, aP)$, and sends (ID_A, W_A, T_A) to B. User B randomly chooses b from $\mathbb{Z}_q^*$, computes $(W_B, T_B) = (bQ_A, bP)$, and sends (ID_B, W_B, T_B) to A. Both user A and user B calculate $h_A = \mathscr{H}_0\,(W_A, T_A, ID_B)$ and $h_B = \mathscr{H}_0\,(W_B, T_B, ID_A)$. Finally, user A calculates the final session key as $sk_A = \mathscr{K}\,(e\,((a + h_A)S_A, W_B + h_B Q_B), aT_B)$, and user B calculates the final session key as $sk_B = \mathscr{K}\,(e\,(W_A + h_A Q_A, (b + h_B)S_B), bT_A)$.

This scheme can provide session key indistinguishability and KGC forward secrecy, and also can be proved to be secure in the CK (Canetti–Krawczyk) model [47]. The session key indistinguishability means that the real session key is indistinguishable from a random value for the adversary. KGC forward secrecy means that although the adversary can not only get the private key of both sides but also get the master secret key of KGC, he still cannot recover the session key previously negotiated by both sides.

4.3. Identity-Based Public Key Encryption (IBE). An identity-based encryption scheme consists a tuple of algorithms (**IBE.Setup**, **IBE.Extract**, **IBE.Enc**, **IBE.De c**).

IBE.Setup $(\lambda_{\mathrm{IBE}}) \longrightarrow (\mathrm{msk}_{\mathrm{IBE}}, \mathrm{params}_{\mathrm{IBE}})$: input security parameter λ_{IBE}, output a master secret key $\mathrm{msk}_{\mathrm{IBE}}$ and public parameters $\mathrm{params}_{\mathrm{IBE}}$. $\mathrm{KGC}_{\mathrm{IBE}}$ keeps $\mathrm{msk}_{\mathrm{IBE}}$ secret, and makes the $params_{IBE}$ public.

IBE.Extract $(\mathrm{msk}_{\mathrm{IBE}}, \mathrm{ID}_i) \longrightarrow sk_{ID_i}$: input master secret key $\mathrm{msk}_{\mathrm{IBE}}$ and a user's identity ID_i, and output the user's private key sk_{ID_i}.

IBE.Enc $(ID_i, m) \longrightarrow CT_i$: input receiver's identity ID_i and a message m, and output the ciphertext CT_i.

IBE.De c $(sk_{ID_i}, CT_i) \longrightarrow m/\perp$: input receiver's private key sk_{ID_i} and the ciphertext CT, and output the plaintext m or a special reject symbol $\perp$.

4.4. Message Authentication Code (MAC). A MAC scheme is a pair of efficient algorithms $(\mathbf{S}, \mathbf{V})$, where S is called a signing algorithm and V is called a verification algorithm. Algorithm S is used to generate tags and V is used to verify tags.

$\mathbf{S}\,(k, m) \longrightarrow t$: k is a key and m is a message, and output t as the tag of message m.

$\mathbf{V}\,(k, m, t) \longrightarrow r$: k is a key, m is a message, and t is a tag, and output r as either *accept* or *reject*.

5. BA-GKE Group Key Exchange Protocol

5.1. Protocol Design. BKEM can only realize forward secure and asynchronous key exchange between the initiator and responders in authenticated-links model [47], and the users and cloud server also need to authenticate each other when using BKEM to construct a secure group key exchange protocol in unauthenticated-links model [47].

Here, combining ID-AKE with DH-based BKEM, a novel group key exchange protocol BA-GKE is proposed. Our scenario consists of the following participants:

(1) The initiator, also the group manager, wants to establish a shared session key with a set of responders.
(2) Multiple responders want to allow the initiator to establish a shared key with them.
(3) The server temporarily stores information assisting the responders to compute the shared session key.

The BA-GKE group key exchange protocol is defined in Figure 4 in the case of one responder and is parameterized by the following components.

(1) **BKEM** = (**KG**, **Encap**, **Blin d**, **De cap**, **Unblin d**) be a DH based BKEM,
(2) **I D − AKE** = (**Setup**, **Extract**, **Interact**) be an identity-based implicit authenticated key exchange scheme,
(3) **IBE** = (**Setup**, **Extract**, **Enc**, **De c**) be an identity-based public key encryption scheme,
(4) $\mathbf{H}_1$ be a secure hash functions,
(5) **MAC** = $(\mathbf{S}, \mathbf{V})$ be a message authentication code scheme (MAC).

The process of the protocol includes two phases: In phase 1, based on the ID-AKE protocol of Sherman et al. [28], the initiator and the server authenticate each other, and a session key sk is negotiated, which is used by the server and multiple

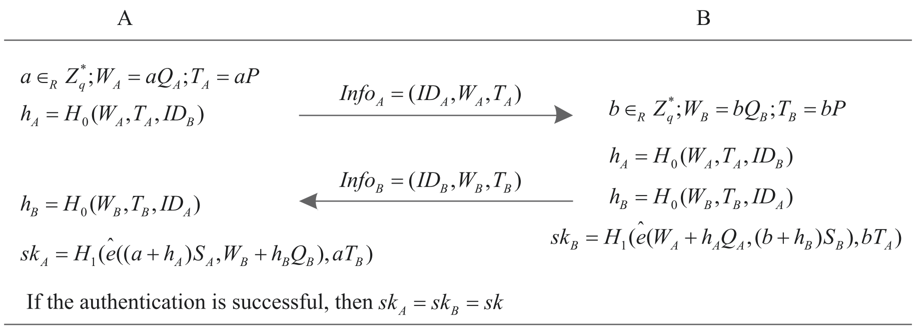

A
B
$a \in_R Z_q^*; W_A = aQ_A; T_A = aP$
$h_A = H_0(W_A, T_A, ID_B)$
$Info_A = (ID_A, W_A, T_A)$
$b \in_R Z_q^*; W_B = bQ_B; T_B = bP$
$h_A = H_0(W_A, T_A, ID_B)$
$h_B = H_0(W_B, T_B, ID_A)$
$Info_B = (ID_B, W_B, T_B)$
$h_B = H_0(W_B, T_B, ID_A)$
$sk_B = H_1(\hat{e}(W_A + h_A Q_A, (b + h_B)S_B), bT_A)$
$sk_A = H_1(\hat{e}((a + h_A)S_A, W_B + h_B Q_B), aT_B)$
If the authentication is successful, then $sk_A = sk_B = sk$

Figure 3: The interaction process of both parties of ID-AKE.

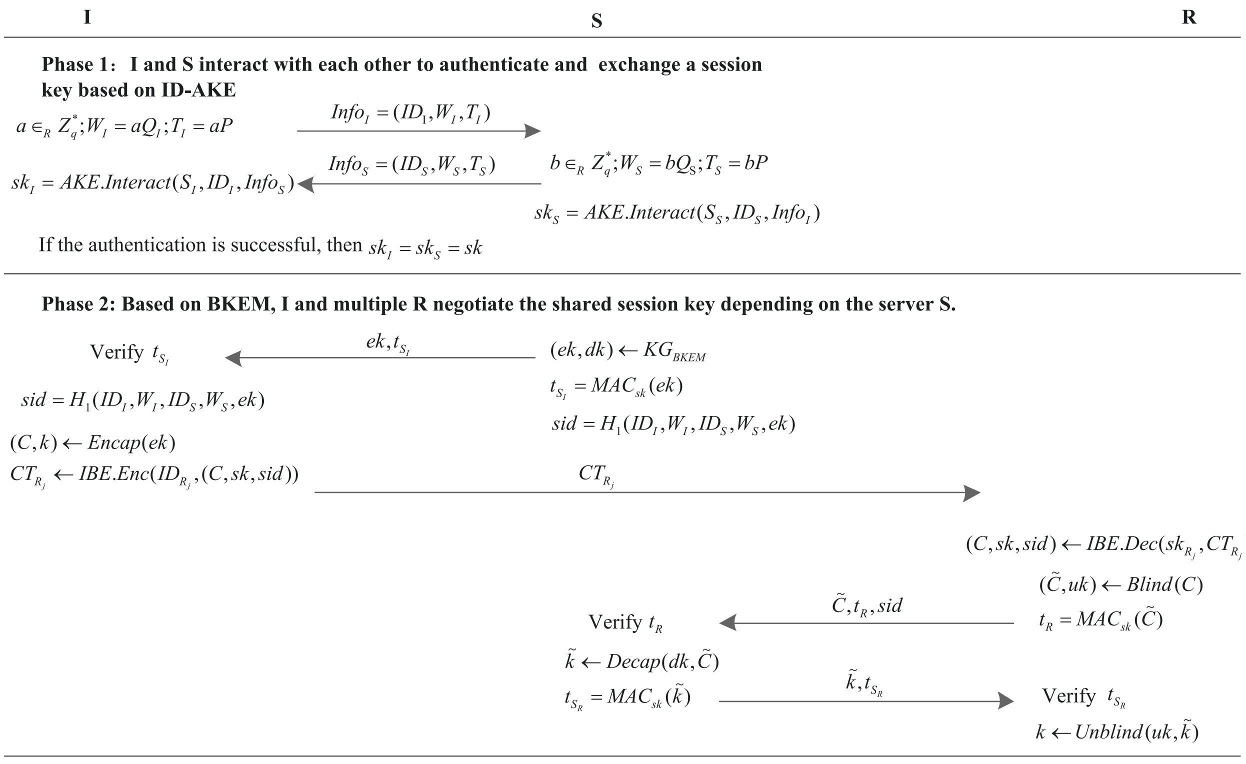

I
S
R
Phase 1: I and S interact with each other to authenticate and exchange a session key based on ID-AKE
$a \in_R Z_q^*; W_I = aQ_I; T_I = aP$
$Info_I = (ID_I, W_I, T_I)$
$Info_S = (ID_S, W_S, T_S)$
$b \in_R Z_q^*; W_S = bQ_S; T_S = bP$
$sk_I = AKE.Interact(S_I, ID_I, Info_S)$
$sk_S = AKE.Interact(S_S, ID_S, Info_I)$
If the authentication is successful, then $sk_I = sk_S = sk$
Phase 2: Based on BKEM, I and multiple R negotiate the shared session key depending on the server S.
Verify t_{S_I}
ek, t_{S_I}
$(ek, dk) \leftarrow KG_{BKEM}$
$t_{S_I} = MAC_{sk}(ek)$
$sid = H_1(ID_I, W_I, ID_S, W_S, ek)$
$sid = H_1(ID_I, W_I, ID_S, W_S, ek)$
$(C, k) \leftarrow Encap(ek)$
$CT_{R_j} \leftarrow IBE.Enc(ID_{R_j}, (C, sk, sid))$
CT_{R_j}
$(C, sk, sid) \leftarrow IBE.Dec(sk_{R_j}, CT_{R_j})$
$(\tilde{C}, uk) \leftarrow Blind(C)$
Verify t_R
$\tilde{C}, t_R, sid$
$t_R = MAC_{sk}(\tilde{C})$
$\tilde{k} \leftarrow Decap(dk, \tilde{C})$
$t_{S_R} = MAC_{sk}(\tilde{k})$
$\tilde{k}, t_{S_R}$
Verify t_{S_R}
$k \leftarrow Unblind(uk, \tilde{k})$

Figure 4: Message sequence chart of the BA-GKE group key exchange protocol in the case of one responder.

responders to authenticate each other in the next phase; in phase 2, based on BKEM, the initiator negotiates a common session key k with multiple responders with the help of server. In this phase, the identity-based encryption scheme (IBE) is used to transfer sk and encapsulation C to each responder. Specifically, BA-GKE works in the following steps.

5.1.1. System Initialization. On input the security parameter λ_{AKE}, the KGC of ID-AKE KGC_{AKE} output msk_{AKE} and public parameters $\text{params}_{\text{AKE}}$.

5.1.2. Server and Initiator Registration. On input server's identity ID_S and initiator's identity ID_I, KGC_{AKE} runs algorithm **AKE.Extract**, uses msk_{AKE} to generate server's private key S_S and initiator's private key S_I, and sends the corresponding private key to them securely.

5.1.3. Responder Registration. If initiator ID_I wants to negotiate a common session key with multiple responders, the initiator acts as the KGC of an IBE scheme. On input security parameter λ_{IBE}, the initiator outputs a master secret key msk_{IBE} and public parameters $\text{params}_{\text{IBE}}$. Then, the initiator runs algorithm **IBE.Extract**, uses msk_{IBE} and each responder's identity ID_R to generate its private key sk_R, and sends the private key to the corresponding responder through secure channel.

5.1.4. Initiator and Server Authentication. Initiator and server run algorithm **AKE.Interact** to exchange information with each other, and finally output the same session key sk, which is used as the MAC key in the following steps.

5.1.5. Encapsulation Key Generation. The server runs algorithm **KG** of BKEM generates an encapsulation key ek and a decapsulation key dk, computes the session identifier $\text{sid} = H_1(ID_I, W_I, ID_S, W_S, ek)$, and uses session key sk as the MAC key to generate ek's tag t_{S_I}. Finally, the server sends (ek, t_{S_I}) to the initiator.

5.1.6. Encapsulation Generation. Upon receiving the messages (ek, t_{S_I}) from the server, the initiator first uses sk to verify t_{S_I}. If it is not a valid tag, the initiator outputs $\perp$; otherwise, it runs **Encap** to get the group session key k and encapsulation C and also computes the session identifier $\text{sid} = H_1(ID_I, W_I, ID_S, W_S, ek)$. For each responder R_j, initiator runs algorithm **IBE**.**Enc**, takes each responder's identity ID_{R_j}, encapsulation C, sk, and session identifier $si\ d$ as input, and outputs the ciphertext CT_{R_j} to the corresponding responder R_j.

5.1.7. Blinded Encapsulation Generation. On receiving the ciphertext CT_{R_j}, each responder R_j runs algorithm **IBE**.**De c**, takes its private key sk_{R_j} and CT_{R_j} as input, and gets the plaintext (C, sk, sid). Then, the responder runs algorithm **Blin d** to generate the blinded encapsulation $\widetilde{C}$ and an unblinding value uk, uses sk as the MAC key to generate $\widetilde{C}$'s tag t_R, and finally sends $(\widetilde{C}, t_R, \text{sid})$ to the server.

5.1.8. Decapsulating for Responders. Upon receiving the messages $(\widetilde{C}, t_R, \text{sid})$ from the responder, the server first uses sk to verify the tag t_R. If it is not a valid tag, the server outputs $\perp$; otherwise, it runs algorithm **De cap**, takes its decapsulation key dk and responder's blinded encapsulation $\widetilde{C}$ as input, and outputs a blinded key $\widetilde{k}$. Finally, the server uses MAC key sk to compute $\widetilde{k}$'s tag t_{S_R} and sends $(\widetilde{k}, t_{S_R})$ back to the responder.

Unblinding for blinded key on receiving the messages $(\widetilde{k}, t_{S_R})$ from the server, the responder uses sk to verify the tag t_{S_R}. If it is not a valid tag, the responder outputs $\perp$; otherwise, it runs algorithm **Unblin d**, uses its unblinding value uk and $\widetilde{k}$ as input, and gets the final group session key k.

5.2. Formal Security Validation with ProVerif. This section presents a formal verification of the proposed protocol using ProVerif [48]. Our verification ensures that BA-GKE provides the secrecy of generated group session key, the authentication between group members, and forward secrecy. ProVerif is an automatic verifier for cryptographic protocols defined in the Dolev–Yao model [49]. In this model, the adversary is an active eavesdropper, who can add, delete, modify, and delay messages on the network. In ProVerif, the cryptographic primitives are considered idealized that they are unbreakable without knowing the employed secret keys.

A protocol description in ProVerif is divided into three parts: the declarations, the process macros, and the main process. The ProVerif description of our scheme is given in the appendix. As described in lines 1–43, the declaration part consists of the used types, the security properties, the cryptographic primitive functions, and the list of defined events and queries. The queries defined in lines 29–31 are used to verify the confidentiality of messages transmitted between the initiator and responders. The initiator and responders select a message and use the new group session key to send the message, and test whether the adversary can get the encrypted data through the queries in lines 30 and 31. Lines 38–43 define a series of events to verify mutual authentication between initiator and responders.

The second part of the ProVerif program describes the process macros for participants: initiator (Lines 44–72), key exchange server (Lines 73–99), and responders (Lines 100–122). When defining the process macro of initiator and responders, the events defined before are inserted to track the authentication results between the two parties.

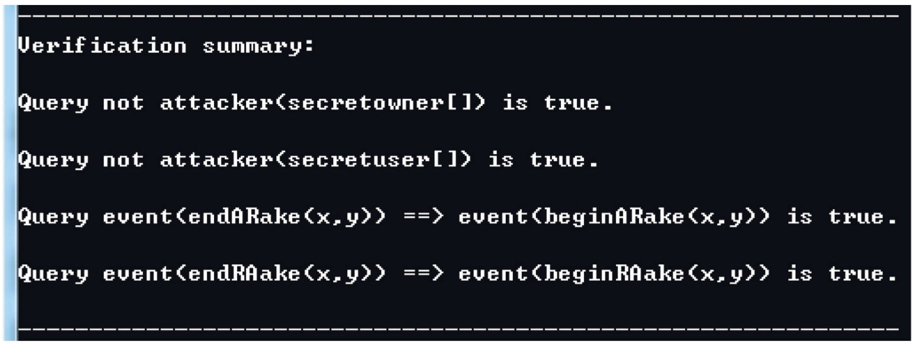
In the last part, the main process of the protocol is defined in lines 123–134. In order to prove that the protocol satisfies forward secrecy, two phases are defined in the main process. In phase 0, instantiate the corresponding key material, insert the key into the correct table, and run the corresponding process macros infinitely. In phase 1, the private key of the users is output to the adversary. Finally, the running result of ProVerif is shown in Figure 5.

Lines 1-2 show the results of the queries $\text{not attacker (secretowner)}$ and $n\text{ot attacker (secretuser)}$ returned by ProVerif. As we shall see, these results are true, which means that the secrecy of the random values secretowner and secretuser are preserved by the protocol. In other words, the secrecy of the group session key generated by our scheme is preserved. In addition, we use phases to prove forward secrecy properties. In phase 1, even if the participants get corrupted (their private key is leaked to the adversary), the secrets exchanged in phase 0 are preserved, so our protocol achieves forward secrecy. Furthermore, lines 3-4 inform us that the proposed scheme provides mutual authentication of the initiator and responders. As such, the proved correspondence property in line 3 implies that the initiator authenticates responders by the fact that responders can correctly retrieve the group session key. Similarly, line 4 indicates that the responders can authenticate the initiator.

6. FSEE: The End-to-End Encryption System for IoT

6.1. System Framework. In this section, we develop FSEE, which allows forward secure end-to-end encrypted message transmission in pub/sub communication in IoT. The core idea of FSEE is using BA-GKE to update the session key per device between its authorized users, and the security of previously negotiated session key is guaranteed even when users' long-term secrets are compromised.

As BA-GKE relies on a semi-trusted key exchange server for noninteractive key exchange and forward secrecy, FSEE adds a semi-trusted key exchange server to the existing IoT architecture. It includes four types of entities: the IoT devices, the message broker, multiple authorized users, and the key exchange server. These entities constitute two networks, and the specific structure is shown in Figure 6. In the actual deployment, the message broker and key exchange server can be deployed together or separately.



```
------------------------------------------------------------
Verification summary:

Query not attacker(secretowner[]) is true.

Query not attacker(secretuser[]) is true.

Query event(endARake(x,y)) ==> event(beginARake(x,y)) is true.

Query event(endRAake(x,y)) ==> event(beginRAake(x,y)) is true.

------------------------------------------------------------
```

Figure 5: Verification results of the BA-GKE group key exchange protocol.

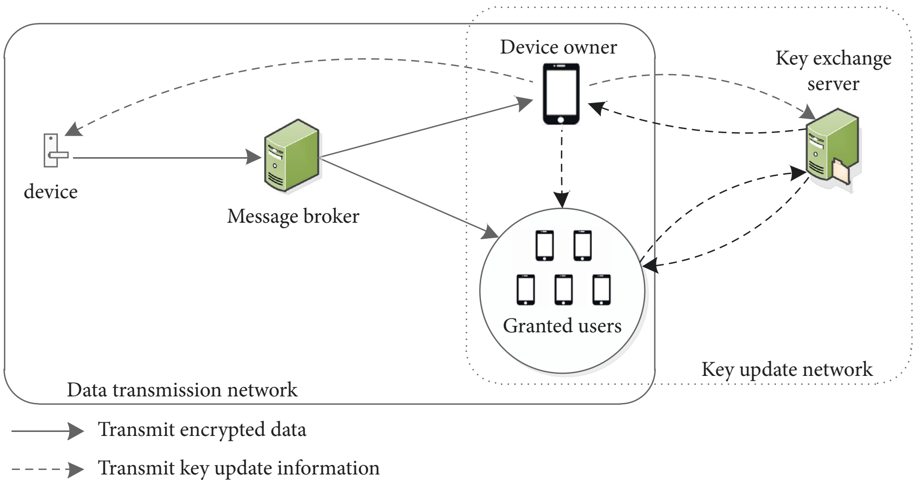


Figure 6: Network architecture of FSEE.

On one hand, the device, message broker, and multiple authorized users constitute a data transmission network. All kinds of information collected by the device will be sent to users' applications through the message broker.

On the other hand, multiple authorized users and the key exchange server form a key update network. It is responsible for updating the group session key between the authorized users and the device when the user set changes. When a user's access rights are revoked by the owner, BA-GKE group key exchange protocol proposed in the previous section is adopted, and the group session key between authorized users is updated through the key exchange server. Since BA-GKE is relatively time-consuming, hash chain could be used to update the group session key when the device owner authorizes a new user to access its device.

6.2. Steps of FSEE. Specifically, FSEE includes the following seven steps.

6.2.1. System Establishment Stage. The device manufacturer is the key generation center of ID-AKE and runs the **AKE.Setup** algorithm, enters the security parameters, and returns public parameter $params_{AKE}$ and master secret key $\mathrm{msk}_{\mathrm{AKE}}$. The device manufacturer entrusts the key exchange service of all its devices to a third-party key exchange server and runs **AKE.Extract** to generate the private key S_S for the third-party key exchange server.

6.2.2. Device Owner Registration. When a user purchases an IoT device, the user is the owner of the device. The manufacturer runs **AKE.Extract** to generate and send the private key S_I to the owner through a secure channel (SMS or sent with the device).

At the same time, the device owner, as the key generation center of IBE, chooses the corresponding security parameters and returns the public parameter $\mathrm{params}_{\mathrm{IBE}}$ and master secret key $\mathrm{msk}_{\mathrm{IBE}}$.

6.2.3. Device Registration. Generally, a newly purchased IoT device begins its life cycle through "device discovery." The device owner operates his mobile APP, and the APP establishes a local connection with the device [1]. After the "device discovery" stage, the device and the device owner complete the mutual authentication and both sides share an initial key *ik* and a key encryption key kek through the local connection. The shared initial key *ik* is the first group session key shared between the device and its owner.

6.2.4. Authorization Stage. When the device owner wants to grant the device access rights to others, the device owner first uses hash chain to generate a new session key k' from the current one k using a hash function, and the generated new session key is sent to newly joined users. At the same time, a key update command is broadcasted, so that the device and other authorized users can also update their shared session key through the same hash function. This is to maintain the consistency of the shared session key between the group members. From then on, the device and all its authorized users use the new session key to encrypt and decrypt transmitted messages.

In addition, for each newly joined users ID_j, the device owner uses its IBE master secret key $\mathrm{msk}_{\mathrm{IBE}}$ to generate new user's IBE private key sk_{ID_j} and sends it to the new user through secure channel. The new user keeps their private key sk_{ID_j} secret.

6.2.5. Revocation Stage. When the device owner wants to revoke the access rights of a user, as the initiator of BA-GKE, the device owner starts a round of BA-GKE to update the group session key with the remaining authorized users. Specifically, it includes two phases: in the first phase, based on ID-AKE, the device owner and the key exchange server authenticate each other; in the second phase, based on BKEM, the device owner negotiates a new shared session key with the remaining authorized users with the help of the key server. Identities of the remaining authorized users are used to encrypt the new encapsulation C, whereas identities of revoked users are not used. The encrypted encapsulation C is sent to the remaining authorized users, which can use their private key to get the encapsulation C, and then interact with the key exchange server to get the updated group session key, whereas the device owner does not encrypt the new encapsulation C for revoked users, who cannot decrypt the ciphertext to get the encapsulation and thus the updated group session key cannot be obtained.

6.2.6. Update the Key of IoT Devices. As IoT devices are generally resource-constrained, the calculation of the device should be as simple as possible. In our scenario, when a new user joins, the device updates the session key through the hash chain just like other users. However, the device does not participate in the revocation process of the above BA-GKE protocol between group members, and the updated session key during the revocation stage is distributed directly by the device owner.

When a group member is revoked, the device owner encrypts the updated session key directly with the key encryption key kek shared with the device and then transfers it to the device. The device uses the key encryption key to get the updated session key. When the device successfully receives the new session key, the device and the device owner also use the shared key encryption key as the input of a hash function, and both sides update the key encryption key kek synchronously to a new one kek'. Key encryption key update is to ensure the forward secrecy of the communication between the device and the device owner.

6.2.7. Update Session Key Using BA-GKE Periodically. When the authorized user set remains unchanged for a long time, the device-specific session key will remain unchanged, which reduces the security of our system. Therefore, we use BA-GKE to update the device-specific session key periodically at the end of each hour (or other intervals). FSEE performs key rotation using BA-GKE only at the end of each hour or when a user is revoked, and cheap symmetric key encryption and hash operation is incurred for the rest of the time.

6.3. System Analysis. In this section, we provide a comprehensive analysis of the proposed FSEE and show that our scheme has achieved all the requirements mentioned in Section 2.

6.3.1. Confidentiality. When new users join the group, the group session key is updated using hash chain. Based on the one-way property of hash function, the session key obtained by the new user cannot be used to derive the former session key.

When a user is revoked, the device owner, as the initiator of BA-GKE, relies on the key exchange server to renegotiate the new group session key. The new session key is independent of the old one. The revoked user cannot use the former session key to derive the new group session key updated by BA-GKE after revocation.

End-to-end encryption ensures that no server processing the messages or any third-party adversaries can read the message sent between the source and destination, so the underlying broker cannot get any valid information.

6.3.2. Forward Secrecy. When the adversary gets the long-term IBE private key of the authorized users, he can get all the encapsulation C generated by the owner every time the system uses BA-GKE to update the group session key, because the ephemeral encapsulation key ek and decapsulation key dk of the key exchange server change once every time the session key is updated. Even if the adversary gets the previously generated encapsulation C, it cannot interact with the server to recover the previously negotiated group session key. When the adversary gets the long-term ID-AKE private key of the key server, he could not get the encapsulation C, and knowing encapsulation key ek and decapsulation key dk could not help him to get the session key too.

6.3.3. Asynchronous Communication. When new users are authorized, the group session key is updated by hash chain, and the offline group members will not affect the key update process. When a user is revoked, as long as the device owner and the key exchange server are online, the online group members can update to the latest group session key, and the key updating process will not be affected by the offline ones. In addition, the device owner can send the encrypted encapsulation CT to the key exchange server for temporary storage. Once the offline users are online, they can get the latest encrypted encapsulation CT from the key exchange server. If the member is not revoked, it can decrypt with its private key to get the latest encapsulation C and then use it to interact with the server to get the latest group session key. In a word, our end-to-end encryption scheme supports asynchronous communication.

6.3.4. Deployability. In our scheme, the message broker is only responsible for message routing and forwarding and does not participate in the group key exchange process. Our scheme does not need to change the existing IoT message broker. When IoT device manufacturers develop new devices, they can deploy our solution in the device and user application, and build their key exchange server for all their customers based on existing cloud service providers. Therefore, our scheme has lower deployment costs and supports incremental deployment.

6.3.5. Decentralized Authorization. FSEE allows each device owner to act as an authority of its devices in its own trust domain, without a single trusted third-party managing the global authorization information.

7. Implementation and Performance Evaluation

We evaluate FSEE in this section. In what follows, we first introduce our prototype implementation and then measure FSEE's performance.

7.1. Prototype Implementation. The prototype of FSEE contains multiple modules, including devices, user applications, a message broker, and a key exchange server. The development of the device and user applications is based on Eclipse Paho Java Client library [50]. The construction of message broker is based on an open-source MQTT server HiveMQ [51]. HiveMQ provides a flexible extension framework that allows developers to create custom extensions, and the key exchange server is implemented as a HiveMQ extension. Cryptographic operations are supported by Java Pairing-Based Cryptography Library [52]. The prototype system runs on a laptop computer configured with Intel Core i7-5600U 2.6 GHZ CPU and 8G RAM. The computer's operating system is Windows7. The development environment of the device, user application, and key exchange server is IntelliJ IDEA 2018.1.6.

7.1.1. BA-GKE Implementation. The BA-GKE group key exchange protocol used in FSEE is constructed based on DH-based BKEM, ID-AKE [28] and IBE. The IBE scheme chooses the identity-based encryption scheme proposed by Boneh-Franklin et al. [53]. The ID-AKE [28]and IBE [53] schemes are constructed based on a symmetric bilinear group. Therefore, to achieve the security level of 80 bits, FSEE uses the configuration file "a.properties" provided by the JPBC library to generate a type "A" symmetric bilinear group based on a 160 bits prime order elliptic curve $y^2 = x^3 + x \bmod p (p \equiv 3 \bmod 4)$ with embedding degree 2.

7.1.2. Client Implementation. Functions of device and user application are simulated via Java console programs that rely on the Eclipse Paho Java Client library. It is relatively easy to implement the IoT device, as there are no complex cryptographic operations on the device, and only symmetric cryptographic algorithms are performed. The user application is also based on Eclipse Paho Java Client library, and two packages of JPBC library are added to support calculations related to bilinear pairing.

7.1.3. Message Broker and Key Exchange Server Implementation. In the prototype system, the message broker and key exchange server are deployed together on the same computer. The message broker is built based on HiveMQ Community Edition [54] to realize message routing and forwarding. HiveMQ is a world-class, enterprise-ready MQTT broker that provides fast, efficient, and reliable movement of data to and from connected IoT devices.

The key exchange server is realized as an extension of HiveMQ based on HiveMQ Community Extension SDK [55]. HiveMQ Community Extension SDK contains multiple HiveMQ interceptors, which provide a convenient way for extensions to intercept and modify MQTT messages. The key server uses Publish Inbound Interceptor to provide key exchange services for each authorized user.

7.1.4. Topic Design of FSEE. FSEE utilizes the key exchange server to provide a symmetric session key per device, which is used by the device and its authorized users to communicate with each other securely. To build a foundation to transmit key materials to and from the key exchange server, FSEE introduces a specific key exchange topic that is used for all communication between the key exchange server and users to establish a secure group session key based on existing pub/sub MQTT protocol. Each client has its subtopic based on its clientID and role (owner or not), and the topics to be handled by the device, device owner, and other authorized users are shown in Table 1.

In FSEE, topics are divided into three categories: (1) topics for data transmission (prefixed with *F*see/data/); (2) topics for key exchange (prefixed with Fsee/keyex/); and (3) topics for key update (prefixed with Fsee/keyup/).

The key exchange topics are divided into two subcategories according to whether the user is the device owner or not: a topic prefixed with Fsee/keyex/I/ is used by the device owner to interact with the server for authentication (prefixed with Fsee/keyex/I/toKS/), to obtain the encapsulated key from the server (prefixed with Fsee/keyex/I/toC/), and to send encrypted encapsulation to other authorized users (prefixed with Fsee/keyex/I/toR/). Another type of topic prefixed with Fsee/keyex/R/ is used by other authorized users to interact with the server to obtain the final symmetric session key.

Each device publishes data to the topic Fsee/data/deviceID, and all authorized users subscribe to the topic, thereby realizing data transmission in the system. When the device owner authorizes a new user to access the device, the device owner issues a key update command to

Table 1: Topic to be handled by each module.

Stages	Mode	Device	Device owner	Other authorized user
Data transmission	Pub	Fsee/data/deviceID	—	—
	Sub	—	Fsee/data/deviceID	Fsee/data/deviceID
BA-GKE key exchange	Pub	—	Fsee/keyex/I/toKS/clientID	Fsee/keyex/R/toKS/clientID
		—	Fsee/keyex/I/toR/authorized user's clientID	—
	Sub	—	Fsee/keyex/I/toC/clientID	Fsee/keyex/I/toR/clientID
—		—	Fsee/keyex/R/toC/clientID	
New uers joins	Pub	—	Fsee/keyup/deviceID	—
	Sub	Fsee/keyup/deviceID	—	Fsee/keyup/deviceID
Device's key update	Pub	—	Fsee/keyup/toD/deviceID	—
	Sub	Fsee/keyup/toD/deviceID	—	—

— represents that there are no topics to deal with.

topic Fsee/keyup/deviceID. The device and other authorized users subscribe to the topic to ensure that once the key update command is received, they use the hash function to update the device-specific session key.

The key exchange server intercepts packets with topic prefixed with Fsee/keyex/I/toKS/ and Fsee/keyex/R/toKS/ in the Publish Inbound Interceptor. If the topic of the packet is Fsee/keyex/I/toKS/clientID, it means that this is a message sent by the device owner to the key exchange server. The server obtains the authentication message from the packet payload, and the server's authentication message and encapsulation key are returned to the owner through topic Fsee/keyex/I/toC/clientID. If the topic is Fsee/keyex/R/toKS/clientID, it means that other authorized users are requesting the server to decapsulate its key. The server obtains the blinded encapsulation from the packet payload, and the blinded key is returned to the user through the topic Fsee/keyex/R/toC/clientID.

When a user's access rights are revoked, the device owner uses BA-GKE to update the session key, and the device owner publishes the updated new session key to the topic Fsee/keyup/toD/deviceID. The device subscribes to the topic to get the latest session key.

7.2. Evaluation. In this section, we evaluate the performance of FSEE based on the prototype system.

7.2.1. Overhead of Updating the Symmetric Key Using BA-GKE. Table 2 shows the performance of our type A curve implemented on JPBC. Preprocessing feature could be used to save time in the long run when a particular element is exponentiated and paired several times. The most time-consuming operation is exponentiation in $\mathbb{G}_1$ and pairing.

Figure 7 gives the computational overhead of each module (including device, device owner, key server, and other authorized users) in the system when using BA-GKE to update a symmetric key. The key server provides decapsulation services for each authorized user, so its computational cost increases linearly with the number of users, which increases by 15 ms for each additional authorized user. The device owner first interacts with the key exchange server based on ID-AKE and then encrypts the

Table 2: Execution time of different cryptographic operations.

Operation	Time (ms)
Pairing	12.4
Pairing with preprocessing	6.4
$\mathbb{G}_1$ exponentiation	15.3
$\mathbb{G}_1$ exponentiation With preprocessing	2.2
$\mathbb{G}_T$ exponentiation	1.4
$\mathbb{G}_T$ exponentiation With preprocessing	0.2

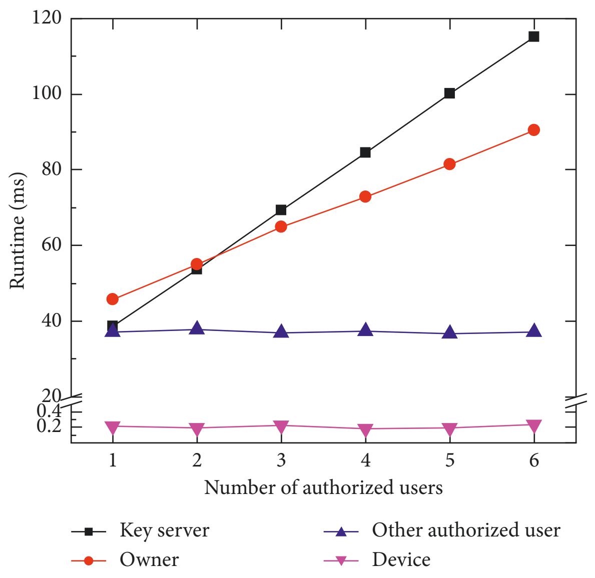


Figure 7: Computational overhead of each module of FSEE.

encapsulation for each authorized user. Its processing overhead is about 45 ms in the case of one authorized user and increases by 9 ms for each additional authorized user. Other users use their own IBE private key to obtain the encapsulation and interact with the server to get the final symmetric key, and the calculation overhead is about 37 ms. The key of the IoT device is directly updated by the owner, and devices only need to perform one symmetric decryption algorithm, so their computational cost is about 0.2 ms.

The overhead of the key server and device owner increases linearly with the number of authorized users. The key server is generally deployed on a workstation or cloud platform with abundant resources. While the device owner only manages the access rights of his own device through a mobile phone application, the number of authorized users n of one device is generally not too large ($n \leq 20$), and it is also considered a practical cost of performing about 2 n asymmetric operation per message on an iPhone 3GS [56]. Furthermore, it is also easy for the device owner to achieve constant calculation overhead using efficient identity-based public key broadcast encryption algorithms [57, 58].

7.2.2. Secure Communication. To realize end-to-end security of pub/sub communication in IoT, clients in FSEE use the computed device-specific key to encrypt and decrypt messages using AES. We study the resulting per-message processing overhead for publishing and subscribing to messages by comparing the computation overhead to unprotected communication, which uses the same pub/sub-stack without any enabled security features. As the runtime depends on the size of transmitted messages, we perform the measurements for message sizes of 128 B, 512 B, 1 kB, and 2 kB to represent smaller and larger value clusters typically used in IoT.

As shown in Figure 8, the increased processing overhead per message is about 0.2 ms-0.3 ms at publisher and subscriber compared to communications without any enabled security features. Overall, the high level of security achieved by applying FSEE to sensitive pub/subcommunication in IoT only introduces a modest increase in computation time, which is suitable even for resource-constrained IoT devices.

7.2.3. Comparison to Existing Schemes. Table 3 compares FSEE with other systems in regard to confidentiality, forward secrecy, decentralized authorization, deployability, and performance. The schemes based on chaos cryptography is designed for one-to-one communication, while our work is aimed at one-to-many communication, so chaos cryptography-based schemes are not included in the scope of comparison. We re-implemented JEDI using our crypto library for a fair comparison and use it as a medium to compare with proxy re-encryption-based schemes [15] and ABE (attribute-based encryption)-based schemes [12, 40].

7.2.4. Confidentiality. Most of the message broker in commercial IoT system is not completely trusted. In a scheme that completely relies on a trusted broker, the broker can get all user's data that does not meet the confidentiality requirement.

7.2.5. Forward Secrecy. Most current designs of secure communication protocols consider forward secrecy an indispensable design goal [19]. None of the existing solutions satisfies forward secrecy.

7.2.6. Decentralized Authorization. In a scheme relying on a centralized trusted server or broker, the authorization and revocation of device access rights must be managed by the server or broker. Therefore, these schemes have poor scalability, and the server may become the target of the adversary.

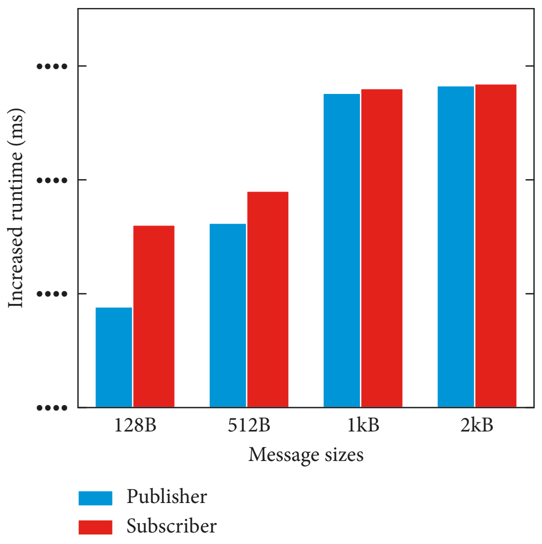


Figure 8: Increased per-message runtime at publisher and subscriber compared to unprotected communication for different message sizes.

7.2.7. Deployability. Reference [38] divides the broker's function into three part, and the designed new broker needs to be customized to meet the corresponding requirements; reference [18] needs to install special hardware, both of which are difficult to deploy.

7.2.8. Performance. Trusted broker-based schemes do not meet the requirement of confidentiality, and References [18, 38] are difficult to deploy, so these schemes are not re-implemented on our experimental platform. Trusted key server-based schemes are faster than FSEE when distribute symmetric key, but similar to FSEE when using a symmetric key to transmit data in the common case. In JEDI [11], it re-implemented PICADOR [15], whose performance is comparable to JEDI. JEDI uses WKD-IBE [59] to distribute the symmetric key, the time to transmit a symmetric key (including one encryption and one decryption) is about 95 ms, and the decryption time contains the time to generate a decryption key for the encrypted pattern and the time to decrypt the ciphertext. When computing the time of JEDI, we use a pattern of 20 attributes, with the first 14 attributes representing the URI and the last 6 attributes representing the time. ABE algorithms are more complicated than IBE, and the computational overhead of ABE-based schemes is much bigger than IBE based ones, which is not appropriate for resource-constrained IoT devices. FSEE takes about 115 ms to distribute a symmetric key using BA-GKE. The computational overhead of FSEE is slightly higher than JEDI, but it can provide forward secrecy.

In summary, existing schemes fall into one of the four categories. (1) Trusted broker-based schemes need to fully

Table 3: Comparison of different end-to-end encryption schemes.

Schemes	Confidentiality	Forward secrecy	Decentralized authorization	Deployability	Performance
Trusted broker based, as TLS [3], MOUCON [9]	×	×	×	✓	—
Trusted key server based, as [13]	✓	×	×	✓	Faster than FSEE when distribute symmetric key, similar to FSEE when using symmetric key to transmit regular data
PRE based, as [15–17]	✓	×	×	✓	Similar to JEDI
Secret sharing based, as [38]	✓	×	×	×	—
Trusted hardware based, as [18]	✓	×	×	×	—
IBE based, as JEDI [11]	✓	×	✓	✓	Takes about 95 ms (including device encryption and user decryption) to distribute symmetric key
ABE based, as [12, 40]	✓	×	✓	✓	Much slower than JEDI
FSEE	✓	✓	✓	✓	Takes about 115 ms to distribute symmetric key in case of one authorized user

— represents that we do not re-implement these schemes on our experimental platform.

trust the broker, which contradicts the confidentiality requirement. (2) In trusted key server- or broker-based schemes, a single component is trusted for all resources in the system. Granting or revoking access to a user requires the participation of the trusted party to generate new keys. (3) Some schemes need to redevelop a customized message broker or install special hardware, which is difficult to deploy. (4) None of the existing schemes achieve forward secrecy.

8. Conclusion

In this work, we propose FSEE, a forward secure end-to-end encrypted message transmission system for pub/sub communication in IoT, which addresses the confidentiality concerns of malicious adversaries that fully compromise the message broker. To support FSEE, a novel group key exchange protocol BA-GKE is designed to distribute symmetric key per device between each device and its multiple authorized users, and its security is verified by ProVerif. FSEE can achieve confidential and forward secure end-to-end communication in IoT, and support decentralized authorization, deployability, and asynchronous communication at the same time. Experimental evaluations were also conducted to test the performance of FSEE. Further explorations of efficient user revocation and encrypted data persistence are also interesting and may be our future research direction.

Appendix

A. The Proverif description of BA-GKE

(1) free c: channel.
(2) type G.
(3) type GT.
(4) type exponent.
(5) const p: G [data].
(6) fun sm (G, exponent): G.
(7) fun e (G, G): GT.
(8) fun addG (G, G): G.
(9) fun addexp (exponent, exponent):exponent.
(10) equation for all x: exponent, y: exponent; sm (sm (p, x), y) = sm (sm (p, y), x).
(11) equation for all $G1 : G$, x: exponent, $G2 : G$, y: exponent; e (sm ($G1, x$), sm ($G2, y$)) = e (sm ($G1, y$), sm ($G2$, x)).
(12) type nonce.
(13) fun senc (bitstring, nonce):bitstring.
(14) reduc forall x:bitstring, y:nonce; sdec (senc (x, y), y) = x.
(15) type $p$key.
(16) type $s$key.
(17) fun pk (skey):$p$key.
(18) fun aenc1 (G, $p$key): bitstring.
(19) reduc forall x:G, y:$s$key; adec1 (aenc1 (x, pk (y)), y) = x.
(20) fun aenc2 (nonce, $p$key):bitstring.
(21) reduc for all x:nonce, y:$s$key; adec2 (aenc2 (x, pk (y)), y) = x.
(22) fun bitstring_to_G (bitstring): G [data, typeConverter].
(23) fun $H0$ (G, bitstring):exponent.
(24) fun $H1$ (GT):nonce.
(25) fun $H2$ (bitstring, G, bitstring, G, G):nonce.
(26) fun $H3$ (nonce, G):nonce.
(27) fun Inverse (exponent):exponent.

```
(28) fun mac (G, nonce):bitstring.
(29) free s, mt, secretowner, secretuser: bitstring [private].
(30) query attacker (secretowner).
(31) query attacker (secretuser).
(32) table akekey (bitstring, G).
(33) table responderPk (bitstring, pkey).
(34) table responderSk (bitstring, skey).
(35) table SIDdk (nonce, exponent).
(36) table SIDsk (nonce, nonce).
(37) not attacker (new masterkey).
(38) event beginARake (bitstring, bitstring).
(39) event endARake (bitstring, bitstring).
(40) event beginRAake (bitstring, bitstring).
(41) event endRAake (bitstring, bitstring).
(42) query x:bitstring, y:bitstring; event (endARake (x, y)) == > event (beginARake (x, y)).
(43) query x:bitstring, y:bitstring; event (endRAake (x, y)) == > event (beginRAake (x, y)).
(44) let processA (IDA:bitstring, IDB:bitstring, IDR: bitstring) =
(45) new a:exponent;
(46) out (c, (IDA, sm (bitstring_to_G (IDA), a)));
(47) get akekey ( = IDA, skA) in
(48) in (c, m:bitstring);
(49) let (IDX:bitstring, WB : G) = m in
(50) let expAA:exponent = addexp (a, H0 (sm (bitstring_to_G (IDA), a), IDX)) in
(51) let expAB:exponent = H0 (WB, IDA) in
(52) let AG1 : G = sm (skA, expAA) in
(53) let AG2 : G = addG (WB, sm (bitstring_to_G (IDX), expAB)) in
(54) let akeAsk:nonce = H1 (e (AG1, AG2)) in
(55) out (c, senc ((IDA, s), akeAsk));
(56) in (c, mes:bitstring);
(57) let ( = IDB, messg:bitstring) = sdec (mes, akeAsk) in
(58) in (c, ek:G);
(59) let Asid:nonce = H2(IDA, sm (bitstring_to_G (IDA), a), IDX, WB, ek) in
(60) new Ai:exponent;
(61) let C : G = sm (p, Ai) in
(62) let ka:G = sm (ek, Ai) in
(63) let kafinal: nonce = H3 (Asid, ka) in
(64) get responderPk ( = IDR, rPk) in
(65) event beginARake (IDA, IDR);
(66) out (c, aenc1 (C, rPk));
(67) out (c, aenc2 (akeAsk, rPk));
(68) out (c, aenc2 (Asid, rPk));
(69) out(c, senc ((IDA, secretowner), kafinal));
(70) in (c, (IDr:bitstring, mfromr:bitstring));
(71) if IDr = IDR then
(72) event endARake (IDA, IDR).
(73) let processB (IDB:bitstring, IDA:bitstring) =
(74) new b:exponent;
(75) out (c, (IDB, sm (bitstring_to_G (IDB), b)));
(76) get akekey ( = IDB, skB) in
(77) in (c, m:bitstring);
(78) let (IDY:bitstring, WA : G) = m in
(79) let expBB:exponent = addexp (b, H0 (sm (bitstring_to_G (IDB), b), IDY)) in
(80) let expBA:exponent = H0(WA, IDB) in
(81) let BG1 : G = addG (WA, sm (bitstring_to_G (IDY), expBA)) in
(82) let BG2 : G = sm (skB, expBB) in
(83) in (c, ms:bitstring);
(84) let ( = IDA, mb:bitstring) = sdec (ms, H1(e (BG1, BG2))) in
(85) let akeBsk:nonce = H1 (e (BG1, BG2)) in
(86) out (c, senc ((IDB, mt), akeBsk));
(87) new dk:exponent;
(88) out (c, sm (p, dk));
(89) let Bsid:nonce = H2(IDY, WA, IDB, sm (bitstring_to_G (IDB), b), sm (p, dk)) in
(90) insert SIDdk (Bsid, dk);
(91) insert SIDsk (Bsid, akeBsk);
(92) in (c, (BCr : G, macr1:bitstring, rsid1:nonce));
(93) get SIDsk ( = rsid1, macverkey:nonce) in
(94) let macr2:bitstring = mac (BCr, macverkey) in
(95) if macr1 = macr2 then
(96) get SIDdk ( = rsid1, siddk:exponent) in
(97) let BK: G = sm (BCr, siddk) in
(98) let macsend:bitstring = mac (BK, macverkey) in
(99) out (c, (BK, macsend, rsid1)).
(100) let processR (IDA:bitstring, IDB:bitstring, IDR: bitstring) =
(101) in (c, c1:bitstring);
(102) in (c, c2:bitstring);
(103) in (c, c3:bitstring);
(104) get responderSk ( = IDR, rsk) in
(105) event beginRAake (IDR, IDA);
(106) let rC: G = adec1 (c1, rsk) in
(107) let MACk:nonce = adec2 (c2, rsk) in
(108) let rsid:nonce = adec2 (c3, rsk) in
(109) new t:exponent;
(110) let BC: G = sm (rC, t) in
```

(111) let tinverse:exponent = Inverse (*t*) in
(112) let macr:bitstring = mac (*BC*, MAC*k*) in
(113) out (*c*, (*BC*, macr, rsid));
(114) in (*c*, (*BKr*: *G*, macfroms:bitstring, rsidfroms: nonce));
(115) let macfroms1:bitstring = mac (BKr, MACk) in
(116) if macfroms1 = macfroms &&;rsidfroms = rsid then
(117) let kr:*G* = *sm* (*BKr*, tinverse) in
(118) let krfinal:nonce = *H*3 (rsid, *kr*) in
(119) in (*c*, (IDi:bitstring, mfromi:bitstring));
(120) if ID*i* = IDA then
(121) event endRAake (IDR, IDA);
(122) out (*c*, senc ((IDR, secretuser), krfinal)).
(123) process
(124) new masterkey:exponent;
(125) new IDA: bitstring; new IDB: bitstring; new IDR: bitstring;
(126) let *skA*: *G* = *sm* (bitstring_to_*G* (IDA), masterkey) in
(127) let *skB*: *G* = *sm* (bitstring_to_*G* (IDB), masterkey) in
(128) insert akekey (IDA, *skA*);
(129) insert akekey (IDB, *skB*);
(130) new *Rsk*:skey;
(131) insert responderSk (IDR, *Rsk*);
(132) let Rpk:pkey = *pk* (*Rsk*) in out (*c*, *Rpk*);
(133) insert responderPk (IDR, Rpk);
(134) (!processA (IDA, IDB, IDR)) | (!processB (IDB, IDA)) | (!processR (IDA, IDB, IDR) | phase 1; out (*c*, Rsk))

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this article.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (62002374).

References

[1] W. Zhou, Y. Jia, Y. Yao et al., "Discovering and understanding the security hazards in the interactions between iot devices, mobile apps, and clouds on smart home platforms," in *Proceedings of the 28th USENIX Security Symposium, USENIX Security 2019*, N. Heninger and P. Traynor, Eds., pp. 1133–1150, USENIX Association, Santa Clara, CA, USA, August 2019, https://www.usenix.org/conference/usenixsecurity19/presentation/zhou.

[2] J. Wilson, R. S. Wahby, H. Corrigan-Gibbs, D. Boneh, P. Levis, and K. Winstein, "Trust but verify: auditing the secure internet of things," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'17*, T. Choudhury, S. Y. Ko, A. Campbell, and D. Ganesan, Eds., pp. 464–474, ACM, Niagara Falls, NY, USA, June 2017, [Online]. Available:

[3] E. Rescorla, *The Transport Layer Security (TLS) Protocol Version 1.3, Internet Engineering Task Force (IETF) Std*, https://www.rfc-editor.org/rfc/rfc8446.txt [Online]. Available:, aug 2018.

[4] M. Henze, R. Matzutt, J. Hiller et al., "Complying with data handling requirements in cloud storage systems," *CoRR*, vol. 11448, 2018, http://arxiv.org/abs/1806.11448.

[5] M. S. Ali, M. Vecchio, M. Pincheira, K. Dolui, F. Antonelli, and M. H. Rehmani, "Applications of blockchains in the internet of things: a comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1676–1717, 2019, https://doi.org/10.1109/COMST.2018.2886932 [Online]. Available:

[6] F. Maggi, R. Vosseler, and D. Quarta, *The Fragility of Industrial Iot's Data Backbone: Security and Privacy Issues in Mqtt and Coap Protocols*Trend Micro Research, Tech. Rep, Jul 2018, https://documents.trendmicro.com/assets/white_papers/wp-the-fragility-of-industrial-IoTs-data-backbone.pdf?v1.

[7] N. Huq, R. Vosseler, and M. Swimmer, "Cyberattacks against intelligent transportation systems," July 2017, https://documents.trendmicro.com/assets/white_papers/wp-cyber-attacks-against-intelligent-transportation-systems.pdf.

[8] M. A. Jan, W. Zhang, M. Usman, Z. Tan, F. Khan, and E. Luo, "Smartedge: an end-to-end encryption framework for an edge-enabled smart city application," *Journal of Network and Computer Applications*, vol. 137, no. 1–10, pp. 1–10, 2019, [Online]. Available.

[9] Y. Jia, L. Xing, Y. Mao et al., "Burglars' iot paradise: understanding and mitigating security risks of general messaging protocols on iot clouds," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy, SP 2020*, pp. 465–481, IEEE, San Francisco, CA, USA, May 2020, [Online]. Available:

[10] C.-M. Chen, X. Deng, W. Gan, J. Chen, and S. K. H. Islam, "A secure blockchain-based group key agreement protocol for iot," *The Journal of Supercomputing*, vol. 77, no. 8, pp. 9046–9068, 2021.

[11] S. Kumar, Y. Hu, M. P. Andersen, R. A. Popa, and D. E. Culler, "JEDI: many-to-many end-to-end encryption and key delegation for iot," in *Proceedings of the 28th USENIX Security Symposium, USENIX Security 2019*, N. Heninger and P. Traynor, Eds., pp. 1519–1536, USENIX Association, Santa Clara, CA, USA, August 2019, [Online]. Available:.

[12] F. Wang, J. Mickens, N. Zeldovich, and V. Vaikuntanathan, "Sieve: cryptographically enforced access control for user data in untrusted clouds," in *Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016*, K. J. Argyraki and R. Isaacs, Eds., pp. 611–626, USENIX Association, Santa Clara, CA, USA, March 2016, [Online]. Available:

[13] 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, M. Gupta, M. Abdelsalam, and S. Mittal, Eds., pp. 78–87, ACM, Virtual Event, USA, April 2021, [Online]. Available:

[14] L. Burkhalter, A. Hithnawi, A. Viand, H. Shafagh, and S. Ratnasamy, "Timecrypt: encrypted data stream processing at scale with cryptographic access control," in *Proceedings of the 17th USENIX Symposium on Networked Systems Design and Implementation*, R. Bhagwan and G. Porter, Eds., USENIX Association, Santa Clara, CA, USA, pp. 835–850, February 2020.

[15] C. Borcea, A. B. D. Gupta, Y. Polyakov, K. Rohloff, and G. Ryan, "PICADOR: end-to-end encrypted publish-subscribe information distribution with proxy re-encryption," *Future Generation Computer Systems*, vol. 71, pp. 177–191, 2017.

[16] Y. Polyakov, K. Rohloff, G. Sahu, and V. Vaikuntanathan, "Fast proxy re-encryption for publish/subscribe systems," *ACM Transactions on Privacy and Security*, vol. 20, no. 4, pp. 1–31, 2017.

[17] H. Shafagh, A. Hithnawi, L. Burkhalter, P. Fischli, and S. Duquennoy, "Secure sharing of partially homomorphic encrypted iot data," Edited by M. R Eskicioglu, Ed., in *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*, vol. 29, ACM, Delft, Netherlands, November 2017.

[18] C. Segarra, R. Delgado-Gonzalo, and V. Schiavoni, "MQT-TZ: secure MQTT broker for biomedical signal processing on the edge," in *Proceedings of the MIE 2020, Medical Informatics*, L. B. Pape-Haugaard, C. Lovis, I. C. Madsen, P. Weber, P. H. Nielsen, and P. Scott, Eds., vol. 270, pp. 332–336, IOS Press, Europe, Geneva, Switzerland, May 2020.

[19] N. Unger, S. Dechand, J. Bonneau et al., "Sok: secure messaging," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy, SP 2015*, pp. 232–249, IEEE Computer Society, San Jose, CA, USA, May 2015.

[20] C. Boyd and K. Gellert, "A modern view on forward security," *The Computer Journal*, vol. 64, no. 4, pp. 639–652, 2021, [Online]. Available:

[21] B. Poettering and P. Rösler, "Towards bidirectional ratcheted key exchange," in *Proceedings of the Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference*, H. Shacham and A. Boldyreva, Eds., vol. 10991, pp. 3–32, Springer, Santa Barbara, CA, USA, August 2018, [Online]. Available:

[22] J. Nam, S. Kim, S. Kim, and D. Won, "Provably-secure and communication-efficient scheme for dynamic group key exchange," *IACR Cryptol ePrint Arch*, p. 115, 2004.

[23] M. C. Gorantla, C. Boyd, and J. M. G. Nieto, "One round group key exchange with forward security in the standard model," *IACR Cryptol ePrint Arch*, p. 83, 2010 [Online]. Available:

[24] M. D. Green and I. Miers, "Forward secure asynchronous messaging from puncturable encryption," in *Proceedings of the 2015 IEEE Symposium on Security and Privacy, SP 2015*, pp. 305–320, IEEE Computer Society, San Jose, CA, USA, May 2015, [Online]. Available:

[25] F. Günther, B. Hale, T. Jager, and S. Lauer, "0-rtt key exchange with full forward secrecy," in *Lecture Notes in Computer Science*, J. Coron and J. B. Nielsen, Eds., vol. 10212pp. 519–548, 2017, [Online]. Available:

[26] H. Krawczyk, "HMQV: a high-performance secure diffie-hellman protocol," in *Proceedings of the Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference*, V. Shoup, Ed., vol. 3621, pp. 546–566, Springer, Santa Barbara, California, USA, August 2005, [Online]. Available:

[27] C. Boyd, G. T. Davies, K. Gjøsteen, and Y. Jiang, "Offline assisted group key exchange," in *Developments in Language Theory*, L. Chen, M. Manulis, and S. A. Schneider, Eds., vol. 11060, Springer, [Online]. Available:, , pp. 268–285, 2018.

[28] S. S. M. Chow and K. R. Choo, "Strongly-secure identity-based key agreement and anonymous extension," in *Proceedings of the Information Security, 10th International Conference, ISC 2007*, J. A. Garay, A. K. Lenstra, M. Mambo, and R. Peralta, Eds., vol. 4779, pp. 203–220, Springer, Valparaíso, Chile, October 2007, [Online]. Available:

[29] B. Kuang, A. Fu, W. Susilo, S. Yu, and Y. Gao, "A survey of remote attestation in internet of things: attacks, countermeasures, and prospects," *Computers & Security*, vol. 112, 2022 [Online]. Available:, Article ID 102498.

[30] M. N. Aman, M. H. Basheer, S. Dash et al., "Hatt: hybrid remote attestation for the internet of things with high availability," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7220–7233, 2020, [Online]. Available:

[31] M. N. Aman, M. H. Basheer, S. Dash et al., "Prom: passive remote attestation against roving malware in multicore iot devices," *IEEE System J.*, 2021.

[32] M. N. Aman, M. H. Basheer, and B. Sikdar, "Data provenance for iot with light weight authentication and privacy preservation," *IEEE Internet of Things Journal*, vol. 6, no. 6, [Online]. Available: Article ID 10441, 2019.

[33] M. J. Sadri and M. R. Asaar, "An anonymous two-factor authentication protocol for iot-based applications," *Computer Networks*, vol. 199, Article ID 108460, 2021.

[34] S. Dramé-Maigné, M. Laurent, L. Castillo, and H. Ganem, "Centralized, distributed, and everything in between," *ACM Computing Surveys*, vol. 54, no. 7, pp. 1–34, 2022, [Online]. Available:

[35] Z. Li, J. Hao, J. Liu, H. Wang, and M. Xian, "An iot-applicable access control model under double-layer blockchain," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 68, no. 6, pp. 2102–2106, 2021, [Online]. Available:

[36] R. Kalis and A. Belloum, "Validating data integrity with blockchain," in *Proceedings of the 2018 IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2018*, pp. 272–277, IEEE Computer Society, Nicosia, Cyprus, December 2018, [Online]. Available:

[37] J. Sun, Y. Su, J. Qin, J. Hu, and J. Ma, "Outsourced decentralized multi-authority attribute based signature and its application in iot," *IEEE Transactions on Cloud Computing*, vol. 9, no. 3, pp. 1195–1209, 2021.

[38] S. Belguith, S. Cui, M. R. Asghar, and G. Russello, "Secure publish and subscribe systems with efficient revocation," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018*, H. M. Haddad, R. L. Wainwright, and R. Chbeir, Eds., pp. 388–394, ACM, Pau, France, April 2018, [Online]. Available:

[39] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *Proceedings of the 2015 IEEE TrustCom/BigDataSE/ISPA*, pp. 57–64, IEEE, Helsinki, Finland, August 2015, . [Online]. Available:

[40] S. Yu, C. Wang, K. Ren, and W. Lou, "Achieving secure, scalable, and fine-grained data access control in cloud computing," in *Proceedings of the INFOCOM 2010. 29th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications*

Societies, pp. 534–542, IEEE, San Diego, CA, USA, March 2010, [Online]. Available:

[41] D. A. Trujillo-Toledo, O. R. López-Bonilla, E. E. García-Guerrero et al., "Real-time rgb image encryption for iot applications using enhanced sequences from chaotic maps," *Chaos, Solitons & Fractals*, vol. 153, Article ID 111506, 2021.

[42] J. D. Díaz-Muñoz, I. Cruz-Vega, E. Tlelo-Cuautle, J. M. R. Cortés, and J. de Jesús Rangel-Magdaleno, "Kalman observers in estimating the states of chaotic neurons for image encryption under mqtt for iot protocol," *The European Physical Journal - Special Topics*, 2021.

[43] A. M. González-Zapata, E. Tlelo-Cuautle, I. Cruz-Vega, and W. D. León-Salas, "Synchronization of chaotic artificial neurons and its application to secure image transmission under mqtt for iot protocol," *Nonlinear Dynamics*, vol. 104, no. 4, pp. 4581–4600, 2021.

[44] L. G. D. la Fraga, C. Mancillas-López, and E. Tlelo-Cuautle, "Designing an authenticated hash function with a 2d chaotic map," *Nonlinear Dynamics*, vol. 104, no. 1, pp. 4569–4580, 2021.

[45] E. García-Guerrero, E. Inzunza-González, O. López-Bonilla, J. Cárdenas-Valdez, and E. Tlelo-Cuautle, "Randomness improvement of chaotic maps for image encryption in a wireless communication scheme using pic-microcontroller via zigbee channels," *Chaos, Solitons & Fractals*, vol. 133, 2020 [Online]. Available:, Article ID 109646.

[46] Y. P. K. Nkandeu and A. Tiedeu, "An image encryption algorithm based on substitution technique and chaos mixing," *Multimedia Tools and Applications*, vol. 78, no. 8, pp. 10013–10034, 2019, [Online]. Available:.

[47] R. Canetti and H. Krawczyk, "Analysis of key-exchange protocols and their use for building secure channels," in *Lecture Notes in Computer Science*, B. Pfitzmann, Ed., vol. 2045, , pp. 453–474, Springer, 2001.

[48] B. Blanchet, "Automatic verification of correspondences for security protocols," *Journal of Computer Security*, vol. 17, no. 4, pp. 363–434, 2009.

[49] K. T. Nguyen, N. Oualha, and M. Laurent, "Authenticated key agreement mediated by a proxy re-encryptor for the internet of things," in *Computer Security - ESORICS 2016*, I. G. Askoxylakis, S. Ioannidis, S. K. Katsikas, and C. A. Meadows, Eds., vol. 9879, pp. 339–358, Springer, 2016.

[50] E. Foundation, Eclipse paho java client, 2021.

[51] M. Q. Hive: documentation v4.7.

[52] A. D. Caro and V. Iovino, "jpbc: Java pairing based cryptography," in *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*, pp. 850–855, IEEE Computer Society, Kerkyra, Corfu, Greece, July 2011, [Online]. Available:

[53] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Advances in Cryptology - CRYPTO 2001*, J. Kilian, Ed., vol. 2139, , pp. 213–229, Springer, 2001.

[54] M. Q. Hive, *Hivemq-Community-Edition*, https://github.com/hivemq/hivemq-community-edition/, 2021.

[55] M. Q. Hive, *Hivemq Extension Sdk 4.7.1 Api*, https://www.hivemq.com/docs/hivemq/4.7/extensions-javadoc/index.html, 2021.

[56] K. Cohn-Gordon, C. Cremers, L. Garratt, J. Millican, and K. Milner, "On ends-to-ends encryption: asynchronous group messaging with strong security guarantees," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018*, D. Lie, M. Mannan, M. Backes, and X. Wang, Eds., pp. 1802–1819, ACM, Toronto, ON, Canada, October 2018, [Online]. Available:

[57] C. Delerablée, "Identity-based broadcast encryption with constant size ciphertexts and private keys," in *Advances in Cryptology - ASIACRYPT 2007*, K. Kurosawa, Ed., in *Proceedings of the Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security*, vol. 4833, pp. 200–215, Springer, Kuching, Malaysia, December 2007.

[58] D. Boneh, C. Gentry, and B. Waters, "Collusion resistant broadcast encryption with short ciphertexts and private keys," in *Proceedings of the Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference*, V. Shoup, Ed., vol. 3621, pp. 258–275, Springer, Santa Barbara, California, USA, August 2005.

[59] M. Abdalla, E. Kiltz, and G. Neven, "Generalized key delegation for hierarchical identity-based encryption," in *Proceedings of the 12th European Symposium On Research In Computer Security Proceedings, ser. Lecture Notes in Computer Science*, J. Biskup and J. López, Eds., pp. 139–154, Springer, Dresden, Germany, September 2007.