SEEMQTT: Secure End-to-End MQTT-Based Communication for Mobile IoT Systems Using Secret Sharing and Trust Delegation

Mohammad Hamad, Andreas Finkenzeller, Hangmao Liu, Jan Lauinger, *IEEE, Member*, Vassilis Prevelakis, *IEEE, Member*, and Sebastian Steinhorst, *IEEE, Senior Member*

***Abstract*—The Publish/Subscribe (Pub/Sub) model offers a communication scheme that is appropriate for a variety of mobile Internet of Things (IoT) systems (e.g., autonomous vehicles). In most of these systems, ensuring the End-to-End (E2E) security of exchanged information is a critical requirement. However, the Pub/Sub scheme lacks appropriate mechanisms to ensure the E2E security, even when state-of-the-art solutions, such as Transport Layer Security (TLS) or Attribute-Based Encryption (ABE), were adopted. These solutions either do not offer E2E security or are infeasible to be adopted in mobile IoT systems with resource-constrained platforms. In this paper, we propose a framework, so-called SEEMQTT, to ensure secure E2E Pub/Sub-based communication for mobile IoT systems. Our solution allows the publisher to encrypt the published messages and control which subscribers can decrypt these messages without violating the decoupling requirement of the Pub/Sub model. Our solution leverages multiple honest-but-curious KeyStores to store secret shares generated from a secret key using a secret sharing scheme. The links between the publisher and every KeyStores are secured using Identity-Based Encryption (IBE). The publisher uses the secret key to encrypt published messages. Trust delegation is used to authorize certain subscribers to access these shares and consequently decrypt the published messages. We provide an Arduino-based library that implements our proposed protocol. Also, we perform an extensive performance evaluation using real IoT hardware. Experimental results show that adopting our proposed solution, SEEMQTT, makes E2E security for mobile IoT systems feasible.**

***Index Terms*—End-to-End Security, Pub/Sub Model, MQTT, Key Sharing, Trust Delegation.**

I. Introduction

Autonomous vehicles, unmanned aerial vehicles (UAVs), and smart bicycles are examples of smart mobility solutions that aim to improve traffic flows, reduce the incidence of critical situations, improve road usability, encourage transportation means-sharing schemes, and increase the drivers' and passengers' comfort. To achieve these goals, these systems need to exchange and share vast amounts of information with different cloud services (e.g., vehicle-to-cloud [1], bicycle-to-cloud [2], and UAV-to-cloud [3]). The main challenges that such communication faces are the unstable connectivity with cloud services due to the continuous movement of the nodes, and the need to exchange privacy-related data securely with many services owned by different authorities.

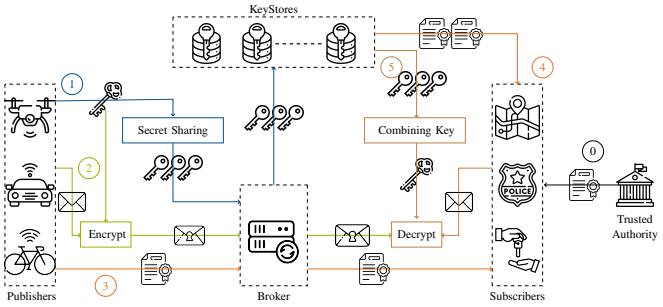


Fig. 1: A high level system architecture of our SEEMQTT framework. (1) The publisher generates secret shares from a key and exchanges these shares with the KeyStores over an IBE-based secure link. (2) Later, the publisher uses that key to encrypt published messages. (3) Also, the publisher defines a security credential that allows subscribers to retrieve the shares. (4) The subscriber uses the received credentials ((0) including the ones received from the Trusted Authority) to request the secret shares from the KeyStores. (6) Then, subscribers combine the shares and get the key to decrypt the received messages.

The Pub/Sub model (§ II-A) is a communication paradigm that enables a sender, known as a publisher, to disseminate messages to multiple receivers, known as subscribers, at once via a mediator, known as a broker [4]. The Pub/Sub model provides full decoupling in time, space, and flow between publishers and subscribers. These characteristics make the Pub/Sub model an excellent candidate to implement communication between mobile IoT systems and the cloud. One of the famous protocols that follow the Pub/Sub model is Message Queue Telemetry Transport (MQTT) (§ II-A2). Many IoT mobile systems already use MQTT to exchange data with cloud services [5]–[7]. However, in terms of security, the Pub/Sub model, in general, and MQTT, in particular, are susceptible to a wide variety of security threats that affect confidentiality, integrity, and access control of published data [8], [9]. Such security vulnerabilities are the main obstacles when using MQTT in systems that require E2E encryption between publishers and subscribers.

M. Hamad, A. Finkenzeller, H. Liu, J. Lauinger, and S. Steinhorst are with the Department of Electrical and Computer Engineering, Technical University of Munich, Munich, 80333 Germany, E-mail: (firstname.lastname@tum.de).

V. Prevelakis is with Institute of Computer and Network Engineering, Technical University of Braunschweig, Braunschweig, 38106 Germany, E-mail: (prevelakis@ida.ing.tu-bs.de).

The common way to ensure confidentiality of the transmitted data while using MQTT is by considering the broker as a trusted component. Then, both the publisher and subscriber can set up a secure connection with the broker (e.g., by using TLS [10]). This solution does not ensure the E2E encryption since the broker still needs to decrypt the message transmitted by the publisher before encrypting and forwarding it to all authorized registered subscribers. Trusting the broker puts the entire system at risk if the broker gets compromised or behaves maliciously (i.e., by forwarding the messages to unauthorized subscribers). Additionally, resource-limited publishers will suffer from the significant communication overhead of the TLS handshake [11], especially in the case of short-lived connections. Another issue is the need for a key management mechanism to store and verify the certificate of each broker. This becomes a severe issue if the publisher is a mobile node (such as a traveling vehicle) that must interact with many brokers who may belong to various certificate authorities.

ABE [12], [13] was adopted recently by many authors (e.g., [14], [15]) to secure the Pub/Sub model. ABE is a type of public-key encryption (asymmetric) that ensures a fine-grained access control mechanism to encrypted data based on flexible access policies. By using ABE, a publisher can encrypt a message using a key extracted from specific attributes where only authorized subscribers (who have these properties) can read the data. However, adopting such a solution comes with significant overhead, even for mid-power and non-constrained devices [16], [17]. That makes the adoption of ABE questionable for devices with limited resources.

Another way to ensure E2E encryption between the publishers and subscribers without violating the decoupling requirements is by using an external *trusted* component to support the storage and distribution of the symmetric keys [18], [19]. These solutions move the problem from trusting the broker to trusting another single point of failure. Also, they require a pre-shared secret between the publisher and the trusted key storage. Thus, this approach is not applicable for mobile IoT systems.

A. Requirements

Based on the properties of mobile IoT systems that we have mentioned above, we identify four requirements that any proposed secure Pub/Sub or MQTT-based solution must fulfill:

- *(R1) No pre-shared secret*: the continuous mobility of the publisher within different domains makes having pre-shared secret keys with other domain nodes (i.e., subscribers or key storage components) impractical.
- *(R2) E2E Encryption*: the system must ensure confidentiality and integrity of the published data without trusting the broker.
- *(R3) Decoupled authorization*: the system must give the publisher of the data the capability to control which subscriber is able to read this data (without violating the decoupling principle).
- *(R4) Efficiency*: it should be possible to deploy and run the system using constrained devices similar to these ones used in the mobile IoT systems.

The existing solutions that we have reviewed do not fulfill one or more of these requirements (a comprehensive comparison of our SEEMQTT framework with existing solutions will be introduced in § VIII).

B. Contribution

In this work, we propose an innovative framework, so-called SEEMQTT, that maintains E2E data confidentiality, integrity, and authorization between the publisher and the subscribers in MQTT-based communication for mobile IoT systems by integrating and extending cryptographic approaches, specifically secret key sharing (§ II-B1), IBE (§ II-B2), and trust management access control (§ III). As shown in Fig. 1, the publisher uses a secret sharing scheme to distribute a symmetric key, which is used to encrypt the published data, among multiple *honest-but-curious*[1] servers, so-called *KeyStores*. Then, the publisher specifies conditions that allow subscribers to retrieve these secret shares from those KeyStores. Only subscribers who have sufficient credentials can receive and reconstruct the key and use it to decrypt the published messages. In order to exchange the shares securely, the publisher needs to establish a secure channel with every KeyStore[2]. IBE is used to support the establishment of such a secure link.

Our solution does not require any pre-shared secrets between the publisher and KeyStores. In addition, it ensures the E2E security of the published messages. Also, it gives the publisher the ability to control which subscribers are authorized to read the decrypted message by changing the key and the initial security policy. Finally, our system was implemented and evaluated using constrained devices.

The main contributions of this paper are as follows:

- We design a secure lightweight Pub/Sub-based framework that ensures the E2E confidentiality and integrity of published data and gives the publisher full capability to control who can decrypt this data. All that is possible without both the need to trust the broker and sharing any pre-secrets with the KeyStores (§ V).
- We develop an open-source Arduino library called SEEMQTT.[3] This library implements the proposed solution and extends the Arduino PubSubClient library which implements MQTT messaging (§ VI).
- We show the feasibility and applicability of our proposed protocol by evaluating it using embedded resource platforms. (§ VII).

Parts of this paper have been published in an earlier conference paper [21]. The critical improvement upon the previous results is the use of IBE and secret sharing. That implies changes in many phases of the protocol. In addition, we provide an Arduino-based library that implements our proposed protocol. Also, we have performed new experiments to evaluate the new aspects of the protocol. Finally, we have performed an intensive comparison with existing solutions.

[1]The honest-but-curious participant in a protocol is a legitimate participant that will follow the defined protocol but will attempt to learn all possible information from legitimately received messages [20].

[2]It would be more practical to have a group of KeyStores in every domain that serves specific numbers of publishers and subscribers

[3]https://github.com/tum-esi/SEEMQTT

II. Background

In this section, we provide an overview of the basic concepts that our proposed solution was built upon. Section II-A discusses the Pub/Sub model components and MQTT, while § II-B introduces the cryptographic background.

A. Publish/Subscribe Model

1) Pub/Sub components: the Pub/Sub model contains three main participants:

- *Publisher (Thing)*: which is responsible for producing or collecting data and publishing that data. Usually, devices with constrained resources in the term of computational power, energy, and memory are used as publishers. In this work, we assume that the publisher is a Class 1 to Class 4 constrained device [22].
- *Subscriber (Service Provider)*: which represents the node that will receive and process the published data. Subscribers can be a thing or a virtual server running in the cloud, and each one of these servers has certain capabilities that make it valid to process specific published data. In this work, we did not consider the case of using a thing as a subscriber.
- *Broker*: represents the device that receives the subscriptions from subscribers, filters incoming data from the thing, and forwards it to interested service providers. Published data can pass through multiple brokers until it reaches the service provider. The broker usually operates in an untrusted public network.

The Pub/Sub model allows subscribers to specify their interest in the published data in different ways. The two most widely-used subscription schemes are topic-based and content-based subscriptions. Using topic-based subscription allows each subscriber to receive all the messages published to topics he subscribed to. On the other hand, using content-based subscription allows each subscriber to receive all messages that have attributes that match the constraints he defined in advance. In this work, we consider the use of topic-based subscription schemes only.

2) MQTT: There are many messaging frameworks which follow the Pub/Sub philosophy. MQTT is one of these frameworks which was standardized in 2016 [23]. MQTT has many advantages, such as the fast response and the low bandwidth usage which make it an ideal communication protocol for resource-constrained devices. The sequence interaction of the Pub/Sub model using MQTT contains the following packets:

- `CONNECT`: each node (publisher and subscriber) needs to connect to the broker before being able to publish or receive any messages; therefore, the first packet that a node must exchange with the broker is the `CONNECT` packet. This message is required once over a network connection.
- `CONNACK`: whenever the node is connected, the broker sends an acknowledgment in response to the `CONNECT` request received from that node.
- `SUBSCRIBE`: this packet is sent by a node to the broker to register its interest in one `topic` or more. The broker sends an acknowledgment (`SUBACK`) packet to that node for each subscription request.
- `PUBLISH`: publisher nodes use this packet to publish messages into certain `topic`. The broker uses this packet also to forward these messages to all nodes which subscribed this `topic`. Based on the quality of service that was defined by the publisher, the broker may send an acknowledgment to the publisher confirming the reception of each published messages.

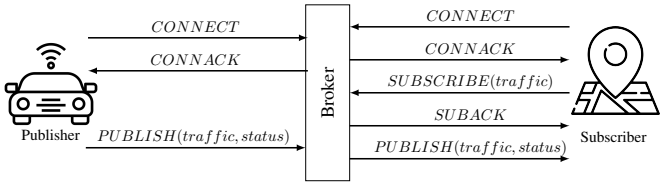


Fig. 2: MQTT control packets during the communication between a vehicle (Publisher) that reports the traffic status to one cloud service (Subscriber). Messages are ordered from top to bottom. However, they can have different order too due to MQTT decoupling. [5]

Fig. 2 shows an example of using all these packets when a vehicle and one cloud service use the MQTT protocol to exchange the traffic status. The figure shows how the two nodes firstly connect to a message broker. Then, the cloud service subscribes to a topic "`traffic`." The vehicle uses the same topic to publish the current status. Each time the vehicle publishes a reading, the broker forwards that reading to the cloud service to analyze that data. In this work, we use an open source message broker that implements the MQTT protocol called Eclipse Mosquitto[4].

3) MQTT Built-in Security Solutions: One of the main characteristics of the MQTT-based communication is that the broker has access to all the published data (this characteristic is inherited from the Pub/Sub model and can be found in all protocols that implement this model). From a security perspective, this design decision demands trusting the broker. However, trusting such a device is not always possible in many applications (e.g., vehicle-to-cloud communication [21]) where ensuring end-to-end security between the publisher and the subscriber is required. A recent study [24] shows that many systems which used MQTT-based communication are vulnerable to many attacks. Furthermore, the same study shows that ad-hoc security solutions that were used to secure MQTT-based communication were ineffective and insecure.

The MQTT standard [25] recommends the use of TLS (when the usage is feasible) to set up a secure connection between the nodes (publishers or subscribers) and the broker and to ensure the confidentiality of the exchanged messages. To achieve that, each broker must have a cryptographic certificate (issued by a trusted authority) that can be used by the other nodes to authenticate that broker and set up a session key to secure the connection. Also, MQTT provides a capability for the system administrator to control which

[4]https://mosquitto.org/

[5]The figure and other figures have been designed using icons from Flaticon.com

node can publish and subscribe to a topic by using an access control list implemented within each broker. The authorization decision is based on information that the client provides, such as a user-name or an IP address.

All these solutions are based on the assumption that brokers are trusted. Also, TLS supports securing one-to-one communication but it does not support securing one-to-many communication, which is one of the main principles behind the Pub/Sub communication model. Furthermore, a TLS-based solution does not offer E2E security since the broker can still read all the published messages. The overhead of re-establishing a TLS-based secure link for short-lived connections is another issue that limits the use of such a solution. Finally, the authorization solutions do not give the publisher or its owner the capability to control who can access their data if the broker belongs to an administrative domain that is different from the one of the publisher.

B. Cryptographic Background

1) Secret Sharing Scheme: Secret sharing is a mechanism that enables a dealer to share a secret s among n shareholders. The reconstruction of the secret is only possible when a qualified subset of t shareholders cooperate. Shamir [26] proposed the first secret sharing scheme over $\mathbb{Z}_p$, where p is a prime number. This scheme is known as (t, n)-threshold secret sharing scheme. It has two main algorithms: `GenShare` to generate n shares of s, denoted as $\alpha_1, ..., \alpha_n$, and `ComShare` to combine any t valid shares and recover the secret s.

- $(\alpha_1, \ldots, \alpha_n) \leftarrow \texttt{GenShare}(n, t, s)$: this algorithm takes as an input n which represents the total number of shares that will be generated (it is required that $n < p$), t which represents the minimum number of shares that can be used to reconstruct the secret, and the secret s. The algorithm chooses $t-1$ random numbers $a_1, ..., a_{t-1} \xleftarrow{R} \mathbb{Z}_p$ and defines the polynomial f, which has the degree $t-1$ using (1).

$$f(x) := s + a_1 \cdot x + a_2 \cdot x^2 + \cdots + a_{t-1} \cdot x^{t-1} \ mod\ p \quad (1)$$

  where $f(0) = s$.
  Next, it computes $y_i = f(x_i)$ for $x_i = 1, \ldots, n$ and $y_i \in \mathbb{Z}_p$. Finally, each $\alpha_i := (x_i, y_i)$ should be distributed to the appropriate shareholders securely.
- $s \leftarrow \texttt{ComShare}(\alpha_1, \ldots, \alpha_t)$: this algorithm takes as an input t valid shares $\alpha_1, \ldots, \alpha_t$ (where $\alpha_i := (x_i, f(x_i))$) that belong to t points of the polynomial f of degree $t-1$. To calculate the secret s, the function uses the Lagrange Interpolation Formula to interpolate the t points using (2).

$$s = f(0) = \sum_{i=1}^{t} f(x_i) \prod_{1 \le j \le t, j \ne i} \frac{-x_j}{x_i - x_j} \ mod\ p \quad (2)$$

2) Identity Based Encryption: To securely send a message between a sender and a receiver using traditional public-key cryptography, the sender needs the receiver's public key. To obtain that key, a key distribution mechanism is required. In addition, the sender needs authentication mechanism to ensure that the public key belongs to the receiver. Therefore, a trusted party (which we refer to as Certificate Authority (CA)) is required to certify the key by binding the key to the receiver's identity and creating a so-called digital certificate. Using the CA's public key, the sender can verify the receiver's certificate before using its public key to initiate any secure link.

In order to simplify this process, IBE [27] was proposed. The main advantage of IBE is to eliminate the need for storing and verifying the digital certificates, thus eliminating the need for a CA. By adopting IBE, any string which identifies a party (e.g., IP address, email address, etc.) can be used as public key for that party. Using such an identity as a public key will simplify the key distribution process too.

It is worth to mention that IBE still requires a trusted party, called Private Key Generator (PKG), to create public parameters that can be used by all participants and to extract a secret key for each participant based on its public identity. Using the identity (ID) of the receiver with public parameters, the sender can encrypt a message that only the intended receiver can decrypt.

The IBE scheme includes four algorithms: `Setup`, `Extract`, `Encrypt`, and `Decrypt`. Boneh and Franklin [28] provide the first practical implementation of this scheme using pairings (Weil-Pairing) over elliptic curves and finite fields:

- $(MK_{PKG}, params) \xleftarrow{R} \texttt{Setup}()$: This algorithm is run by the PKG to create a master secret key (MK_{PKG}) and public parameters $params$. These parameters are made public for all participants while the MK_{PKG} is kept secret. To implement that, the algorithm generates a prime number q, two cycling groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order q, and bilinear map $\hat{e} : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$. Then, it chooses a random generator $P \in \mathbb{G}_1$, a random number $\alpha \in \mathbb{Z}_p^*$ and assign it to the master secret key ($MK_{PKG} = \alpha$), a cryptographic hash functions $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1^*$, a cryptographic hash functions $H_2 : \mathbb{G}_2 \rightarrow \{0,1\}^n$ for some n. The public system parameters are $params = \langle q, \mathbb{G}_1, \mathbb{G}_2, \hat{e}, n, P, \alpha P, H_1, H_2 \rangle$.
- $prKID \xleftarrow{R} \texttt{Extract}(MK_{PKG}, ID)$: This algorithm is run by the PKG too. it takes as input the master key (MK_{PKG}), the public parameters ($params$), and an ID ($ID \in \{0,1\}^*$) to create a private key ($prKID$) for that ID. To do so, the algorithm calculates $Q_{ID} = H_1(ID) \in \mathbb{G}_1^*$, and then it sets the private key using the calculated Q_{ID} as follows: $prKID = \alpha Q_{ID}$ where α is the master secret key (MK_{PKG}). The private key ($prKID$) is shared securely with the owner of ID.
- $C \xleftarrow{R} \texttt{Encrypt}(M, params, ID)$: This algorithm is run by the sender. It takes as input the message to be encrypted (M), the public parameters ($params$) and the ID of the receiver. It returns a ciphertext C. To perform that, the algorithm calculates $Q_{ID} = H_1(ID) \in \mathbb{G}_1^*$. Then, it selects a random number $r \in \mathbb{Z}_p^*$ and uses it to calculate C as follows: $C = \langle rP, M \oplus H_2(g_{ID}^r) \rangle$ where $g_{ID} = \hat{e}(Q_{ID}, \alpha P) \in \mathbb{G}_2^*$. It is important to note that calculating g_{ID} and Q_{ID} is independent of the message M. Therefore, these values can be calculated once for every public key ID [28] (We refer to this as the one

time overhead during the encryption algorithm).

- $M \leftarrow$ `Decrypt`$(C, prKID)$: This algorithm is run by the receiver. It takes as input he ciphertext $C = \langle c_1, c_2 \rangle$ where $c_1 = rP$ and $c_2 = M \oplus H_2(g_{ID}^r)$ and the private decryption key $prKID$ correspond to ID. It returns the message M as follows: $c_2 \oplus H_2(\hat{e}(prKID, c_1)) = M \oplus H_2(g_{ID}^r) \oplus H_2(\hat{e}(prKID, rP)) = M$.

The IBE is correct when (3) always holds for any message M, given the private key $prKID$ that is generated by algorithm `Extract` and the corresponding public key ID.

$$Decrypt(Encrypt(M, params, ID), prKID) = M \quad (3)$$

III. Decentralized Trust Delegation

The need for maintaining the space decoupling between the publisher and the subscriber makes the publisher's ability to control the subscribers' access to the published messages a very challenging aim. The standard approach for enabling such a control is by using a centralized access control scheme, which is usually implemented on the broker. Managing such a scheme requires a central authority with very detailed knowledge of the entire system (i.e., publishers, subscribers, topics, and access rules). However, this solution is not practical due to many reasons. The management of such a scheme in massive IoT systems is an infeasible mission. Moreover, using such an approach would pose increased security risks, especially if implemented within untrustworthy platforms such as the broker.

In regard to access control of published messages, the optimal solution is by letting each publisher directly control which subscribers are trusted to access the published data without the need for knowing their identities (i.e., public keys, IP addresses, etc.). KeyNote Trust Management [29], [30] seems very suitable to implement this solution. Trust Management is used to validate certain actions against a security policy. The authorization of these actions is based on the *credentials* that the requester has more than its actual identity. Each credential contains information about the entity granting the authorization (known as Authorizer), information about the recipient(s) of the authorization (each one known as Licensee), and the condition under which the Authorizer trusts the Licensee(s) to perform certain action(s). We will refer to each credential as $CR_{Licensee}^{Authorizer}$ where both the Authorizer's and Licensee's fields contain public keys.

A. Trust Delegation

One of the main characteristics of the KeyNote trust management is the trust delegation. Each Licensee can play the Authorizer's role and delegate the trust that he/she gained by a credential to other actors (Licensees) with new conditions (without violating the initial conditions). Fig. 3 shows how such a property can allow the creation of a trust relationship between one Authorizer, such as Pub, and a Licensee, such as Sub_1, indirectly via many intermediate Licensees (e.g., TA and TA_1). This property perfectly aligns with the decoupling needs of the Pub/Sub paradigm since Pub and Sub_1 are fully decoupled and do not require to know each other.

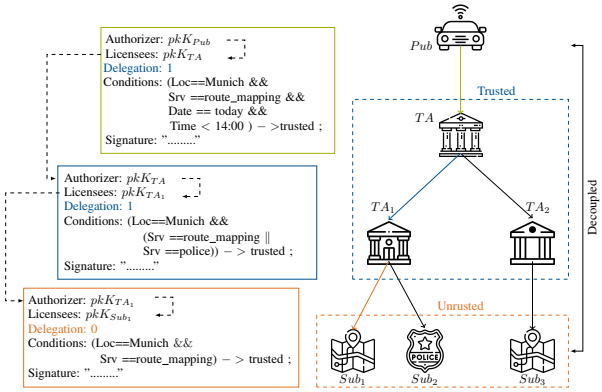


Fig. 3: Trust delegation between a vehicle (Pub) and cloud services (subscribers) via trusted TAs. The left part shows the chained credentials that reflect the trust delegation. Extended from [21].

To clarify how trust delegation can take place, we consider an example where a vehicle (as a publisher) shares traffic flow information that a cloud service can use to support dynamic routing. These published messages could include information that could be used to trace the vehicle if malicious services accessed it. Therefore, a car is interested in keeping its data secret and ensures that only services that analyze the data for dynamic routing ($Srv == route\_mapping$) and are responsible for the area where the car is traveling, let us say in the city of Munich, ($Loc == Munich$) can access this data for a specific period (e.g., $Date == Today \;\&\&\; Time <$ 14:00). Any other services should not be able to retrieve the published messages. In such a scenario, maintaining the information of each route mapping service is not practical. Instead, the vehicle needs only information about a global Trusted Authority (TA) (such as the German Federal Ministry of Transport and Digital Infrastructure) to issue a credential that allows this TA to access its published data under the aforementioned conditions. This TA creates another credential $CR_{TA_1}^{TA}$ which authorizes another trusted TA_1 to access or delegate the access of the data for certain services and in specific location (TA_1 can represent the Bavarian State Ministry for Housing, Construction and Transport). Using the delegation capability, TA_1 itself is able to authorize different subscribers to access this information based on the services they provide. Now, whenever the vehicle (Pub) issues CR_{TA}^{Pub}, it indirectly authorizes Sub_1.

Trust delegation is a double-edged sword. On the one hand, it enables the indirect trust between the publisher and the subscribers. On the other hand, it opens the door for any malicious subscribers to extend the trust maliciously by delegating the gained trust to other untrustworthy subscribers. To mitigate that, we have *extended* the KeyNote policy definition language to restrict the delegation capability to trusted parties only by adding a new field called *Delegation* to the structure of credentials. The Licensees can delegate the trust to other Licensees as long as their credentials allow that (i.e., Delegation:1). Fig. 3 shows how TA_1 is able to delegate the trust to Sub_1 while Sub_1 is not allowed to delegate the trust further (i.e., malicious subscribers will not be able to delegate

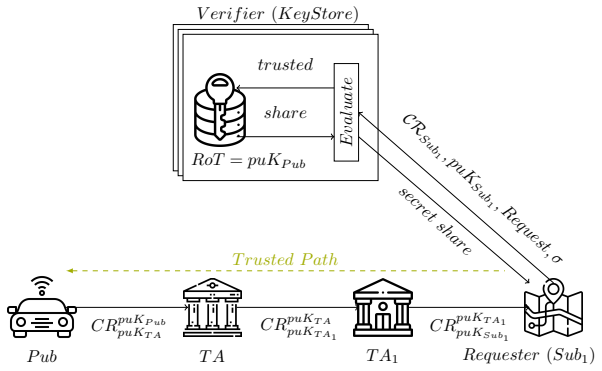


Fig. 4: The KeyStore evaluates the subscriber's request, and authorized it if a trusted path between the RoT and the subscriber can be found.

the trust to other subscribers since that will be detected during the evaluation of such newly created credentials).

It is critical to understand the frequency and the order of issuing such credentials. In our example, Pub can create a credential for TA every time it wants to change any of the existing conditions such as moving to a new location or extending the time of a previous credential. On the other hand, TA needs to create credentials to other intermediate trusted authorities (e.g., TA_1 and TA_2) only *once*. In the same manner, TA_1 needs to issue a credential for each subscriber once, too. Each subscriber can use its credential as long as it is valid (note that each Authorizer can limit the validity of each credential it creates by adding a validity period to its condition). Assigning these credentials usually happens during the system setup and does not need to happen after Pub has created its credential. As a result, both TA and TA_1 do not need to be online each time Pub publishes data or creates a new credential. The properties (such as location, service type, validity, etc.) that can be included in the credential condition must be shared with every publisher so they can form their credentials properly. Any new subscriber which would access the published data needs to communicate with the appropriate trusted authority to receive a security credential. The method of how subscribers contact a trusted authority to get a credential is beyond the paper's scope.

B. Credential Evaluation

To evaluate a request of one principal asking to access a particular resource, the verifier needs to determine whether there exists a chain of trust, from the owner of the resource, a.k.a. the Root of Trust (RoT), to that principal, granting access to the requested resource. Also, the principal must satisfy all the required conditions determined by the RoT to access that resource. The verifier can be the owner of the resource itself, or it can be any other component that considers the resource's owner as a RoT. In our system, the resource is the part of the key (secret share) that used by the publisher to encrypt the published message. The subscribers do not request the secret shares from the publisher (Pub) directly, but from any KeyStore in the system. Thus, a KeyStore plays the role of the verifier. To ensure that KeyStores will not deliver the secret shares to any subscriber unless it is authorized by the publisher Pub, each KeyStore considers the Pub as the RoT for every secret share generated by that Pub.

Each subscriber needs to sign its request and provide a list of all the credentials that it has. These list of credentials should authorize the subscriber to retrieve the secret shares from every KeyStore. We will refer to this list as $\mathcal{CR}_{Sub_1}$. To authorize that request, the KeyStore needs to validate the signature of the request and find the so-called "Trusted Path" (see Fig. 4), which links the requester's key (i.e., Sub_1's key) with the key of the RoT. If such a path is found and all the conditions in all credentials that form that path are satisfied, the KeyStore authorizes Sub_1's request and shares with it the secret share. Otherwise, the request will be denied.

C. Trust Delegation Algorithms

We define two main algorithms:

- $CR^{puK_{auth}}_{PuK_{licn}} \leftarrow \texttt{GenCredential}(puK_{auth}, puK_{licn}, Conditions, prK_{auth})$: This algorithm is run by the authorizer. It takes as input the authorizer's public key (puK_{auth}), the licensee's public key (puK_{licn}), a set of conditions, and the authorizer's private key (prK_{auth}) to create the credential $CR^{puK_{auth}}_{PuK_{licn}}$. This credential is signed using the authorizer's private key (prK_{auth}).
- $\{1, 0\} \leftarrow \texttt{Evaluate}(puK_{RoT}, \mathcal{CR}_{requester}, puK_{requester}, Request)$: This algorithm is run by the verifier. It takes as input the RoT's public key (puK_{RoT}), the list of all credentials that the requester has to support its request ($\mathcal{CR}_{requester}$), the requester's public key ($puK_{requester}$), and a $Request$. The verifier checks whether it can find a trusted path that links $puK_{requester}$ with puK_{RoT} based on $\mathcal{CR}_{requester}$. If so, the algorithm returns 1 (or something similar such as $trusted$). Otherwise, it returns 0.

IV. System and Threat Models

A. System Model

Our system contains:

- A Publisher node Pub which has a public key (puK_{Pub}) and a private key (prK_{Pub}). Pub can publish messages on different topics tp_1, tp_2, ..., tp_Z.
- Multiple Brokers Br_1, Br_2, ..., Br_B. Each broker can forward the published messages to the subscribers who are interested in these messages. Also, it can forward these messages to neighboring brokers to ensure data availability. For the sake of simplicity, we will consider the existence of one broker Br only.
- Subscriber nodes Sub_j. Each subscriber has one or more security credentials. Each subscriber can subscribe to any topics; thus, it receives published messages from the topics it has subscribed to. We refer to the set of subscribers who subscribe to a topic tp_i as $\mathcal{SUB}_{tp_i}$, as defined in (4).

$$\mathcal{SUB}_{tp_i} = \{Sub_j | Sub_j \rightarrow Br: \ SUBSCRIBE(tp_i)\} \quad (4)$$

- A Trusted Authority (TA) which is responsible for issuing credentials to each of providers. TA has a public key (puK_{TA}) which needs to be known by every component in the system.
- A trusted node that plays the role of a PKG. It generates the public parameters that can be used by all publishers and extracts a secret key for the KeyStore based on its public identity.
- A set of KeyStores $\mathcal{KS} = \{ks_1, ks_2, \ldots, ks_n\}$ where $|\mathcal{KS}| = n$. Each KeyStore has an identity (e.g., ID_{ks_1}, ID_{ks_2}, etc.), which is used as a public key, and a corresponding private key (e.g., $prKID_{ks_1}$, $prKID_{ks_2}$, etc.) which is extracted and shared by the PKG. In addition, each KeyStore has a repository to store cryptographic keys securely, and a Policy Evaluation Module (PEM) that evaluates the access requests to the stored keys (as explained in § III-B).

B. Threat Model

In this subsection, we present the threat model linked to the system model which was described above.

- *Trusted* TA and PKG: we assume that TA is fully trusted. TA will handle its private keys well and will not issue any credential to any subscriber who is not trustworthy. TA will verify the subscriber's identity (its public key) and check that the subscriber meets the conditions stated in the credential before issuing it a credential (delegate the trust to it). It is important to note that even though the subscriber is trustworthy and legitimate at the time of receiving the credential, it can turn into a malicious node any time later. Also, we assume that PKG is trusted, and it will handle its private key well. We assume that both TA and PKG will not try to decrypt and share any information about keys and exchanged messages.
- *Honest* publishers: we assume that the publisher will not publish malicious data and will not flood the system with a large number of messages to disrupt the broker's functionality (i.e., perform Denial of Service (DoS) attack). Also, we assume that publishers do not trust each other. Thus, they do not share the same key to encrypt messages published on the same topic.
- *Honest-but-curious* broker: we assume that each broker will perform all protocols correctly. The broker (or the attacker who compromises that broker) will attempt to learn all possible information about the content of published messages and the cryptographic keys that were used to encrypt the messages and the secret shares from legitimately received messages. In addition, the broker can alter any of the exchanged messages. Finally, we assume that the broker will not drop the exchanged messages to cause a DoS attack.
- *Honest-but-curious* KeyStores: on the one hand, we assume the KeyStore will be honest and it will follow the protocol steps and evaluate the subscriber credentials honestly. On the other hand, we assume that every KeyStore may attempt to learn all possible information about the content of published messages, the cryptographic keys of other KeyStores, and the key that the publisher uses to encrypt the published data from legitimately exchanged messages. It is critical to know that each KeyStore will store one share of the key. This share will not be sufficient alone to decrypt the published messages.
- *Malicious* subscriber: we suppose the absence of any mechanism to control or prevent any subscriber from subscribing to any topic. Therefore, each subscriber will try to subscribe to all topics in the system and try to extract contents of published messages on these topics even without valid credentials. Also, we assume that the malicious subscribers may try to delegate the trust to other subscribers (this threat was already discussed in III-A).
- *Colluding* components: we assume that malicious subscriber can collude with malicious brokers to try to get all the transmitted messages. Also, we assume that compromised KeyStore(s) can collude with each other and with any subscribers or broker.
- *External malicious* attacker: we assume the adversary model from [31] where attackers are able to access, modify, initiate all the message of the protocol.

The mitigation of these threats is discussed in detail in the analysis section.

V. SEEMQTT Protocol

Our proposed protocol has five phases namely the setup phase (§ V-A), symmetric master key distribution (§ V-B), topic key distribution (§ V-C), encrypted message transmission (§ V-D), and key retrieval and message decryption (§ V-D). In the remainder of this section, we explain each phase in more detail. Fig. 5 shows the exchanged messages during every phase of the protocol. Also, Table I contains the notations and symbols used throughout the proposed protocol. We use Alice&Bob notation [33] to describe our security protocol.

A. Phase 0: Setup Phase

This phase occurs *once*. Throughout this phase:

- The PKG creates the PKG's master key MK_{PKG} and the public parameters $params$ by running the algorithm `Setup`. The $params$ will be shared with the broker which will share them later with every publisher.

$$PKG : (MK_{PKG}, params) = Setup(1^{\lambda})$$
$$PKG \rightarrow Br : params$$

- Each one of the KeyStores registers with the PKG to get a private key $prKID_{ks_i}$ derived from its identity ID_{ks_i}. The PKG shares this private keys using a secure channel. Also, each KeyStore subscribes to a predefined topic, called MK, to receive messages during the next two phases.

TABLE I: An overview of the notation used in this work.

Notation	Description
Pub	Publisher node
Sub_j	Subscriber node with the index j
tp_i	Topic with the index i
Br	Broker
PKG	Private Key Generator
Ks_i	KeyStore with index i
$\mathcal{KS}$	the group of all KeyStores
puK_A	Public key of the node A
prK_A	Private key of the node A
ID_A	Identity of the node A. This identity is used as a public key for the IBE encryption
$prKID_A$	Private key of the node A. This key is used as a private key for the IBE decryption. This key is different from prK_A
$symK_{A-B}$	Symmetric key between nodes A and B
$symK_{tp_i}$	A symmetric key used to encrypt all published messages under the topic tp_i
$C = E(M, K)$	Symmetric encryption of a message M using a key K; the output is a ciphertext C
$M = D(C, K)$	Symmetric decryption of a ciphertext C ($C = E(M, K)$) using the key K, the output is the message M
$S(M, K)$	Sign a message M using a key K. If the key is a symmetric key, the algorithm S calculates the message authentication code of the message M and outputs a tag. If the key is a private key, the algorithm S performs a digital signature on the message M and outputs a signature σ
$V(M, tag, K)$	Verify whether the output of $S(M, K)$ is equal to the tag to return $accept$. Otherwise, it returns $reject$. The type of the key used in this algorithm is determined by the nature of the key used in $S(M, K)$
$E_{EtM}(M, K)$	Encrypt-then-MAC (EtM) [32] a message M using a key K. It is critical to use different keys for the encryption and MAC operations (i.e., K_{enc}, K_{mac}). E_{EtM} is defined as follow: $E_{EtM}(M, (K_{enc}, K_{mac}))) := C \xleftarrow{R} E(M, K_{enc}), tag \xleftarrow{R} S(C, K_{mac})\ return\ (C, tag)$. For the sake of simplicity, we state only one key for both E_{EtM} and D_{EtM}
$D_{EtM}(C, K)$	Verify the integrity of a cipher C and decrypt it using a key K. D_{EtM} is defined as next: $D_{EtM}((C, tag), (K_{enc}, K_{mac}))) :=$ if $V(C, tag, K_{mac}) = accept$, then output $D(C, K_{enc})$, otherwise output $reject$. For the sake of simplicity, we state only C as an input for D_{EtM} instead of (C, tag).
$CR^{puK_A}_{puK_B}$	Credential which shows that principle A trusts principle B to perform one or more operations under certain condition(s)
$H(M)$	Hashing the message M using a hash function H
nc_i	A non-repeated random number or nonce
$A \rightarrow B : M$	Read as A sends a message M to B over an insecure channel [33]
$A \xrightarrow{s} B : M$	Read as A sends a message M to B over a secure channel.
$A : X$	Read as A performs X [34]
$\|\|$	Concatenation operation

$$
\begin{aligned}
&For\ all\ ks_i \in \mathcal{KS}\\
&\quad ks_i \rightarrow PKG : ID_{ks_i}\\
&\quad PKG : prKID_{ks_i} = Extract(MK_{PKG}, ID_{Ks_i})\\
&\quad PKG \xrightarrow{s} ks_i : prKID_{ks_i}\\
&\quad ks_i : SUBSCRIBE(MK)
\end{aligned}
$$

- Each subscriber (Sub_j) needs to have particular credentials from TA to authorize it to obtain the key that was used to encrypt the published messages for a certain topic. Also, each Sub_j registers its interest to receive messages on the particular topic, let us say tp_i (The Pub will use this topic to publish messages), by sending a `SUBSCRIBE` message to the broker.

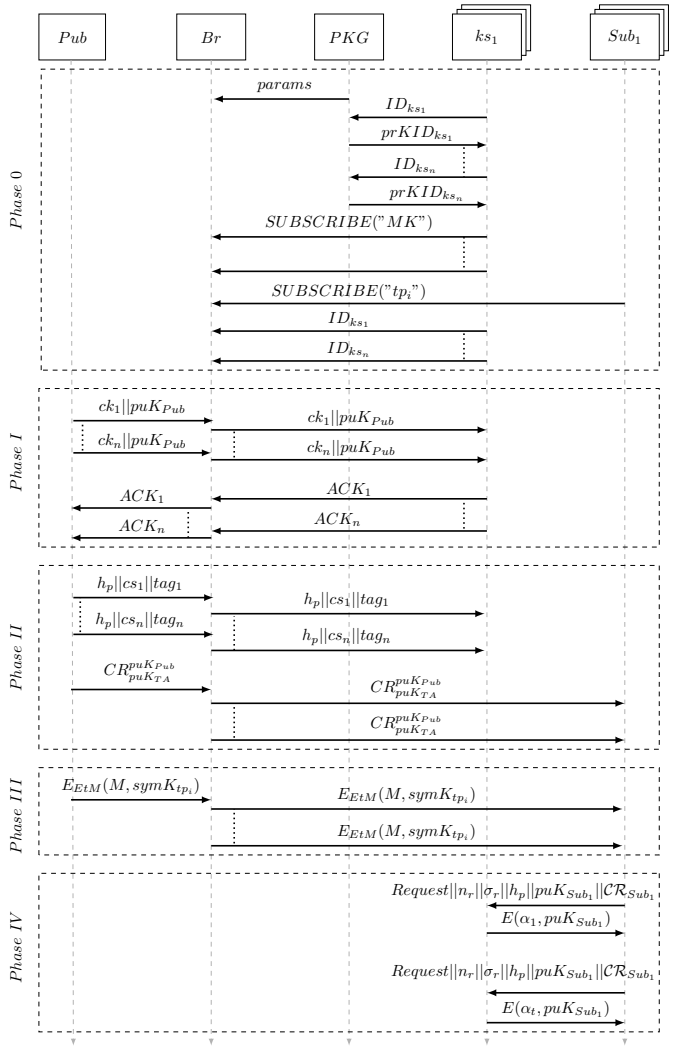


Fig. 5: The five phases of the SEEMQTT protocol.

$$Sub_j \rightarrow Br : SUBSCRIBE(tp_i)$$

B. Phase I: Symmetric Master Key Distribution

This phase contains two sub-phases: Phase II-1 that is used to setup a symmetric master key between the publisher and every KeyStore ($symK_{Pub-ks_i}$), and Phase II-2 that is used to receive an acknowledgment from every KeyStore to confirm the receipt of a master key.

1) Phase I-1: Whenever Pub receives the information about the existing $\mathcal{KS}$, it creates a set of symmetric master keys ($symK_{Pub-ks_1}, \ldots, symK_{Pub-ks_n}$) using algorithm `GenKey()` and stores these keys into a tuple $\langle ID_{ks_i}, symK_{Pub-ks_i}, nc_i, expT \rangle$. Each tuple includes a KeyStore ID (ID_{ks_i}), a symmetric master key between the

publisher and that KeyStore ($symK_{Pub-ks_i}$), a nonce nc_i, and an expiration time $expT$ of the symmetric master key. The Pub must renew that symmetric master key before reaching its $expT$. To share that key with the relevant KeyStore, Pub uses ID_{ks_i} to encrypt (using IBE) the generated symmetric master key together with the nonce nc_i and the hash value of its public key ($h_p = H(puK_{Pub})$). Then, the Pub concatenates the computed cipher message (ck_i) with its public key and publishes them to the broker.

$$
\begin{aligned}
&For\ all\ ks_i \in \mathcal{KS}\\
&\quad Pub : symK_{Pub-ks_i} = GenKey()\\
&\quad Pub : m = symK_{Pub-ks_i}||nc_i||h_p\\
&\quad Pub : ck_i = Encrypt(m, params, ID_{ks_i})\\
&\quad Pub \rightarrow Br : ck_i \ ||\ puK_{Pub}
\end{aligned}
$$

2) Phase I-2: Upon receiving these messages, the broker forwards them to KeyStores. Each ks_i decrypts the received message using its private key $prKID_{ks_i}$ and extracts the symmetric master key, nonce, and the hash value of publisher's public key. Each KeyStore uses a tuple to store these information. Each tuple $\langle H(puK_{Pub}), RoT, symK_{Pub-ks_i}, tp_i, symK_{tp_i}\rangle$ contains the hash value of the publisher's public key $H(puK_{Pub})$ which serves as an identity for that publisher, the root of trust which will be filled by the publisher's public key, the symmetric master key, a topic identifier (tp_i), and a topic key to secure messages published on this topic ($symK_{tp_i}$). Note that each publisher can have multiple message keys, one for each topic. However, it needs only one symmetric master key with each KeyStore to exchange all these topic keys. In this stage of the protocol, both tp_i and $symK_{tp_i}$ are empty:

$$
\begin{aligned}
&For\ all\ ks_i \in \mathcal{KS}\\
&\quad Br \rightarrow ks_i : ck_i \ ||\ pubK_{Pub}\\
&\quad ks_i : \langle symK_{Pub-ks_i}, nc_i', h_p\rangle = Decrypt(ck_i, prKID_{ks_i})
\end{aligned}
$$

After decrypting the received message, each ks_i checks whether h_p is equal to $H(pubK_{Pub})$. Then, each ks_i needs to acknowledge the Phase I-1's success to Pub and confirm that the symmetric master key was linked to Pub's public key. To do so, each ks_i forms a message by XORing the received nonce nc_i' with the hash value of puK_{Pub}. This message is encrypted using the received $symK_{Pub-ks_i}$ and sent to Br, which forwards it to Pub who is waiting for this acknowledgment.

$$
\begin{aligned}
&For\ all\ ks_i \in \mathcal{KS}\\
&\quad ks_i : ACK_i = E(nc_i' \oplus h_p, symK_{Pub-ks_i})\\
&\quad ks_i \rightarrow Br : ACK_i\\
&\quad Br \rightarrow Pub : ACK_i
\end{aligned}
$$

Pub uses the master symmetric key linked to each KeyStore to decrypt the received acknowledgments ACK_i. Then, it verifies whether the received nonce is equal to the one that was shared with the KeyStore during Phase I by using the `Compare` algorithm ($\{1, 0\} \leftarrow$ `Compare`(a, b, c): it compares whether $a \oplus b == c$ to return 1. Otherwise, it returns 0):

$$
Pub : \sum_{i=1}^{n} Compare(D(ACK_i, symK_{Pub-ks_i}), h_p, nc_i) \stackrel{?}{=} n
$$

This phase ends whenever Pub has received acknowledgments from every KeyStore and has verified received nonces successfully. Otherwise, the protocol tries to share a new set of master symmetric keys with KeyStores.

C. Phase II: Topic Key Distribution

After setting up a symmetric master key between Pub and every ks_i, Pub use the algorithm `GenKey` to generate a topic key $symK_{tp_i}$. This key is used to encrypt all published messages under the topic tp_i. The Pub adopts secret sharing scheme (see subsection II-B1) to share this key with the n KeyStores. To achieve that, the Pub uses the algorithm `GenShare()` to create n shares $(\alpha_1, \ldots, \alpha_n)$ by specifying the minimum number of shares t that needs to be used to recover the topic key ($symK_{tp_i}$).

$$
\begin{aligned}
&Pub : symK_{tp_i} = GenKey()\\
&Pub : (\alpha_1, \ldots, \alpha_n) = GenShare(n, t, symK_{tp_i})
\end{aligned}
$$

For each one of the n KeyStores, Pub uses the $symK_{Pub-ks_i}$ to encrypt and authenticate a message that contains one share and the hash value of topic tp_i. The $H(tp_i)$ will be used as an identifier for that topic.[6] Then, Pub shares the hash value of its public key h_p, the encrypted message cs_i, and the authentication code tag_i with the relevant KeyStores via the broker.

$$
\begin{aligned}
&For\ all\ ks_i \in \mathcal{KS}\\
&\quad Pub : \langle cs_i, tag_i\rangle = E_{EtM}(\alpha_i||H(tp_i), symK_{Pub-ks_i})\\
&\quad Pub \rightarrow Br : h_p||cs_i||tag_i
\end{aligned}
$$

To control who is able to get this key, Pub uses the algorithm `GenCredential` to creates a credential $CR^{puK_{Pub}}_{puK_{TA}}$. This credential authorizes a TA (directly) and all subscribers that fulfill certain conditions $Conds$ and are trusted by TA (indirectly) to retrieve n shares from the n KeyStores. $Conds$ specify the topic identity $H(tp_i)$ and for how long this credential will stay valid.

$$
\begin{aligned}
&Pub : GenCredential(puK_{Pub}, puK_{TA}, Conds, prK_{Pub})\\
&Pub \rightarrow Br : CR^{puK_{Pub}}_{puK_{TA}}
\end{aligned}
$$

The broker forwards the received messages from Pub to every KeyStore. Using the received h_p, each ks_i can determine which symmetric master key should be used to verify and decrypt the message. If the verification succeeds, the ks_i will check the existence of the $H(tp_i)$ to update the topic key, otherwise, it will create a new tuple to store the share in the field of the topic key.

[6]Using the same key for encryption and MAC is not secure. Therefore, it is very critical to ensure the use of two different keys (one for the encryption and the other for the MAC) that are derived from the symmetric master key. In our implementation, we use AES with the GCM mode to ensure that.

$$For\ all\ ks_i \in \mathcal{KS}$$
$$Br \rightarrow ks_i : h_p||cs_i||tag_i$$
$$ks_i : \alpha_i||H(tp_i) = D_{EtM}(cs_i, tag_i, symK_{Pub-ks_i})$$

Also, the broker Br forwards the credential $CR^{puK_{Pub}}_{puK_{TA}}$ to every member of $\mathcal{SUB}_{tp_i}$.

$$For\ all\ Sub_j \in \mathcal{SUB}_{tp_i}$$
$$Br \rightarrow Sub_j : CR^{puK_{Pub}}_{puK_{TA}}$$

D. Phase III: Encrypted Message Transmission

The publisher aims to guarantee the confidentiality and integrity of published messages. Therefore, Pub uses $symK_{tp_i}$ to encrypt and authenticate every published message under the tp_i.

$$Pub \rightarrow Br : E_{EtM}(msg, symK_{tp_i})$$

The broker Br forwards the messages to every member of $\mathcal{SUB}_{tp_i}$.

$$For\ all\ Sub_j \in \mathcal{SUB}_{tp_i}$$
$$Br \rightarrow Sub_j : E_{EtM}(msg, symK_{tp_i})$$

E. Phase IV: Key Retrieval and Message Decryption

After receiving the encrypted messages and credential, every subscriber needs to decrypt these messages. To do so, each Sub_j needs to communicate with the n KeyStores to retrieve at least t secret shares. Then, each subscriber uses the algorithm `ComShare` to reconstruct the $symK_{tp_i}$ using those t collected shares. It is important to note that this communication does not need to go through the Br.

To get the shares, each Sub_j uses its private key prK_{Sub_j} to sign a $Request$ with a nonce nc_r (the nonce is used to prevent replay attacks) as follows: $\sigma_r = S(Request||nc_r, prK_{Sub_j})$. Then, it sends the next values to t KeyStores (at least): $Request$, the signature σ_r, its public key puK_{Sub_i}, the $H(puK_{Pub})$ (the puK_{Pub} can be extracted from the received credential in Phase II), a list of all credentials $\mathcal{CR}_{Sub_i}$ ($m = |\mathcal{CR}_{Sub_i}|$) that prove that TA trusts the Sub_j and it fulfills all conditions stated in $CR^{puK_{Pub}}_{puK_{TA}}$ (note that the received credential will be included in the credential list, i.e., $CR^{puK_{Pub}}_{puK_{TA}} \in \mathcal{CR}_{Sub_i}$).

$$For\ t\ ks_i \in \mathcal{KS}$$
$$Sub_j \rightarrow ks_i : Request||nc_r||\sigma_r||h_p||puK_{Sub_j}||\mathcal{CR}_{Sub_i}$$

Each ks_i verifies the signature of $Request||nc_r$. Then, it uses the algorithm `Evaluate` which tries to find a path of trust that links the $pubK_{Sub_j}$ with the $pubK_{Pub}$ based on the credentials provided by Sub_j. Note that, the KeyStore uses the received $H(puK_{Pub})$ to determine the RoT's public key. If the algorithm finds such a trusted path, the ks_i encrypts the secret share α_i using puK_{Sub_i} and sends it back to Sub_i:[7]

[7]There is no need for encrypting the share if the communication between the ks_i and Sub_j goes over a secure link.

$$For\ t\ ks_i \in \mathcal{KS}$$
$$ks_i : V(Request||nc_r, \sigma_r, puK_{Sub_j}) \stackrel{?}{=} accept$$
$$ks_i : Evaluate(puK_{Pub}, \mathcal{CR}_{Sub_i}, puK_{Sub_j}, Request) \stackrel{?}{=} 1$$
$$ks_i \rightarrow Sub_j : E(\alpha_i, puK_{Sub_j})$$

The Sub_j decrypts the received message using its private key and extracts α_i.

$$For\ (t)\ ks_i \in \mathcal{KS}$$
$$ks_i : \alpha_i = D(E(\alpha_i, puK_{Sub_j}), prK_{Sub_j})$$

Then, it uses the algorithm `ComShare` to reconstruct the $symK_{tp_i}$ using the received t shares. Later, it uses this key to decrypt the message and check its integrity.

$$Sub_j : symK_{tp_i} = ComShare(\underbrace{\alpha_1, \ldots \alpha_t}_{t \leq n})$$
$$Sub_j : msg = D_{EtM}(E_{EtM}(msg, symK_{tp_i}), symK_{tp_i})$$

F. Remarks

- Pre-shared Secret: One of the main advantages of our proposed protocol is that publishers do not need to be registered with any of the trusted parties that we have (i.e., TA or PKG). All the information that publishers need can be collected from the brokers without the need to contact these parties directly. These information includes puK_{TA}, $params$, and the KeyStores' IDs. One way to avoid exchanging the KeyStores' IDs with every publisher is by letting these identities to follow an uniform naming convention (e.g., "city-name_KeyStore_id"). In this case, the publisher needs only to know the number of the existing KeyStores.
- Malicious Publishers: We assume only honest publishers will participate in our system. As a result, every publisher can participate in the protocol. It is possible to adapt the protocol so that only authorized publishers are allowed to participate. By implementing the trust delegation mechanism, only authorized publishers can store shares in the KeyStores. This would require every publisher to have a credential from a TA that is trusted by the KeyStores directly or indirectly. During Phase I-1, the publisher sends all credentials to the KeyStores. These credentials will be evaluated by each KeyStore to verify that the publisher is legitimate and capable of storing shares.
- Renewal of the Symmetric Master Keys: One of the challenges that any protocol faces is deciding when to change or renew a symmetric key that was used for a certain amount of time. Changing the symmetric keys is necessary since the security of a key is weakened if a key is used over an extended period. In order to renew a symmetric master key between the publisher and the KeyStores, we could simply run the initial key exchange mechanism again using IBE. However, this would pose a high computational effort. Instead, our system uses the previous symmetric master key to encrypt a new one.

TABLE II: The frequency of protocol's phases.

Phase	Phase 0	Phase I	Phase II	Phase III	Phase IV
Frequency	Once	Every P_1 or adding new ks_i	Every P_2	Per message	Every P_2 and Per message

- Adding or Removing KeyStores: adding new KeyStores requires setting up IBE public and private keys for these KeyStores. On the other hand, adding KeyStores after performing Phase II (topic key distribution) does not require re-performing that phase. Subscribers can still decrypt the encrypted messages using the topic key that was already shared with the existing KeyStores. Deleting x KeyStores after that phase requires re-transmitting a new topic key only if $x > n - t$.
- Phase Frequencies: to understand the introduced overhead by the protocol, it is critical to know how frequently each of the protocol's phases is performed. As shown in TABLE I, Phase 0 occurs once. As we have discussed before, $symK_{Pub-ks_i}$ must not be used for a long time, therefore, Phase I must be performed each P_1 period of time. Also, this phase is performed after the addition of a new KeyStore. Similar to $symK_{Pub-ks_i}$, $symK_{tp_i}$ must not be used for a long time. Therefore, Phase II should also be performed periodically (i.e., every P_2 period). The Phase III occurs every time the publisher publishes a message. The decryption of the message in Phase IV is performed every time an encrypted message is received, while getting the shares from the KeyStores and constructing the $symK_{tp_i}$ occur every P_2 period.
- Trust Revocation: The KeyNote trust management natively supports time-based revocation [35]. The Authorizer can determine the validity of the credentials by encoding the validity period in the "Conditions" field of the credential. After that date, the credential will not be valid anymore (i.e., revoked). In addition, revocation can also be handled independently from the KeyNote trust management system by adopting other schemas such as Certificate Revocation Lists (CRLs). Before evaluating the credentials, every KeyStore checks the CRLs, which contain the revoked Authorizers, to ensure the validity of the requester credentials. Implementing the revocation using CRLs is beyond the scope of the current work.

VI. Implementation

We have implemented an open-source Arduino platform-specific C++ library, called SEEMQTT[8], that implements the publisher in the SEEMQTT protocol. The library is derived from the base class PubSubClient, which provides the basic MQTT client functionality[9]. To implement the IBE, we have used the implementation of the Boneh and Franklin IBE (BF-IBE) algorithm [28]. We chose the Pairing-Based Cryptography (PBC) library to accomplish the underlying pairing operations in BF-IBE scheme[10]. The PBC library itself is built on

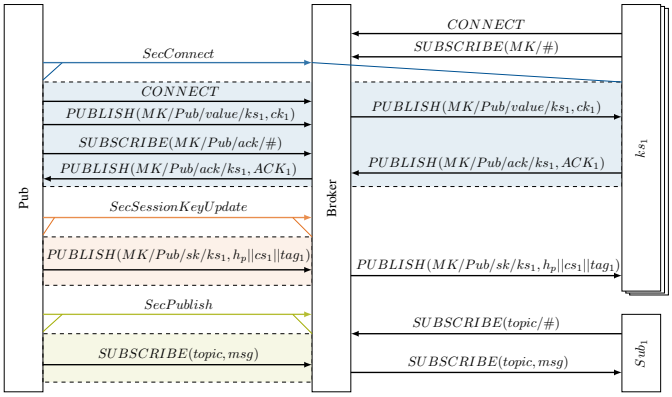


Fig. 6: The three main calls of SEEMQTT library.

the Multiple Precision Arithmetic (GMP) library[11]. Both PBC and GMP libraries were cross-compiled on a Linux machine for the Arduino ESP32 board and were provided as static libraries (i.e., libpbcesp32.a and libgmpesp32.a). we have implemented our own Arduino-based library that implements Shamir's secret sharing scheme based on an existing Linux-based implementation [36]. Other cryptographic operations were implemented using the Mbed TLS library[12].

It is important to note that our implementation does not require any changes to the broker implementation, making it compatible with other brokers' implementations. Fig. 6 shows the main three functions that the SEEMQTT library provides:

- `SecConnect`: This function is used by the publisher to connect to the broker and set up the symmetric master key with every KeyStore. This function encapsulates the `CONNECT` control packet to connect with the broker, and the `Encrypt` algorithm to encrypt a master key for every KeyStore. To exchange the encrypted symmetric master key with all Keystrokes, the publisher publishes it under the predefined topic "MK/Pub/value/ID_{ks_i}". Remember that each KeyStore has already subscribed to that topic. To handle the acknowledgments, the publisher subscribes to another predefined topic "MK/Pub/ack/#". Based on the KeyStore's ID, the publisher can validate the acknowledgment.
- `SecSessionKeyUpdate`: this function is used to set up a topic key (if it was called for the first time) or update an existing topic key. The function uses the algorithm `GenShare` to create the secret shares from the generated topic key. The publisher uses a third predefined topic, "MK/Pub/sk/ID_{ks_i}", to exchange these shares with the KeyStores.
- `SecPublish`: the publisher uses this function to send authenticated encrypted messages. The AES cipher with

[8]https://github.com/tum-esi/SEEMQTT
[9]https://www.arduino.cc/reference/en/libraries/pubsubclient/
[10]https://crypto.stanford.edu/pbc/
[11]https://gmplib.org/
[12]https://tls.mbed.org

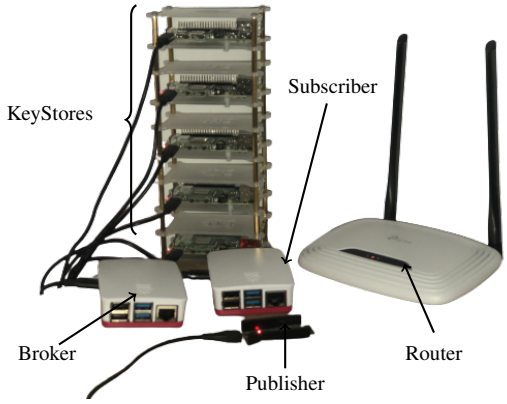


Fig. 7: The testbed that was used to evaluate SEEMQTT.

Galois/Counter Mode (AES-GCM) [37] is used to encrypt the shares before publishing them to the broker.

We have implemented the KeyStore functionality using the C programming languages and based it on the Eclipse Mosquitto client library and the KeyNote trust management library. All cryptographic operations were implemented using the OpenSSL library.

VII. Experiments and Analysis

In this section, we present our testbed that was used to carry out our experiments (see subsection VII-A). Also, we present a comprehensive performance analysis that covers all phases of the proposed protocol (see subsection VII-B). Finally, we provide an informal security analysis of our proposed protocol (see subsection VII-C).

A. Testbed

We have deployed and evaluated our implementation using the testbed that is shown in Fig. 7. The testbed includes one Espressif ESP32-WROOM-32D development board that is used as a publisher. The goal behind using this board is to show that our proposed system is practical for resource constrained embedded platforms. The ESP32 board is equipped with an Xtensa dual-core 32-bit LX6 microprocessor with $240\,\mathrm{MHz}$ clock frequency and integrated Wi-Fi functionalities. Additionally, the testbed includes seven Raspberry Pi 4 Model B (RPI4). Each RPI4 includes a Broadcom BCM2711 which is an SoC based on a 64-bit ARM-Cortex A72 quad core running at $1.5\,\mathrm{GHz}$ and runs 32-bit Raspberry Pi OS.[13] Five of these Raspberry Pis were used as KeyStores, one was used as a broker, and the last one was used as a subscriber. All these devices were interconnected wirelessly using a TP-Link TL-WR820N router. The network was isolated from the Internet

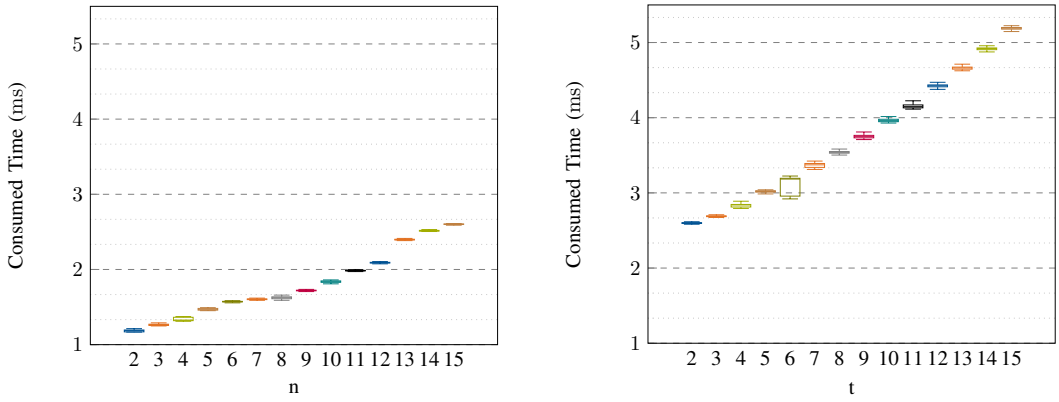
B. Performance Analysis

The evaluation aims to measure the overhead of using Shamir's secret sharing scheme and IBE on the publisher.Furthermore, it aims to measure the overhead of every phase of the proposed protocol.

1) Shamir's Secret Sharing Scheme: We started the evaluation by looking at the overhead of using Shamir's secret sharing scheme to understand the effect of choosing the number of KeyStores n and the threshold t. Since the publisher performs only the $\mathrm{GenShare}$ algorithm during our proposed protocol, we focused only on the evaluation of that algorithm here (Note that evaluating the `ComShare` algorithm using the ESP32 board is presented in the appendix A.). To evaluate the scheme, we measured the consumed time to perform the `GenShare` algorithm in order to share the topic key $symK_{tp_i}$ (the size of this key is $16\,\mathrm{B}$) using different combinations of n and t. The first test was to evaluate the effect of using different number of KeyStores (i.e., n) while using the same value of t. To conduct that, we measured the consumed time when $n = 2, 3, \ldots, 15$ and $t = 2$. The result of this test is presented in Fig. 8a. The result shows that the consumed time increases slightly with the increase of n. In order to study the effect of increasing the value t for the same value of n, we run another test by using $n = 15$ and measured the time for every value of t between 2 and 15. The result of this measurement is shown in Fig. 8b. The result shows that the consumed time increases also by the increase of t.

Besides its effects on the consumed time of generating and combining the secret, the value of t significantly impacts the system's security and availability. Although choosing a small value of t (compared to n) introduces a small overhead, it weakens the system security: if t is relatively small, an attacker needs to compromise only a small number of KeyStores (at most t out of n) to re-combine the secret. On the other hand, choosing a large value of t, where n is large, makes restoring the key maliciously more difficult since the attacker needs to compromise a high number of KeyStores (better security). However, it risks the system's availability since combining the key requires the availability of a large number of KeyStores (at least t out of n). Furthermore, we have seen that using a large value of t introduces more overhead. Since using a large and small value of t is not perfect solution, it seems that using a median value of t (compared to n) could be a good option. In order to evaluate that, we measured consumed time of the `GenShare` algorithm when $n = 2, \ldots, 15$ while $t = n$ (high security, low availability), $t = \lfloor n/2 \rfloor + 1$ (the median value), and $t = 2$ (low security, high availability). The results obtained from this test are presented in Fig. 9. The results shows that for small value of n (i.e., $n \leq 5$), the difference between the consumed time is negligible, while it is not for the larger values of n. Fig. 9 shows that different combinations of n and t can consume the same time. For example, the figure shows that the consumed time to perform the `GenShare` algorithm using $n = 11$ and $t = 6$ is almost incidental to the cases of using $n = 15$ and $t = 2$ (High availability) and $n = 9$ and $t = 9$ (high security). It is important to note that we performed the measurements using $n \leq 15$ to reflect a realistic and practical scenario. However, for the sake of completeness, we evaluated the algorithm using larger values of n and presented the results in Fig. 10. The results show that even for larger values of n, the consumed time is still in the range of milliseconds.

[13] https://www.raspberrypi.com/software/operating-systems/



(a) $t = 2$ (b) $n = 15$

Fig. 8: The consumed time during the executing of `GenShare` using a secret of 16 B and different values of n and t.

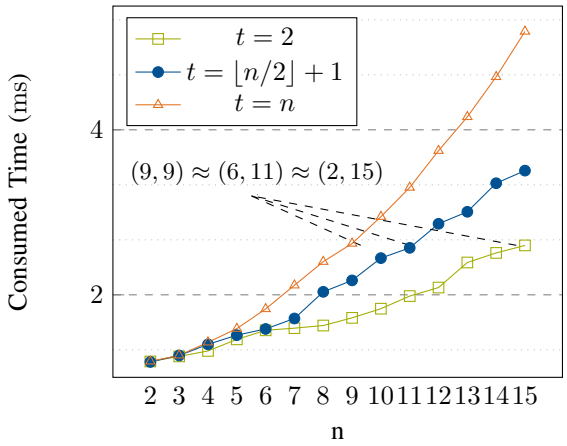


Fig. 9: Comparing the consumed time during the execution of `GenShare` using a 16 B key using Shamir's secret sharing when $n = 2, \ldots, 15$ and $t = 2, t = \lfloor n/2 \rfloor + 1, n$.

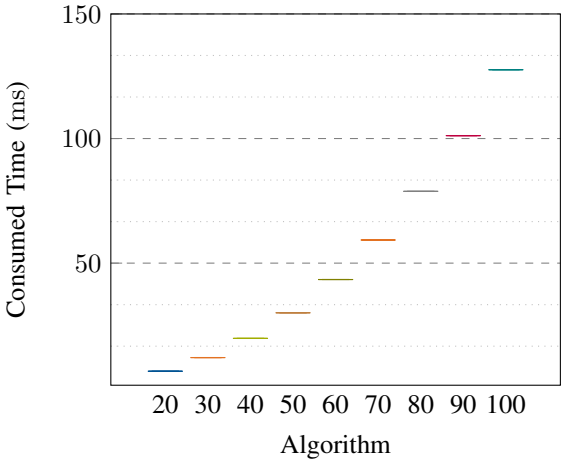


Fig. 10: The consumed time during the executing of `GenShare` using a secret of 16 B using larger values of n and $t = \lfloor n/2 \rfloor + 1$.

2) IBE vs TLS: Having a secure link between the publisher and every KeyStore using TLS can be argued as a better option than using IBE. The goal of this subsection is to compare both options based on the consumed time, the size of stored data, and the size of exchanged messages during the process of setting a secure link between the Pub and KeyStore using IBE and TLS.

For TLS, we measured the required time to set up a secure link using TLS v1.2 with the cipher suite $TLS\_ECDHE\_RSA\_AES\_256\_GCM\_SHA384$. During the handshake protocol, the publisher needs to verify the KeyStore certificate and the exchanged parameters to proceed with the setup of the secure session using Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) key exchange protocol [38]. Regarding IBE, we measure the required time for the publisher Pub to calculate the $g_{ID_{ks_1}}$ of the KeyStore (see § II-B2), use that to encrypt the session key, and receive the acknowledgment from the KeyStore (Remember that all the messages go through the broker in the case of IBE).

Fig. 11 presents the measured time using both schemes. The results show that setting up the session key using IBE consumes more time compared to the case of using TLS. Nevertheless, by looking closely at this time when IBE is used, we found that almost 70% of the measured time is resulting from the computation of $g_{ID_{ks_1}}$, while only 30% of that time is caused by the encryption of the session key and the communication process with the KeyStore (see the right part of Fig. 11). Since the calculation of $g_{ID_{ks_1}}$ is independent of the session key encryption, the publisher does not need to be connected to the KeyStore to perform that operation. Thus, the needed time for both nodes to be connected is very short (around 300 ms). On the other hand, using TLS requires that both the publisher and the KeyStore are connected during the entire duration of the protocol (i.e., around 1038 ms). However, the need for such a long period to contact with every single KeyStore can be quite challenging in the case of moving publishers.

Wireshark[14] was used to measure the size of the exchanged messages during the set up of the secure link using both schemes. Based on the data presented in TABLE III, it is

[14]https://www.wireshark.org/

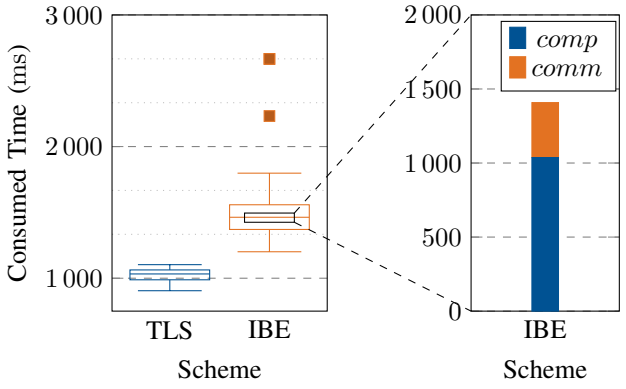


Fig. 11: The time required to set up a 32 B session key between the publisher and a KeyStore using TLS and IBE. The right sub-figure shows the communication (comm) and computation (comp) time of the median value while using IBE.

TABLE III: The exchanged and stored data when using IBE and TLS to set up a 32 B session key between a publisher and a KeyStore.

Scheme	Exchanged Messages (B)	Stored Data (B)
IBE	1061	453
TLS	3569	1346

apparent that the size of exchanged messages is three times larger when TLS was used. Similarly, the table shows that the size of data that the publisher needs to store is three times larger when TLS was used. This data includes the CA certificate in case of using TLS, while it includes the system parameters $params$ and the identity of the KeyStore (we used "keystore1@tum.com" as an ID for the KeyStore) in the case of IBE.

3) SEEMQTT Phases: In this subsection, we present the performance evaluation of every phase of the protocol.

a) Phase 0: The two primary operations of this phase are `Setup` algorithm to create the system public parameters $params$ and the `Extract` algorithm to set up the private key of every KeyStore. Fig. 12 presents the consumed time by these two algorithms. The measurement was performed by running these two algorithms on one of the RPI4. The measurement was repeated for 100 times for each algorithm. The results show that the algorithm `Setup` takes a longer time (almost 40%) compared to the time consumed by the `Extract` algorithm. It is important to note few things: 1) The `Extract` algorithm must be performed for every KeyStore in the system while the `Setup` algorithm runs once. The result shown in Fig. 12 represents the time that is needed to perform `Extract` for one KeyStore. 2) The measured time of `Extract` algorithm includes neither the time for receiving the ID of the KeyStore nor the time for sending the private key. 3) Finally, these two algorithms are usually performed only once during the system setup.

Since These two algorithm were implemented as part of the SEEMQTTlibrary, we have measured their performance when they run over the ESP32 platform too. The results of that evaluation are presented in the appendix A.

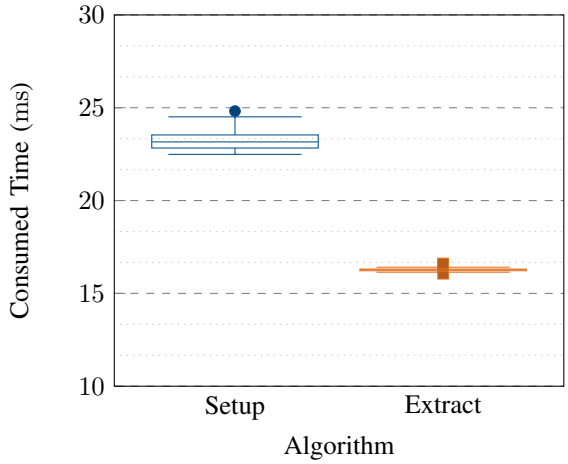


Fig. 12: The performance evaluation of `Setup` and `Extract` algorithms when they run on RPI4.

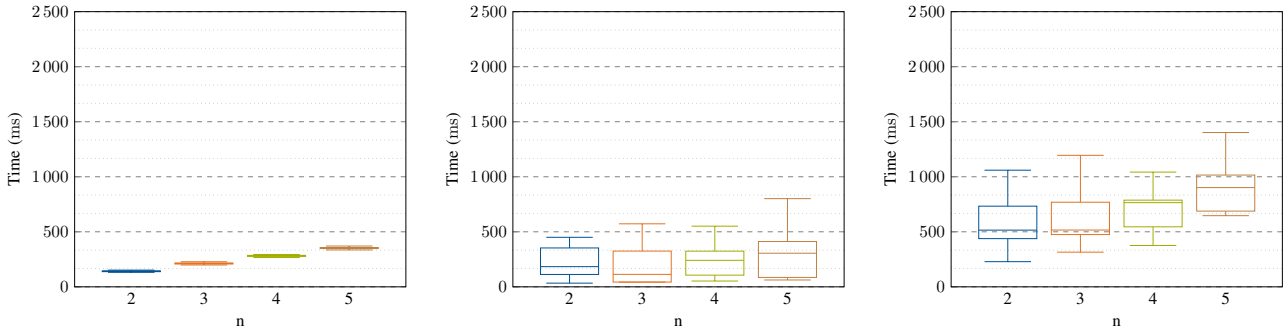
b) Phase I: The time consumed by Phase I can be divided into three sub-times as defined in (5):

$$T_{PhaseI} := T^{Pub}_{Connect} + \underbrace{\sum_{i=1}^{n}(T^{Pub}_{Encrypt} + T^{Pub}_{PublishK} + T^{Pub}_{Other})}_{T_{PhaseI-1}} + \underbrace{\max_{1\leq i\leq n}(T^{net+ks_i}_{Pub\leftrightarrow ks_i} + T^{Pub}_{Compare})}_{T_{PhaseI-2}} \tag{5}$$

Where $T^{Pub}_{Connect}$ represents the time for the publisher to connect to the broker, $T_{PhaseI-1}$ represents the time to perform Phase I-1 and $T_{PhaseI-2}$ represents the time to perform Phase I-2. We measured the time consumed by Phase I when different numbers of KeyStores are used (i.e., $n = 2, 3, 4, 5$).

The $T_{PhaseI-1}$ is defined as the sum of the time to encrypt one symmetric master key ($T^{Pub}_{Encrypt}$) using IBE-based encryption, the time to publish the encrypted key ($T^{Pub}_{PublishK}$), and the time for other operations (T^{Pub}_{Other}) (e.g., generating the master symmetric key and nonce, etc.) for the n KeyStores. It is critical to note that all these times are independent of the number of KeyStores. The same applies to $T^{Pub}_{Connect}$. It is also important to distinguish between the time to encrypt ($T^{Pub}_{Encrypt}$) using IBE encryption and the time to calculate $g_{ID_{ks}}$ ($T^{Pub}_{q_{ID}}$) of every KeyStore. As we have mentioned before, this operation happens once and should not be considered as constant part of the encryption. Therefore, we have not included this time in the calculation of $T_{PhaseI-1}$. However, this time was measured and included in TABLE IV besides all other operations in Phase I that are independent of the number of KeyStores .

The $T_{PhaseI-2}$ is calculated as the maximum time of $T^{net+ks_i}_{Pub\leftrightarrow ks_i}$ and $T^{Pub}_{Compare}$ of the n KeyStores, where $T^{net+ks_i}_{Pub\leftrightarrow ks_i}$ represents the period between publishing the encrypted symmetric key to the n^{th} KeyStore and receiving the acknowledgment sent by ks_i. This period includes the time required to transmit the encrypted symmetric key over the broker, decrypt it at the KeyStore ks_i, and send the acknowledgment back to



(a) $T_{PhaseI-1}$ (b) $T_{PhaseI-2}$ (c) T_{PhaseI}

Fig. 13: The consumed time during (a) Phase I-1, (b) Phase I-2, and (c) Phase I in total when different numbers of KeyStores n were used.

TABLE IV: Time performance of the main operations in Phase I that are independent of the number of KeyStores n. $Q1$ represents the first quartile and $Q3$ represents the third quartile.

Sub-Phase	**Operation**	**Time** (ms)		
		Q1	Median	Q3
	$T^{Pub}_{Connect}$	70.86	205.38	235.20
	$T^{Pub}_{qID_{ks}}$	1034.1	1037.12	1040.08
$Phase\ I-1$	$T^{Pub}_{Encrypt}$	65.08	67.03	68.35
	$T^{Pub}_{PublishK}$	1.20	1.70	1.76
$Phase\ I-2$	$T^{Pub}_{Compare}$	0.04	5.00	5.10

the publisher. $T^{Pub}_{Compare}$ represents the time required to decrypt the acknowledged sent by ks_i and compare the received nonce with the stored one.

Fig. 13a presents the consumed time by Phase I-1 when different numbers of KeyStores were used. The figure shows that the time of Phase I-1 increases linearly when n increases. Similarly, the time of Phase I-2 increases when n increases as shown in Fig. 13b. Fig. 13c shows the time of the entire Phase I. Similar to Phase I-1, the time increase when n increases. The variance in the interquartile range (IQR) of the results in Fig. 13b and Fig. 13c is resulting from the effect network latency introduced by $T^{Pub}_{Connect}$ and $T_{PhaseI-2}$.

c) Phase II: The consumed time of the Phase II is defined as in (6):

$$T_{PhaseII} := T^{Pub}_{GenCredential} + T^{Pub}_{PublishCre} + T^{Pub}_{GenShare} + \sum_{i=1}^{n}(T^{Pub}_{EncSS} + T^{Pub}_{PublishSS}) \quad (6)$$

where $T^{Pub}_{GenCredential}$ is the time required to create the credential, $T^{Pub}_{PublishCre}$ is the time required to publish the generated credential, $T^{Pub}_{GenShare}$ is the time required to perform `GenShare` algorithm, T^{Pub}_{EncSS} is the time to encrypt the secret share using AES-CBC with 128-bit key, $T^{Pub}_{PublishSS}$ is the time to publish the encrypted secret share, and n is the number of KeyStores $n = 2, 3, \ldots, 5$. Note that based on the result we obtained in § VII-B1, we decided to use $t = \lfloor n/2 \rfloor + 1$. All the mentioned times except $T^{Pub}_{GenShare}$ are independent of the number of KeyStores. TABLE V includes the measured time for all these operations.

TABLE V: Time performance of main operations in Phase II that are independent of the number of KeyStores n.$Q1$ represents the first quartile and $Q3$ represents the third quartile.

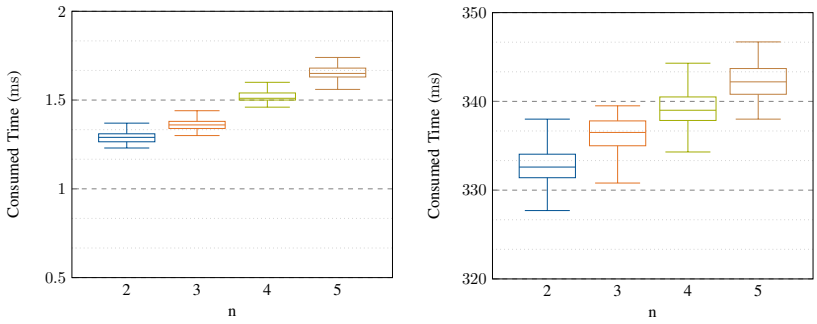
Operation	**Time** (ms)		
	Q1	Median	Q3
$T^{Pub}_{GenCredential}$	309.38	310.76	312.15
$T^{Pub}_{PublishCre}$	1.35	1.37	1.40
T^{Pub}_{EncSS}	0.27	0.28	0.28
$T^{Pub}_{PublishSS}$	0.97	0.98	1.02

Fig. 14a shows the consumed time during the execution of the `GenShare` algorithm using $n = 2, 3, \ldots, 5$ and $t = \lfloor n/2 \rfloor + 1$, while Fig. 14b shows the consumed time during the entire Phase II. The figure shows that the consumed time increases linearly by the increase of n. Based on the result presented in TABLE V and Fig. 14b, it is clear that $T^{Pub}_{GenCredential}$ presents almost 90% of the consumed time of this phase. This is not surprising if we remember that the publisher needs to digitally sign the credential. This operation is very time consuming compared to the other operations in this phase.

d) Phase III: The consumed time of Phase III is defined as in (7):

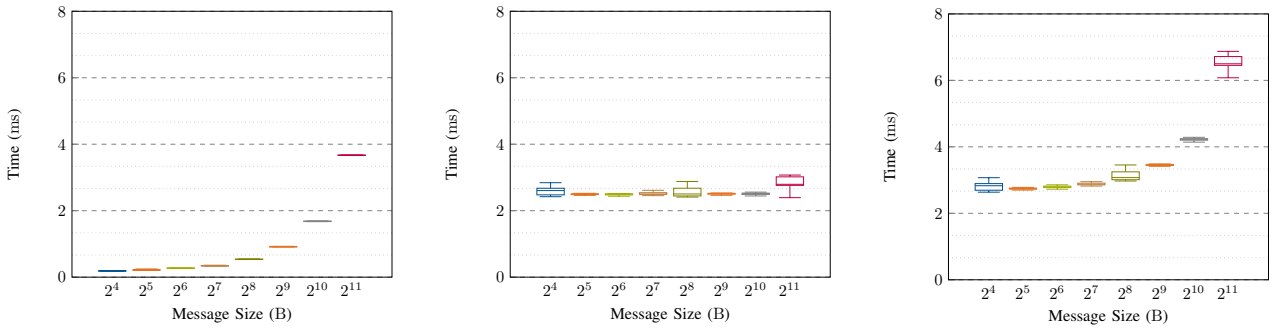
$$T_{PhaseIII} := T^{Pub}_{EtM} + T^{Pub}_{PublishMSG} \quad (7)$$

where T^{Pub}_{EtM} is the time required to perform authenticated encryption on a message using AES-GCM with 128-bit key and $T^{Pub}_{PublishMSG}$ is the required time to publish this encrypted message. Both times are independent of the number of the KeyStores, but they are related to the size of the transmitted message. To show that relation, we measure both T^{Pub}_{EtM} and



(a) $T^{Pub}_{GenShare}$ (b) $T_{PhaseII}$

Fig. 14: The consumed time (a) to perform GenShare for $t = \lfloor n/2 \rfloor + 1$, and (b) to perform the entire Phase II when different numbers of KeyStores n were used.



(a) T^{Pub}_{EtM} (b) $T^{Pub}_{PublishMSG}$ (c) $T_{Phase\ III}$

Fig. 15: The consumed time (a) to encrypt a message using AES-GCM with 128-bit key, (b) to publish the encrypted message, and (c) the sum of these both operations using different message sizes.

$T^{Pub}_{PublishMSG}$ for different message sizes as shown in the horizontal axes of Fig. 15.

Fig. 15a shows that T^{Pub}_{EtM} increases by the increase of the message size. On the other hand, Fig. 15b shows that $T^{Pub}_{PublishMSG}$ almost the same for the all message size less than 2^{11} B. We see an increase on the consumed time when the message size 2^{11} B was used. Fig. 15b shows the consumed time of the entire Phase III, which follows the same behavior of T^{Pub}_{EtM}.

e) Phase IV: The consumed time of Phase IV is defined as in (8):[15]

$$T_{PhaseIV} := \sum_{i=1}^{t}(T^{net}_{Request} + T^{ks_i}_{Evaluate} + T^{net}_{share}) + T^{Sub}_{ComShare} + T^{Sub}_{DEtM} \quad (8)$$

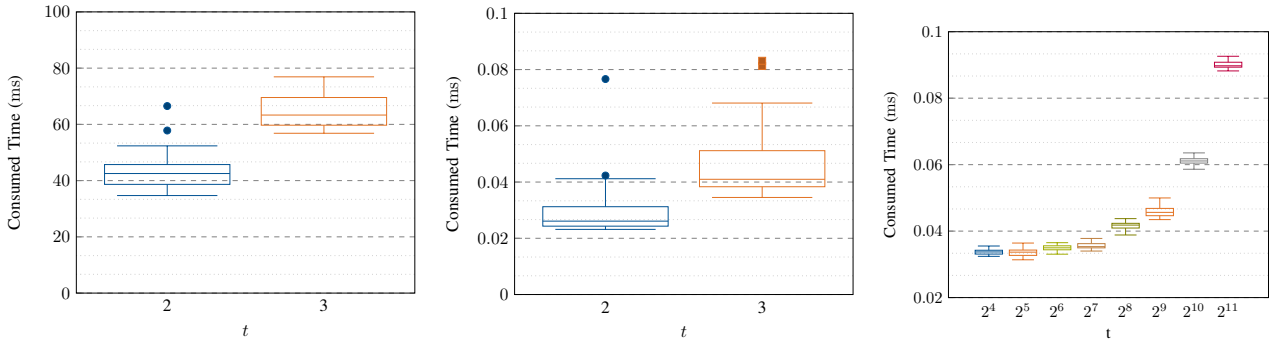
where $T^{net}_{Request}$ is the time required to prepare the request and send it to the KeyStore over the network, $T^{ks_i}_{Evaluate}$ is the time to evaluate the request based on the submitted credentials, T^{net}_{share} is the time required to receive the secret share, $T^{Sub}_{ComShare}$ is the time to perform the ComShare to combine the t received shares, T^{Sub}_{DEtM} is the time to decrypt the message using AES-GCM with 128-bit key, and t is the threshold number for reconstructing the secret from the collected shares. Fig. 16a shows the sum of $T^{net}_{Request}$, T^{net}_{share}, and $T^{ks_i}_{Evaluate}$ for $t = 2$ and 3. The figure shows that time increases when t is increases. We have measured the time $T^{ks_i}_{Evaluate}$ within every KeyStore and we found that it takes 12.8 ms (the median value) for the KeyStore to evaluate the request that includes two credentials. Fig. 16b presents the relation between $T^{Sub}_{ComShare}$ and t. It is important to note that the number of KeyStores n is not affecting the $T^{Sub}_{ComShare}$. The figure shows that time increase when t increases. Finlay, Fig. 16c presents the measured time to decrypt the received message using the re-constructed topic key. The results show that the decrypting time increases when the sizes of the messages increase (similar to the case of encryption as shown in Fig. 15a).

[15]if sending the requests and getting the secret shares from the t KeyStores were implemented to run in parallel, the $\sum_{i=1}^{t}()$ in (8) must be replaced by $\max_{i=1}^{t}()$. Also, only T^{Sub}_{DEtM} occurs every time during this phase, while the other operations occur when a new topic key is exchanged.

C. Informal Security Analysis

This section presents an informal security analysis for the phases of our proposed protocol and discusses how they are secure against various famous attacks.

a) Phase 0: Since The PKG and every KeyStore use a secret link to exchange the extracted $prKID_{ks_i}$, attackers can not obtain and change this key. The other information (i.e., $params$ and ID_{ks_i}) are public and do not need to be protected. It is critical to mention that compromising the private key of any TAs in a specific domain will allow



(a) $T^{net}_{Request} + T^{ks_i}_{Evaluate} + T^{net}_{share}$ (b) $T^{Sub}_{ComShare}$ (c) T^{Sub}_{DEtM}

Fig. 16: The consumed time by the different operations of Phase IV.

attackers to issue credentials to malicious intermediate TAs or malicious subscribers in that domain. Also, compromising the master private key of PKG will allow attackers to decrypt all messages.

b) Phase I: Within this phase, the $SymK_{Pub-ks_i}$ is encrypted using IBE. Attackers (i.e., external attackers, malicious brokers, malicious KeyStores, or malicious subscribers) can intercept the message ck_i. However, they can not decrypt it and obtain the $symK_{Pub-ks_i}$ without having the ks_i's private key. Storing this key securely within each KeyStore makes obtaining this key maliciously very difficult. Also, the use of non-repeated nonce nc_i makes it impossible for attackers to reuse previous captured ck_i. Although the puK_{Pub} is exchanged without any protection, the KeyStore can detect if attackers try to replace this key with another one by comparing the decrypted $H(puK_{Pub})$ and the calculated one from the received puK_{Pub}. Finally, any malicious changes to the acknowledgment message ACK_i are detectable by the publisher using the `Compare` algorithm.

c) Phase II: During this phase, every secret share α_i is encrypted with the symmetric master key $SymK_{Pub-ks_i}$. Attackers need to break the symmetric cipher to obtain the shares. However, using the AES-GCM with 128-bit key makes it not feasible for attackers to break the protocol by breaking this cipher since it is considered practically secure against all attacks. In addition, any changes to encrypted shares will be detected due to the use of the authenticated encryption algorithm. Finally, it is important to note that the security of this phase (and the entire proposed system) is based on the security of Shamir's secret sharing scheme. An attacker needs to collect at least t shares to reconstruct the $symK_{tp_i}$. Such a case is possible only if the attacker can compromise t KeyStores or if this t KeyStores decide to collude. Choosing t carefully and using secure storage systems to store the $symK_{Pub-ks_i}$ within every KeyStore makes the extraction of the t shares practically very difficult.

d) Phase III: Since the published messages are encrypted using the $symK_{tp_i}$, the security of this phase depends on the security of Phase II. Attackers can decrypt the message only if they manage to reconstruct $symK_{tp_i}$. Also, any change to the encrypted message will be detected by the authorized subscriber before the decryption due to the authenticated encryption algorithm.

e) Phase IV: Using the non-repeated nonce nc_r prevents attackers from submitting pre-exchanged requests to the KeyStores. Also, using the signature σ_r will ensure that no attacker can manipulate the request without being detected. The only way unauthorized subscribers can decrypt the message is by colluding with or compromising t KeyStores or retrieving the private key of the TA (or intermediate TAs) to sign a credential that authorizes access to the topic key from the KeyStores. Both cases seem practically very difficult.

f) Conclusion: Based on the previous discussion, we can say that our protocol is immune enough against the eavesdropping attack, replay attack, Man-In-The-Middle (MitM) attack, and the arbitrary behavior of the attackers.

VIII. Related Work

Securing Pub/Sub-based systems was a hot topic during the last few years, thus it was presented heavily [25], [39]–[41]. This section discusses some recent related researches that aim to secure the Pub/Sub model, in general, and the MQTT protocol in particular. We classified the related research into three main categories based on the main goal of every paper into solutions that focused on ensuring either the confidentiality, authorization, or both of the published message. Then, we compare our proposed solution to other existing solutions based on the requirements that we discussed in § I.

A. Confidentiality

One of the most common solutions to ensure data confidentiality between the publishers and subscribers is using TLS to secure the link between these nodes and the broker as was recommended by [23]. A certificate authority is needed to support the publisher and the subscriber to verify the broker's certificate. A key problem of this solution is the need to trust the broker. Lee et al. [42] proposed an extension to the TLS, called MQTLS, to provide E2E security between publishers and subscribers via an untrusted broker. In their solution, both publisher and every subscriber need to set up a TLS-based secure link with the broker. Then, they exchange an ephemeral Diffie Hellman (DH) public component through the broker to set up a symmetric key, the so-called one-time delivery

key. The delivery key is used to encrypt another key, the so-called payload encryption key. This key is used to encrypt the payload of the published messages. Like TLS, the proposed solution provides secure "one-to-one" communication only. I.e., each publisher needs to set up a one-time delivery key with *every* subscriber to exchange the payload key encryption. Consequently, the publisher needs to verify and authenticate every subscriber's certificate each time it connects to the broker or new subscribers subscribe to the publisher's topic. This makes the proposed solution inappropriate for mobile IoT devices with unstable connections and many subscribers. In addition, the publisher uses the subscriber's certificate to authenticate and authorize that subscriber. Thus, the solution does not ensure the decoupled authorization (i.e., authorization without violating decoupling principles).

Similar to the TLS-based solution, Peng et al. [43] proposed a secure Pub/Sub system. However, instead of using Public Key Infrastructure (PKI), they proposed the use of IBE. Their solution adopts a third trusted party to play the roles of PKG and topic address provider. All the subscribers, as well as the broker (gateway), need to register with this server to get a private key. The gateway itself is used as PKG for all publisher nodes. The main drawback of both solutions is that they do not provide publisher-subscriber end-to-end encryption. In [44], the authors proposed, in contrast to our untrusted broker requirement, a trusted security-enhanced broker that supports E2E secure group communication. The broker sets up a session with every registered client. This session key is used to encrypt a topic key generated by the broker for every topic. All the clients will share the same topic keys. Clients (publishers and subscribers) will use the topic key to encrypt and decrypt the messages published under that topic.

To avoid the involvement of the untrusted broker, many authors propose the use of a central trusted server to create a symmetric session key between publisher and subscriber and ensure end-end data confidentiality [18], [19], [45], [46]. In [46], Mektoubi et al. used a central server, called Certificate Authority (CA), to create a certificate for each registered node. Also, the CA creates a private key and certificate for each topic. Each subscriber uses a topic's certificate to communicate with the publisher and get the topic's private key. The authors mentioned that private keys and topic certificates are distributed manually to the publisher. This represents the main limitation to adopt this solution, especially if we consider the need to change this key periodically and automatically.

Nolan [18] presented a symmetric authenticated payload encryption scheme to ensure the confidentiality of published data over MQTT. In his scheme, Nolan used a central server to create a shared key between the publisher and the subscribers. All nodes need to be registered with the server to get a Pre-Shared Key (PSK) used to secure the exchange of a generated secret session key, which is used to encrypt and decrypt the published data. The author did not elaborate on how to exchange such a secret key with each node. Publisher does not have control on who can access the session key. Similarly, Dahlmanns et al. [19] proposed the use of a trusted server, so called key server, to support end-to-end security in publish/subscribe systems. A pairwise PSK must be shared between every node and the key server and used during the handshake protocol to set up fresh session keys. The main pitfall of these solutions is the need for the PSK between every publisher and the server. Additionally, the server is a single point of failure.

Borcea et al. [47] proposed the use of Proxy Re-Encryption (PRE) [48] to ensure the E2E security between the publisher and the subscriber without the need for any pre-negotiation. A trusted policy authority is used to create a re-encryption key for each subscribers. That key is used by the broker to re-encrypt the published message by the publisher. This method suffers from a number of pitfalls: 1) The use of asymmetric encryption to encrypts the published message (messages are encrypted using the public key of the publisher). 2) The need to change the implementation of the broker. 3) It does not ensure the integrity of the published messages. 4) The reuse of the same key for multiple topics.

B. Authorization

Few papers focused only on providing authorization solutions for Pub/Sub-based communication. Most research tried to solve the authorization as part of a comprehensive framework that ensures authorization besides confidentiality. However, there are still few papers where the authorization was the primary focus. Among these papers, many authorization mechanisms were adopted. For example, Access Control List (ACL) was adopted in [49], [50], Attribute Based Access Control (ABAC) [51], [52] security scheme was adopted to secure MQTT protocol in [53], Role-Based Access Control (RBAC) [54] was also adopted by many research such as [55], [56]. In most of these solutions, the goal is to determine whether a certain node is allowed to publish or subscribe to a particular topic. Also, the policy is not derived directly from the owner of the data. Finally, the broker was used as a central trusted component to enforce the policy rules. Due to this fact, the broker becomes a single point of failure.

Besides the conventional access control models that we mentioned before, some research papers adopted alternative approaches such as policy-based access control [57], [58] and capability-based access control [59]–[61]. In [57], authors proposed the use of a policy-based approach, similar to our solution, to allow the data publisher to control who can access their publications and under what condition. However, the broker was fully trusted and used to evaluate and enforce the security policy.

C. Confidentiality and Authorization

Providing a comprehensive solution which ensures both the confidentiality of the published messages and gives the publisher a level of access control on the published messages was proposed by many papers such as [15], [21], [62]–[65].

Hamad et al. [21] proposed the use of secret splitting [66] to divide the symmetric topic key and share it with many KeyStores using the RSA cryptosystem [67]. With the help of the KeyStores, the publisher can control who can access the part of the key by issuing a security policy that grants the access right to the subscribers without violating the decoupling property

TABLE VI: Comparison of SEEMQTT with other systems.

Year	Solution	R1	R2	R3	R4	Message Encryption	Topic Key Exchange	Authorization
2016	AIPS [43]	●	○	○	○	AES (Symmetric)	IBE	–
2016	[23]	●	○	○	●	AES (Symmetric)	TLS-based	ACL
2018	[18]	○	●	○	●	AES (Symmetric)	AES with PSK	ACL
2017	PICADOR [47]	●	●	○	○	PRE (Asymmetric)	–	–
2020	[50]	●	○	◑	○	AES (Symmetric)	TLS-based	ABAC
2007	[57]	●	◑	●	○	–	–	Trust delegation
2021	ENTRUST [19]	○	●	○	●	ChaCha20 (Symmetric)	ECDH with PSK	ACL
2012	P3S [64]	●	●	●	○	CP-ABE (Asymmetric)	ABE	Attribute-based
2015	SMQTT [15]	●	●	●	○	CP/KP-ABE (Asymmetric)	ABE	Attribute-based
2014	[16]	●	●	●	○	AES (Symmetric)	ABE & AES	Attribute-based
2021	SPPS [21]	●	●	●	○	AES (Symmetric)	RSA & AES	Trust delegation
2022	SEEMQTT	●	●	●	●	AES (Symmetric)	IBE & AES	Trust delegation

○ Requirement is not met ◑ Requirement is not discussed but it could be partially met ● Requirement is met
R1: No pre-shared Key R2: E2E Encryption R3: Decoupled authorization R4: Efficiency

of the Pub/Sub model. The need to verify the certificate of every KeyStore can cause an overhead. This overhead can be significant performance-wise for mobile devices. Also, using secret splitting for dividing the key requires the availability of all KeyStores to retrieve the key by subscribers.

Singh et al. [15] investigated the use of Ciphertext-Policy ABE (CP-ABE) and Key Policy based ABE (KP-ABE) to implement a secure MQTT protocol. In their proposal, they used the broker as a PKG that creates master public parameters and private keys for each subscriber based on an access policy. A publisher node uses the public parameters, provided by the broker, and the access policy to encrypt a message. Only subscribers who fulfill the access policy can decrypt that message. Similarly, Pal et al. [64] proposed a system for content-based Pub/Sub system where CP-ABE was used to encrypt published messages and store it in repository services. Subscribers who have properties that satisfy the publisher policy can obtain the message and decrypt it. Instead of using ABE to encrypt the published data, Wang et al. [16] proposed the use of an ABE-based scheme to encrypt a symmetric key that is used to encrypt the data. Only subscribers with proper properties can get the symmetric key and consequently decrypt the message. Although these solutions ensure both data authorization and confidentiality, they come with a massive overhead due to pairing operations needed in ABE. Also, the number of attributes that specify the authorized subscriber plays a significant role in the imposed overhead. In our proposed solution, we overcame that by not deriving the encryption key from the access policy.

D. Comparative Study

1) Solutions and Requirements: TABLE VI presents a comparison between our proposal and some of the solutions presented above. The comparison is based on the requirements introduced in § I, and the mechanism used to implement these requirements (mainly R2 and R3). This includes the encryption algorithms that were used to encrypt the published messages and the topic key, and the mechanism used to implement the access control solution. Regarding R4, we considered this requirement was met only if the solution was implemented and evaluated using a constrained device.

In our protocol, we use a symmetric encryption to protect the data, which is far lighter than asymmetric methods likes the ones used by [15], [47], [64]. Although, many other presented solutions used symmetric encryption similar to our solution, they differ in other aspects such as the need for a pre-shared secret with the key store [18], [19], trusting the broker [23], [43], [50], or using different asymmetric encryption schemes to support the topic key exchange [16], [21]. Based on TABLE VI, the most two comparable solution to our system are [21] and [16]. Both use symmetric encryption to protect the published messages and enable the publisher access control. However, different asymmetric encryption schemes were used to protect the topic key. Furthermore, both solutions were not evaluated using constrained devices (more about that later). Based on the system architecture, both [18] and [19] are very comparable to our system since they used a server to generate and store the topic keys. However, both solutions require a PSK between every node and that server, while our solution does not require that. Also, our solution used many KeyStores instead of one server. Finally, our proposal does not consider each server as a fully trusted node.

2) Performance Evaluation: Comparing our proposed solution with existing ones is a challenging task due to the difficulty of reproducing the previous results of most of the research, the use of different platforms, or the inapplicability to run these solutions using constrained devices. However, we try in this section to detail the performance evaluation of the four most comparable solutions (i.e., [16], [18], [19], [21]) that were discussed in § VIII-D1. TABLE VII shows that performance evaluation as well as the platform used to evaluate each one of these solutions. Although using a PSK between the Key server and the publisher in both [16] and [19] makes the exchange of a topic key faster compared to our system, it makes both solutions impracticable for the mobile IoT system as we have shown in TABLE VII. Furthermore, the performance evaluation of SPPS [21] does not consider the time required to verify the KeyStores' certificates. The verification of digital certificates can introduce a considerable overhead when it is performed on a constrained devices. Finally, while it can be seen at first glance that the performance of [16] is comparable to our system's performance, it is essential to note that [16] was evaluated using a very powerful

TABLE VII: Time performance comparison of SEEMQTT with other systems.

Solution	Performance Evaluation	Platform
[18]	$\approx 520\,\mathrm{ms}$ to exchange the topic key	Nordic Semiconductor NRF52 Development Kit
ENTRUST [19]	$\approx 100\,\mathrm{ms}$ to perform ECDH and topic key exchange	RedBear Duo with $120\,\mathrm{MHz}$ ARM Cortex-M3 CPU
[16]	$\approx 1000\,\mathrm{ms}$ to encrypt a 128-bit key using CP-ABE with a single attribute	PC with $1.60\,\mathrm{GHz}$ Intel Quad-Core i7 2677M CPU and $4\,\mathrm{GB}$ RAM
SPPS [21]	$\approx 200\,\mathrm{ms}$ to exchange topic key with two KeyStores	Raspberry Pi 3 Model B+ with $1.4\,\mathrm{GHz}$ quad-core ARM Cortex-A53 CPU and $1\,\mathrm{GB}$
SEEMQTT	$\approx 900\,\mathrm{ms}$ to exchange topic key with two KeyStores	ESP32 board with a $240\,\mathrm{MHz}$ Xtensa dual-core 32-bit LX6 microprocessor

platform compared to the one that we used.

IX. Conclusion

Using the Pub/Sub model to support the communication of mobile IoT systems seems promising if security concerns are solved. This paper presents SEEMQTT, an E2E secure MQTT-based communication for mobile IoT systems. SEEMQTT allows the publisher to use a symmetric key to encrypt the published data. To share this key with the subscribers without violating the decoding requirement of Pub/Sub model, our solution adopts Shamir's secret sharing scheme to generate and share the key with several honest-but-curious KeyStores. Also, our system employs IBE to set up secure links between the publisher and every KeyStore that are used to exchange the generated secret shares. Finally, our system uses the trust delegation to enable the publisher to authorize subscribers to access these shares. Only authorized subscribers can obtain the shares and combine them to reconstruct the key that was used to encrypt the published message. We have developed an Arduino-based library to support the implementation of the publisher on the ESP32 platform. Also, we have provided a Linux-based implementation for KeyStore. Our proposed solution was evaluated using real IoT devices. The experiment results show that our solution outperforms alternative state-of-the-art methods such as ABE-based solutions even though these solutions were evaluated on very powerful platforms compared to ours. Based on that, our solution is considered a very efficient solution to ensure secure E2E MQTT-based communication for mobile IoT systems. As a future work, we plan to investigate the adoption of lightweight IBE schemes. In addition, we plan to investigate other trust delegation access control schemes which could be used instead of the KeyNote trust management system.

Acknowledgments

This work is partially supported by the European Union's Horizon 2020 research and innovation programme through the following H2020 projects: nIoVe (A Novel Adaptive Cybersecurity Framework for the Internet of Vehicles) under Grant Agreement No. 833742 and CONCORDIA under Grant Agreement No. 830927.

Appendix

Performance Evaluation of ComShare Algorithm Using ESP32 Platform

The `ComShare` will be executed on the subscribers. Since we are more interested in the evaluation of the protocol on the Pub, we did not include the performance evaluation of this algorithm in the results section and decide to move it here to the appendix. Fig. 17a presents the required time to combine two shares (i.e., $t = 2$) when different values of n were used ($n = 2, \ldots, 15$). As it was expected, the results shows that the time is almost the same regardless of the value of n. This result is in line with (2) which indicates that calculating the key depends only on the number of valid shares t that are used to define the polynomial f of degree $t - 1$. Fig. 17b presents the required time to combine t shares for all values of $t \in [2, 15]$. The figure show that the using a large value of t will introduce more overhead during the re-construction of the key. It important to note that the presented time does not include the time to obtain the shares from different KeyStores.

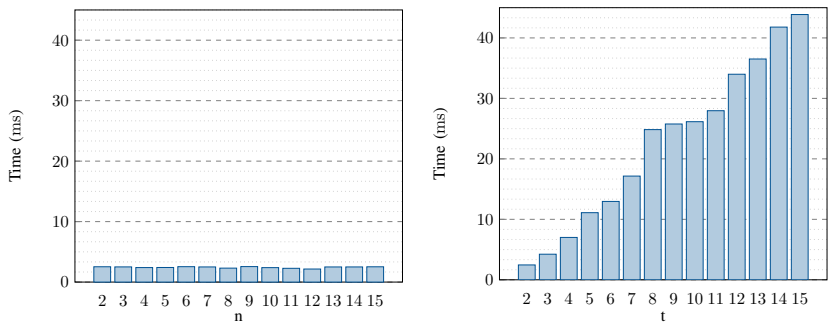
Performance Evaluation of IBE Algorithms Using ESP32 Platform

In this section, we present the evaluation of the four IBE algorithms, namely `Setup`, `Extract`, `Encrypt`, and `Decrypt`, when they were executed on resource-constrained IoT devices such as the ESP32 board that was used as a publisher in our testbed. Fig. 18a presents the time required to execute the `Setup` and `Extract` algorithms. From the figure, it can be seen that the time to execute the `Setup` algorithm is larger than the time needed to perform the `Extract` algorithm. This outcome is consistent with the result that we have obtained when RPI4 was used (see Fig. 12). However, the measured time for every algorithm is almost 24 times slower than the measured time when RPI4 was used. Fig. 18b presents the time required to `Encrypt` different message sizes. No significant difference was found between the consumed time of encrypting different message sizes. This was also the case for the decryption time as shown in Fig. 18c. By comparing the consumed time of both algorithms, it is clear that the decryption time is almost 7 times slower than the encryption time. Note that the time for calculating the g_{ID}, which is $1033.1\,\mathrm{ms}$, was not included as a part of the encryption time.

References

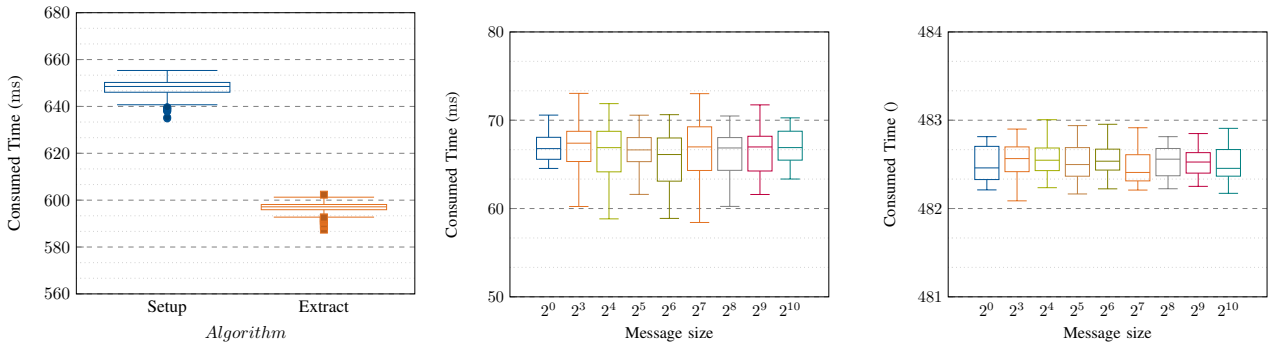
[1] S. Herrnleben, M. Pfannemüller, C. Krupitzer, S. Kounev, M. Segata, F. Fastnacht, and M. Nigmann, "Towards adaptive car-to-cloud communication," in *2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. IEEE, 2019, pp. 119–124.

[2] S. Shen, Z.-Q. Wei, L.-J. Sun, Y.-Q. Su, R.-C. Wang, and H.-M. Jiang, "The shared bicycle and its network—internet of shared bicycle (iosb): A review and survey," *Sensors*, vol. 18, no. 8, p. 2581, 2018.



(a) $t = 2$ (b) $\forall n \in [2, t]$

Fig. 17: The consumed time during the executing of the `ComShare` on an ESP32 platform while using different values of n and t.



(a) (b) (c)

Fig. 18: The consumed time during the executing of (a) the `Setup` and `Extract` algorithms, (b) the `Encrypt` algorithm, and (c) `Decrypt` algorithm over an ESP32 platform when different message sizes were used.

[3] F. Luo, C. Jiang, S. Yu, J. Wang, Y. Li, and Y. Ren, "Stability of cloud-based uav systems supporting big data acquisition and processing," *IEEE Transactions on Cloud Computing*, vol. 7, no. 3, pp. 866–877, 2017.

[4] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[5] T. Mishra, D. Garg, and M. M. Gore, "A publish/subscribe communication infrastructure for vanet applications," in *2011 IEEE Workshops of International Conference on Advanced Information Networking and Applications*. Biopolis, Singapore: IEEE, 2011, pp. 442–446.

[6] P. Nunes, C. Nicolau, J. P. Santos, and A. Completo, "From a traditional bicycle to a mobile sensor in the cities." in *VEHITS*, 2020, pp. 81–88.

[7] A. Chodorek, R. R. Chodorek, and P. Sitek, "Uav-based and webrtc-based open universal framework to monitor urban and industrial areas," *Sensors*, vol. 21, no. 12, p. 4061, 2021.

[8] C. Wang, A. Carzaniga, D. Evans, and A. L. Wolf, "Security issues and requirements for internet-scale publish-subscribe systems," in *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*. Big Island, HI, USA: IEEE, 2002, pp. 3940–3947.

[9] S. Andy, B. Rahardjo, and B. Hanindhito, "Attack scenarios and security analysis of mqtt communication protocol in iot system," in *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2017, pp. 1–6.

[10] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 8446, August 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt

[11] H. Hidayat, P. Sukarno, and A. A. Wardana, "Overhead analysis on the use of digital signature in mqtt protocol," in *2019 International Conference on Electrical Engineering and Informatics (ICEEI)*. Indonesia: IEEE, 2019, pp. 87–92.

[12] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE symposium on security and privacy (SP'07)*. Oakland, California, USA: IEEE, 2007, pp. 321–334.

[13] A. Sahai and B. Waters, "Fuzzy identity-based encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Aarhus, Denmark: Springer, May 2005, pp. 457–473.

[14] M. Ion, G. Russello, and B. Crispo, "Design and implementation of a confidentiality and access control solution for publish/subscribe systems," *Computer networks*, vol. 56, no. 7, pp. 2014–2037, 2012.

[15] M. Singh, M. Rajan, V. Shivraj, and P. Balamuralidhar, "Secure MQTT for Internet of Things (IoT)," in *2015 Fifth International Conference on Communication Systems and Network Technologies*. Gwalior, MP, India: IEEE, April 2015, pp. 746–751.

[16] X. Wang, J. Zhang, E. M. Schooler, and M. Ion, "Performance evaluation of attribute-based encryption: Toward data privacy in the iot," in *2014 IEEE International Conference on Communications (ICC)*. Sydney, Australia: IEEE, 2014, pp. 725–730.

[17] B. Girgenti, P. Perazzo, C. Vallati, F. Righetti, G. Dini, and G. Anastasi, "On the feasibility of attribute-based encryption on constrained iot devices for smart systems," in *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. Washington, DC, USA: IEEE, 2019, pp. 225–232.

[18] S. Nolan, "Authenticated payload encryption scheme for internet of things systems over the mqtt protocol," Master's thesis, Dublin, Ireland: Trinity Collage Dublin, The University of Dublin, 2018.

[19] 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*, 2021, pp. 78–87.

[20] A. Paverd, A. Martin, and I. Brown, "Modelling and automatically analysing privacy properties for honest-but-curious adversaries," *Tech. Rep*, 2014.

[21] M. Hamad, E. Regnath, J. Lauinger, V. Prevelakis, and S. Steinhorst, "Spps: Secure policy-based publish/subscribe system for v2c communication," in *2021 Design, Automation Test in Europe Conference Exhibition (DATE)*. Virtual: IEEE, 2021, pp. 529–534.

[22] C. Bormann, M. Ersue, A. Keränen, and C. Gomez, "Terminology

for Constrained-Node Networks," Internet Engineering Task Force, Internet-Draft draft-bormann-lwig-7228bis-07, Oct. 2021, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-bormann-lwig-7228bis-07

[23] ISO, "Information technology – Message Queuing Telemetry Transport (MQTT) v3.1.1," International Organization for Standardization, Geneva, Switzerland, ISO ISO/IEC 20922:2016, 2016.

[24] Y. Jia, L. Xing, Y. Mao, D. Zhao, X. Wang, S. Zhao, and Y. Zhang, "Burglars' iot paradise: Understanding and mitigating security risks of general messaging protocols on iot clouds," in *2020 IEEE Symposium on Security and Privacy (SP)*. virtual: IEEE, 2020, pp. 465–481.

[25] E. Onica, P. Felber, H. Mercier, and E. Rivière, "Confidentiality-preserving publish/subscribe: A survey," *ACM computing surveys (CSUR)*, vol. 49, no. 2, pp. 1–43, 2016.

[26] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[27] ——, "Identity-based cryptosystems and signature schemes," in *Workshop on the theory and application of cryptographic techniques*. Paris, France: Springer, April 1984, pp. 47–53.

[28] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*. Santa Barbara, California, USA: Springer, 2001, pp. 213–229.

[29] M. Blaze, J. Feigenbaum, and J. Lacy, "Decentralized trust management," in *Proceedings 1996 IEEE Symposium on Security and Privacy*. Oakland, CA, USA: IEEE, 1996, pp. 164–173.

[30] J. Feigenbaum, D. J. Ioannidis, A. D. Keromytis, and D. M. Blaze, "The KeyNote Trust-Management System Version 2," RFC 2704, Sep. 1999. [Online]. Available: https://rfc-editor.org/rfc/rfc2704.txt

[31] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on information theory*, vol. 29, no. 2, pp. 198–208, 1983.

[32] M. Bellare and C. Namprempre, "Authenticated encryption: Relations among notions and analysis of the generic composition paradigm," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2000, pp. 531–545.

[33] S. Briais and U. Nestmann, "A formal semantics for protocol narrations," *Theoretical Computer Science*, vol. 389, no. 3, pp. 484–511, 2007.

[34] M. Bugliesi and P. Modesti, "Anbx-security protocols design and verification," in *Joint Workshop on Automated Reasoning for Security Protocol Analysis and Issues in the Theory of Security*. aphos, Cyprus: Springer, March 2010, pp. 164–184.

[35] M. Blaze, J. Ioannidis, and A. D. Keromytis, "Experience with the keynote trust management system: Applications and future directions," in *International Conference on Trust Management*. Springer, 2003, pp. 284–300.

[36] F. T. Penney, "Implementation of Shamir's Secret Sharing in C," 2017, accessed: 2021-11-27. [Online]. Available: http://fletcher.github.io/c-sss/index.html

[37] D. McGrew and J. Viega, "The galois/counter mode of operation (gcm)," *submission to NIST Modes of Operation Process*, vol. 20, pp. 0278–0070, 2004.

[38] S. Blake-Wilson, N. Bolyard, V. Gupta, C. Hawk, and B. Moeller, "Elliptic curve cryptography (ecc) cipher suites for transport layer security (tls)," RFC 4492, Tech. Rep., 2006.

[39] C. Esposito and M. Ciampi, "On security in publish/subscribe services: A survey," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 2, pp. 966–997, 2014.

[40] A. V. Uzunov, "A survey of security solutions for distributed publish/subscribe systems," *Computers & Security*, vol. 61, pp. 94–129, 2016.

[41] J. Munster, "Securing publish/subscribe," Master's thesis, University of Toronto, 2018.

[42] H. Lee, J. Lim, and T. T. Kwon, "Mqtls: Toward secure mqtt communication with an untrusted broker," in *2019 International Conference on Information and Communication Technology Convergence (ICTC)*, 2019, pp. 53–58.

[43] W. Peng, S. Liu, K. Peng, J. Wang, and J. Liang, "A secure publish/subscribe protocol for internet of things using identity-based cryptography," in *2016 5th International Conference on Computer Science and Network Technology (ICCSNT)*. Changchun,China: IEEE, December 2016, pp. 628–634.

[44] H.-Y. Chien, P.-C. Lin, and M.-L. Chiang, "Efficient mqtt platform facilitating secure group communication," *Journal of Internet Technology*, vol. 21, no. 7, pp. 1929–1940, 2020.

[45] A. Bashir and A. H. Mir, "Securing communication in mqtt enabled internet of things with lightweight security protocol," *EAI Endorsed Transactions on Internet of Things*, vol. 3, no. 12, pp. 1–6, 10 2017.

[46] A. Mektoubi, H. L. Hassani, H. Belhadaoui, M. Rifi, and A. Zakari, "New approach for securing communication over mqtt protocol a comparaison between rsa and elliptic curve," in *2016 Third International Conference on Systems of Collaboration (SysCo)*. Casablanca, Morocco: IEEE, 2016, pp. 1–6.

[47] C. Borcea, Y. Polyakov, K. Rohloff, G. Ryan *et al.*, "Picador: End-to-end encrypted publish–subscribe information distribution with proxy re-encryption," *Future Generation Computer Systems*, vol. 71, pp. 177–191, 2017.

[48] A.-A. Ivan and Y. Dodis, "Proxy cryptography revisited," in *The Network and Distributed System Security Symposium (NDSS)*. San Diego, California, USA: The Internet Society, 2003.

[49] Y. Upadhyay, A. Borole, and D. Dileepan, "Mqtt based secured home automation system," in *2016 Symposium on Colossal Data Analysis and Networking (CDAN)*. Indore, Madhya Pradesh, India: IEEE, 2016, pp. 1–4.

[50] A. Gabillon, R. Gallier, and E. Bruno, "Access controls for iot networks," *SN Computer Science*, vol. 1, no. 1, p. 24, 2020.

[51] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, K. Scarfone *et al.*, "Guide to attribute based access control (abac) definition and considerations (draft)," *NIST special publication*, vol. 800, no. 162, pp. 1–54, 2013.

[52] P. Colombo and E. Ferrari, "Access control enforcement within mqtt-based internet of things ecosystems," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, ser. SACMAT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 223–234. [Online]. Available: https://doi.org/10.1145/3205977.3205986

[53] A. Rizzardi, S. Sicari, D. Miorandi, and A. Coen-Porisini, "Aups: an open source authenticated publish/subscribe system for the internet of things," *Information Systems*, vol. 62, pp. 29–41, 2016.

[54] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[55] J. Bacon, D. M. Eyers, J. Singh, and P. R. Pietzuch, "Access control in publish/subscribe systems," in *Proceedings of the second international conference on Distributed event-based systems*. Rome, Italy: ACM, July 2008, pp. 23–34.

[56] A. Belokosztolszki, D. M. Eyers, P. R. Pietzuch, J. Bacon, and K. Moody, "Role-based access control for publish/subscribe middleware architectures," in *Proceedings of the 2nd international workshop on Distributed event-based systems*. San Diego California: ACM, June 2003, pp. 1–8.

[57] L. Opyrchal, A. Prakash, and A. Agrawal, "Supporting privacy policies in a publish-subscribe substrate for pervasive environments." *J. Networks*, vol. 2, no. 1, pp. 17–26, 2007.

[58] J. L. Hernández-Ramos, G. Baldini, R. Neisse, M. Al-Naday, and M. J. Reed, "A policy-based framework in fog enabled internet of things for cooperative its," in *2019 Global IoT Summit (GIoTS)*, 2019, pp. 1–6.

[59] Y. Nakamura, Y. Zhang, M. Sasabe, and S. Kasahara, "Capability-based access control for the internet of things: An ethereum blockchain-based scheme," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[60] J. L. H. Ramos, A. J. Jara, L. Marín, and A. F. Gómez-Skarmeta, "Distributed capability-based access control for the internet of things," *J. Internet Serv. Inf. Secur.*, vol. 3, pp. 1–16, 2013.

[61] S. Gusmeroli, S. Piccione, and D. Rotondi, "A capability-based security approach to manage access control in the internet of things," *Mathematical and Computer Modelling*, vol. 58, no. 5, pp. 1189–1205, 2013, the Measurement of Undesirable Outputs: Models Development and Empirical Analyses and Advances in mobile, ubiquitous and cognitive computing. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S089571771300054X

[62] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM conference on Computer and communications security*. Alexandria, Virginia, USA: ACM, 2006, pp. 89–98.

[63] M. Ion, "Security of publish/subscribe systems," Ph.D. dissertation, University of Trento, 2013.

[64] P. Pal, G. Lauer, J. Khoury, N. Hoff, and J. Loyall, "P3s: A privacy preserving publish-subscribe middleware," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Beijing, China: Springer, December 2012, pp. 476–495.

[65] M. A. Tariq, B. Koldehofe, A. Altaweel, and K. Rothermel, "Providing basic security mechanisms in broker-less publish/subscribe systems," in *Proceedings of the Fourth ACM International Conference on Distributed*

Event-Based Systems. Cambridge, United Kingdom: ACM, 2010, pp. 38–49.

[66] S. Bruce, *Applied cryptography: protocols, algorithms, and source code in C*. Wiley, 1996, p. 70.

[67] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, p. 120–126, feb 1978. [Online]. Available: https://doi.org/10.1145/359340.359342

Mohammad Hamad received the B.Eng. degree in software engineering and information system from Aleppo University, Syria, in 2009 and the Ph.D. (Dr.-Ing.) degree in computer engineering from the Institute for Data Technology and Communication Networks at the Technical University of Braunschweig, Germany, in 2020.

Since 2020, he is a postdoctoral researcher in the Embedded Systems and Internet of Things group in the Faculty of Electrical Engineering and Information Technology at the Technical University of Munich (TUM). His research interests are in the area of autonomous vehicle and IoT security.

Andreas Finkenzeller received both the B.Sc. and M.Sc. degree in electrical engineering and computer science at Technical University Munich (TUM), Germany.

Since 2021, he is working in the Embedded Systems and Internet of Things Group at TUM as a PhD candidate. His research interests include embedded systems, secure communication, and IoT Security.

Hangmao Liu received the B.Sc. degree in electrical and computer engineering from Technical University of Kaiserslautern, Germany, in 2017, and the M.Sc. (Dipl.-Ing.) degree in electrical engineering and information technology from Technical University of Munich (TUM), Germany, in 2021.

His research interest focuses on the cyber security in IoT (Internet of Things) networks.

Jan Lauinger is a Research Associate and PhD Candidate at the Professorship of Embedded Systems and Internet of Things, Technical University of Munich. He obtained his B.Sc. and M.Sc. degree in Electrical Engineering and Information Technology at TUM focusing on large scale anomaly detection, network architectures and services, automation and robotics. His current field of research encompasses cyber security solutions for the IoT/Internet of Vehicles, where he contributes to the EU-funded nIoVe project.

Vassilis Prevelakis is the professor of embedded computer security at the Technical University, Braunschweig, in Germany. He holds B.Sc. degrees with Honours in Mathematics and Computer Science and M.Sc. in Computer Science from university of Kent at Canterbury, U.K. and a Ph.D. in Computer Science from university of Geneva, Switzerland. He has worked in various areas of security in Systems and Networks both in his current academic capacity and as a freelance consultant.

Prevelakis current research involves issues related to vehicular automation security, secure processors, security aspects of software engineering, auto-configuration issues in secure VPNs, etc.

Sebastian Steinhorst received the M.Sc. (Dipl.-Inf.) degree in computer science from the Goethe University Frankfurt, Germany, in 2005 and the Ph.D. (Dr. phil. nat.) degree in computer science from the same university in 2011.

He is an associate professor at Technical University of Munich (TUM) in Germany. He leads the Embedded Systems and Internet of Things group in the Department of Electrical and Computer Engineering. He was also a co-program PI in the Electrification Suite and Test Lab of the research center TUMCREATE in Singapore. The research of Prof. Steinhorst centers around design methodology and hardware/software architecture co-design of secure distributed embedded systems for use in IoT, automotive and smart energy applications.