Time-Specific Integrity Service in MQTT Protocol

Haotian Yan
The Hong Kong Polytechnic University
Hong Kong, China
haotian-ht.yan@connect.polyu.hk

Haibo Hu
The Hong Kong Polytechnic University
Hong Kong, China
haibo.hu@polyu.edu.hk

Qingqing Ye
The Hong Kong Polytechnic University
Hong Kong, China
qqing.ye@polyu.edu.hk

ABSTRACT

Message Queuing Telemetry Transport (MQTT) is a classic transmission protocol in IoT scenarios, where a subscriber subscribes to the messages, and a publisher publishes the message to the broker, who then sends the message to the subscriber. In MQTT, it is essential to ensure its security in the temporal domain. On one hand, some messages are time-sensitive, so their integrity should only be valid in a determined time interval. On the other hand, when the subscriber is out of the service, his key should not be able to verify the message, which is known as subscriber validity. This paper discusses the time-specific integrity service in the MQTT protocol. To solve the problem, we propose Time-specific Signcryption (TSSC) and the Dynamic Time-specific Signature (DTSS). Furthermore, a Progressive Dynamic Time-specific Signature scheme (Pro-DTSS) is proposed to save communication costs. The theoretical and experimental results show that our proposed schemes are more efficient than the existing solutions.

KEYWORDS

Internet of Things, Message Queuing Telemetry Transport, Timed Integrity, Message Integrity, Subscriber Validity

1 INTRODUCTION

The Internet of Things (IoT) has been proposed for many years and become widely adopted in daily life. There are several transmission protocols in IoT scenarios to achieve message exchange, such as HTTP [9], MQTT[25], and COAP [11]. Among them, MQTT is a practical and lightweight transmission protocol, whose header is only 2 bytes. Additionally, it can support various network configurations, such as one-to-many, many-to-one, and many-to-many networks. There are three entities in the MQTT scenario: the Publishers (data senders), the Subscribers (data receivers), and the Broker (the intermediary entity facilitating communication between Publishers and Subscribers).

IoT can provide several kinds of service. Among them, the timed service [8] [28] [6] has attracted much attention. For instance, while an Unmanned Aerial Vehicle (UAV) is flying in the smart city to gather messages, some IoT devices (e.g., Road Units) cannot communicate with the UAV until it moves inside their transmission range. Another application is the sleeping mode. In the MQTT scenario, the IoT device gives feedback (i.e., ACK packet) to the sender to confirm the successful reception of messages. However, the IoT device might be in sleeping mode to save energy and cannot send any messages. In this situation, the sender cannot determine whether the IoT device is in sleeping mode or has been out of service. When the timed service is applied in such a situation, the service will continue if the receiver gives feedback during the predetermined time interval. Otherwise, the service will be terminated.

Unlike the traditional service, the timed service is only available during a predetermined time interval. Therefore, it is essential to maintain the message integrity [33] within the designated time interval, so that the adversary cannot modify the message during transmission. Digital signature is a practical solution to this problem and has been widely applied in IoT scenarios. However, when one participant is not actively engaged in the service, his key must be revoked from him to stop using it to verify the message. On one hand, the participant can only receive and verify the message when it is available. On the other hand, when the participant is out of the service, he cannot ensure the correction of the message in the future. Formally, it is called the timed integrity problem.

In cryptography, using timestamp can guarantee that the signature is generated in the predetermined time instant and protect the validity of the signature. However, it cannot verdict that a key is invalid when the service is not in the predetermined time interval. A naive solution is to adopt Time-specific Sign-then-Encrypt (TSS&E) for this scenario. The sender first signs the message and encrypts a secret value a. Then, he sends the ciphertext to the receiver. Upon receiving the message, the receiver decrypts the ciphertext, verifies the message, and updates the key. If the receiver is not in the service, his key cannot be applied to verify the message in the future. Unfortunately, this method is not efficient. Alternatively, distance-bounding protocols [21] can solve the timed integrity problem for mobile IoT devices (e.g., UAVs). However, such protocols cannot work for stationary IoT devices in sleeping mode, as the distance between the sender and receiver does not change.

Quality of Service (QoS) is a crucial feature in the MQTT scenario that guarantees message delivery, commonly by the subscriber sending acknowledgement when receiving the message. However, there are two situations in which subscribers cannot send acknowledgement immediately. Either the subscribers are in sleeping mode, or they are out of the service. To the best of our knowledge, previous works have not studied on timed message integrity in MQTT by key cancellation. We propose two solutions to address the problem. The first, Time-specific Signcryption (TSSC), is a baseline solution. The second one is called Dynamic Time-Specific Signature (DTSS). TSSC is based on the classic signcryption scheme, while the DTSS combines the signature scheme with dynamic secret sharing to solve the timed integrity problem. Currently, our proposed schemes are embedded in EMQX [14], which is a popular MQTT service provider. The proposed schemes can also be applied to other MQTT service providers.

There are some examples in practical scenarios where our proposed schemes can solve the timed integrity problem.

979-8-3503-6201-5/24/$31.00 ©2024 IEEE
DOI 10.1109/IPSN61024.2024.00014

(1) **Spatial-temporal Crowdsourcing.** Crowdsourcing allows the workers to accomplish tasks from the platform. Nevertheless, neither the platform nor the workers can be trusted entirely. The platform should ensure that the messages received from the workers are gathered within the predetermined time interval. In the meantime, the message integrity should be guaranteed. Therefore, the TSSC or DTSS can be used to sign the message. The timed integrity can be assured if the message can be verified in the predetermined time interval. Otherwise, the platform can ignore the message. Furthermore, when the worker is out of the service, the platform can cancel the worker's key. Consequently, the worker cannot use the key or give the key to other workers.

(2) **Mobile Edge Computing.** Due to the limited transmission range and mobility, IoT devices (e.g., UAVs) cannot exchange messages with static edge nodes (e.g., street lights). Consequently, a time-constraint IoT service is proposed. The UAV will move near the street light and gather the messages during the predetermined time interval. As such, ensuring the message integrity for the duration of the time interval is necessary. The UAV can apply the TSSC or DTSS to guarantee message integrity in the predetermined time range. Specifically, the UAV can specify a time interval during which it will travel inside the communication range of the static edge node. Then, it can get the keys based on the time interval and apply them for verification. Furthermore, when the UAV is not in service, its key will be cancelled. It cannot have the service even if it returns to the communication range again.

(3) **Age of Information (AoI).** It aims to measure the freshness of the data. Since the data only update periodically, a tiny AoI can lead to a high profit. In the crowdsensing scenario, selfish workers can claim that the AoI of the data is small when transmitting the data to the requester. For instance, if the worker received the data with a large AoI, he can wait until the data is updated again and transmit the data to the requester. When the TSSC or DTSS signs the data, the worker cannot delay the transmission of the message. Otherwise, the requester cannot verify the message since the key is outdated, and the message integrity cannot be guaranteed.

In summary, the contribution of this paper is as follows.

- We propose that the TSSC and DTSS achieve timed integrity service for MQTT. In the meantime, the service provider can cancel the public key. To the best of our knowledge, it is the first work that focuses on the timed integrity problem with key cancellation in the MQTT scenario.
- We propose a communication-efficient and lightweight scheme called Pro-DTSS in the data stream transmission situation, which can not only inherent the advantage of the DTSS but also decrease communication costs without compromising security.
- We use the theoretical analysis and the experimental evaluation on the performance of our proposed schemes. The results prove that our proposed schemes achieve good efficiency.

The rest of this paper is organized as follows. Sec. 2 defines the main problem and the network model. Sec. 3 gives some relevant background knowledge. Sec. 4 introduces our proposed schemes, and Sec. 5 gives an improved scheme in the long-range transmission. Sec. 6 analyzes the performance of the proposed schemes. Sec. 7 uses the experiments to show the performance in practical situations. Sec. 8 shows some related work. Sec. 9 concludes this paper.

2 PROBLEM DEFINITION AND NETWORK MODEL

2.1 Problem Definition

In this section, we define the problem we are trying to solve: how the verifier can check the message integrity in a determined time interval. The problem can be divided into message integrity, timed integrity, and subscriber validity.

Formally, the definition is given as follows,

Definition 1. ***Message Integrity:*** *The message integrity can be guaranteed so that the adversary cannot modify the message without changing the signature.*

$$\begin{aligned} &verify(x \in \mathcal{X}, t \in \mathcal{T}, pk, \sigma) = 1 \\ &verify(x' \in \mathcal{X}, t \in \mathcal{T}, pk, \sigma) = 0 \end{aligned} \tag{1}$$

Definition 2. ***Timed Integrity:*** *The timed integrity means the message integrity can only be guaranteed during the determined time interval.*

$$\begin{aligned} &verify(x \in \mathcal{X}, t \in \mathcal{T}, pk, \sigma) = 1 \\ &verify(x \in \mathcal{X}, t' \notin \mathcal{T}, pk, \sigma) = 0 \end{aligned} \tag{2}$$

Insides, x and x' are two different messages from the domain $\mathcal{X}$. t and t' are two different time instants. Specifically, t is from the predetermined time domain T, while t' is not. pk is the public key for verification and σ is the signature of x to ensure the message integrity.

In practical application, the timed integrity service can ensure that the messages have not been changed during the time interval. Otherwise, the correction of the message cannot be guaranteed. For instance, when the IoT devices are in sleeping mode, they cannot send the ACK message when they sleep. Message integrity cannot be ensured once the waiting time exceeds the predetermined time interval. The other application is the time-constraint IoT service. The mobility of the UAV will affect the message delivery time. The UAV has left the restricted area if the message does not arrive during the predetermined time interval. Then, the message is out-of-date, and the UAV can no longer ensure the message integrity.

In the meantime, the subscriber validity is important, which is defined as follows,

Definition 3. ***Subcriber Validity:*** *The subscriber has no further access to the integrity service when he is out of the service.*

$$\begin{aligned} &\exists i \quad verify(x_i \in \mathcal{X}, t_i \notin \mathcal{T}, pk, \sigma_i) = 0 \\ \Rightarrow &\forall j > i \quad verify(x_j \in \mathcal{X}, t_j \in \mathcal{T}, pk, \sigma_j) \neq 1 \end{aligned} \tag{3}$$

x_i and x_j are two different messages. σ_i and σ_j are the corresponding signature. t_i is not in the predetermined time domain, while t_j is in that time domain. Following the previous example, his key for the service is still valid even if the subscriber does not verify the message during his working time or he is situated beyond the transmission range of the targeted IoT device. Consequently, the service provider should remotely deactivate the keys to prohibit the

subscriber from having the integrity service until he registers again. With subscriber validity, the key cancellation can be achieved.

2.2 Network Model

Fig. 1 shows the system, which includes four entities: the Publishers, the Subscribers, the Broker, and the Server. The server performs as a key provider of security services. The key provider will generate the key and other parameters for security requests. The subscriber will subscribe to the message so that the publishers will provide the relevant message. The broker provides the message distribution service, which can distribute the message from the publishers to the subscriber.

First, both publisher and subscriber will register in the system to get the keys for signature. When the subscriber needs the message, he will send the topics to the broker. Publishers provide messages to brokers, who then disseminate them to subscribers. To ensure key cancellation, both the publisher and subscriber should update the keys. Each subscriber and publisher will follow the same process in one-to-many or many-to-many scenarios.

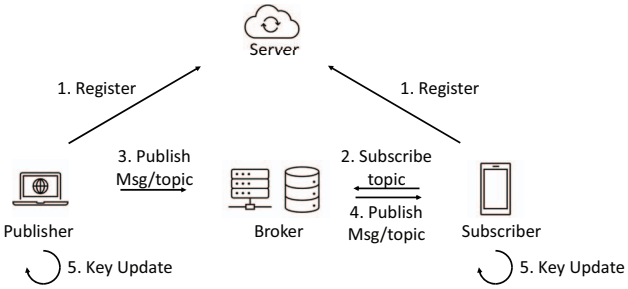


Figure 1: System Model

We also define the adversary model in the system. The publishers, subscribers, and brokers are trusted. The attacker can observe the signatures for a polynomial of numbers. Therefore, he has infinite computation resources to forge the message. Specifically, the attacker will try to modify the message from x to x' without changing the signature. Consequently, the message cannot be used for further usage by the Subscriber. Furthermore, in the data stream transmission model, the attacker may delay message transmission or change the order of the message arrival to influence the message verification. Specifically, the message x should arrive between $[T_0, T_1]$. However, the adversary may replay the same message. In the meantime, the adversary can delay the arrival time of x so that the message may be in or out of the predetermined time interval.

3 PRELIMINARIES

3.1 Message Queuing Telemetry Transport (MQTT)

MQTT is a lightweight network protocol for message transmission. The traditional IoT protocol is based on the request-respond model. The sender will send the message directly to the receiver. On the contrary, MQTT is based on the Publish-Subscribe model, which has three entities: Publisher, Subscriber, and Broker. The subscriber and publisher will not exchange the message. Instead, the broker can help distribute the message between them. Generally, there are three steps in this process.

(1) The Subscriber will subscribe to a topic and send the topic to the broker.
(2) The Publisher will publish the message with the topic to the broker.
(3) The Broker will distribute the message to the subscriber.

The Quality of Service (QoS) is an important property in MQTT. With different QoS, the number of transmissions to ensure the delivery rate of the message is different. Formally, there are three QoS.

- QoS0: The Publisher sends the message at most once. In other words, the publisher will not consider whether the subscriber has received the message.
- QoS1: The Publisher sends the message at least once. The publisher will send the message when the subscriber gives the broker an acknowledgment message (ACK).
- QoS2: The Publisher sends the message exactly once.

In QoS0, the broker disseminates the message. Therefore, the delivery rate cannot be guaranteed. In QoS1 and QoS2, the subscriber will give feedback to the broker to ensure the message is received. However, in the IoT scenario, the devices do not have much energy since batteries power them. To extend the operational lifetime of IoT devices, they have sleeping mode. In this mode, the device will not respond to incoming messages until he wakes up. Consequently, the subscriber may not send the ACK message back in time. Hence, timed security is essential in QoS1 and QoS2.

Besides the different levels of QoS, another advantage of MQTT is that it can carry a significant number of messages because the header length is small. Consequently, many MQTT service providers exist in the IoT scenario, such as EMQX, Mosquitto CLI, and Cloud MQTT.

3.2 Time-Specific Encryption

Time-specific Encryption (TSE) is proposed in [26], which is a particular type of attribute-based encryption [30]. The key is based on time intervals. Thus, it can ensure that the receiver can only decrypt the ciphertext in the predetermined time interval.

- **Setup.** This is the algorithm that takes as input the security parameter λ and time-space T and outputs the master public key (MPK) and the master secret key (MSK).
- **TIK-Ext.** This is the algorithm that takes as input the MPK, MSK, and time interval $t \in T$ and outputs the Time Instant key (TIK) k_T for the time intervals.
- **Enc.** This is the algorithm that takes as input the MPK, a message m and a decryption time interval $[T_0, T_1] \subseteq T$, and outputs the ciphertext c.
- **Dec.** This is the algorithm that takes as input the MPK, a ciphertext, and a key k_T and outputs either a message.

If the receiver wants to decrypt the ciphertext in the predetermined time interval, he should have the specific k_t (i.e., $t \in [T_0, T_1]$). On the contrary, the receiver cannot decrypt the ciphertext in t' when $t' \notin [T_0, T_1]$. he cannot succeed because he does not have the $k_{t'}$.

3.3 Dynamic Secret Sharing

Secret Sharing [31] is a cryptographic primitive that can distribute one secret into n pairs, called shadows. At least t authenticated receivers can recover the secret. Generally, the parameters in secret sharing schemes are static, such as the number of participants and the shared secret.

A dynamic secret sharing is proposed to address the challenge. When the number of participants is modified or the secret is changed, the shadows of the other participants will not be impacted. [41] introduces a dynamic secret sharing scheme, which can support the previous request. It is based on Homogeneous Linear Recursive (HLR). Each participant can choose their private shadows, while the deal can generate the public shadows according to the private ones. Therefore, the participant can verify the public shadows. Moreover, the public bulletin provides specific public information for each secret. This feature inherently enables robust support for multi-secret sharing. When the number of secrets enlarges, the dealer only needs to add more public information. When the secret is changed, the dealer can also change the public information so that the dealer does not need to modify their private shadows.

4 PROPOSED TECHNOLOGY

Timed security is critical in IoT scenarios, particularly in MQTT. On one hand, message integrity should be guaranteed within a predetermined time interval. In essence, when the message surpasses its temporal relevance, the subscriber cannot ensure message integrity. On the other hand, once the subscriber is not involved in the system, his key will not be used. This section will provide one baseline solution called Time-specific Signcryption (TSSC) and one advanced solution called Dynamic Time-specific Signature (DTSS). Table 1 shows the notations of the proposed technology. Both of them can support the key cancellation in the timed integrity problem.

Table 1: Variables List of TSSC and DTSS

Parameter	Description
x	The message from the publisher.
g, p	The generator and the prime of the cyclic group
g_D, p_D	The generator and the prime of the DSS
k_T, K_T	The private and public time interval signing keys
T	The time domain
$[T_1, T_2]$	The determined time range for the service
a	The value for key update
q, Q	The public parameters
$\mathbf{b}, \mathbf{B}$	A group of private and public shadows for DSS
$\mathbf{L}, \mathbf{y}$	The public parameters for shadow update
B_0, b_0	The random number
H	The one-way hash function
t, n	The threshold and the initial number of participants in DSS
m	The number of subscribers
γ	The random nonce

4.1 Time-specific Signcryption

In this section, we will introduce the TSSC. In Sec. 1, we mentioned how to update the key to ensure subscriber validity. The publisher will encrypt a secret value a to a ciphertext c. Then, both the publisher and the subscriber can update the key. A signcryption scheme can achieve the request in this scenario. The publisher can signcrypt the message with secret a. The subscriber unsigncrypts the message and uses a to update the key. We utilize the signcryption in [43] as an example to show how the TSSC works.

(1) **Setup.**(1^λ) The Setup algorithm is a probabilistic algorithm that takes a parameter λ and the time domain T and outputs the private and public keys for (k_T, K_T) for each time instant.
(2) **Signcrypt(k_T, x, Q, a, γ).** For the publisher, it takes the private key k_T, the message x, the public parameters Q, the key update value a, and a random value γ and outputs the signcryption $\sigma_T = \{c_T, r_T, \delta_T\}$.
(3) **Unsigncrypt(K_T, σ_T, q).** For the subscriber, when he receives the message in the predetermined time-interval, he takes the public key K_T, the public parameters q, and the corresponding signcryption σ_T as input, and outputs the received message x', the key update value a and a bit b. If $b = 1$, the integrity of the processed message can be guaranteed in this verification time instant.
(4) **KeyUpd.** For both publisher and the subscriber, they use a and the (k_T, K_T), and output the new key pairs (k'_T, K'_T).

Subsequently, we give a specific introduction to illustrate how TSSC works. In the Setup step, the key provider generates the key for signcryption. Besides, a group of parameters (q, Q) is generated. Insides, $Q = qg$. The subscriber claims the predetermined time interval during which he will receive the message. The publisher selects the corresponding keys and a random value γ according to the time interval. Then, he computes $kh = \gamma Q$, and splits kh into kh_1 and kh_2. Subsequently, he computes $\sigma_T = \{c_T, r_T, \delta_T\}$ according to Eq. 4, Eq. 5, and Eq. 6.

$$r_T = H_{kh_2}(x) \tag{4}$$

$$\delta_T = \gamma(r_T + k_T)^{-1} \tag{5}$$

$$c_T = Enc(kh_1, x||a) \tag{6}$$

Insides, *Enc* is a symmetric encryption function.

Upon receiving σ from the broker, the subscriber begins to unsigncrypt according to the time-instant he received. Based on Eq. 7, he computes kh.

$$kh = (K_T \cdot g^{r_T})^{\delta_T \cdot q} \tag{7}$$

He separates the kh into kh_1 and kh_2. Then, he uses kh_1 to decrypt the c_T by computing $x||a = Dec(kh_1, c)$ and checks whether the message is modified according to kh_2.

Finally, when the subscriber unsigncrypts the message and the publisher sends the message, they can update the key k_T and K_T separately according to Eq. 8.

$$k_T^{\text{new}} = k_T \cdot a \tag{8}$$

$$K_T^{\text{new}} = K_T \cdot a \tag{9}$$

4.1.1 Concrete Example of TSSC. To further illustrate how TSSC works, we give a running example with Fig. 2.

Example 4.1. *We apply a simple curve (e.g.,* $y^2 = x^3 + x + 1 \bmod 101$*), and the generator is* $(45, 66)$*. The topic is temperature. Suppose the time interval is* $[2, 3]$*, the key pairs of signcryption are* $((k_2, K_2) = (3, (74, 58))$ *and* $(k_3, K_3) = (2, (47, 47)))$*. In each time instant,* $\gamma_2 = 5$ *and* $\gamma_3 = 6$*. According to Sec. 4.1, the kh can be computed as* $(54, 2)$ *and* $(52, 35)$ *for different time slots. To simplify*

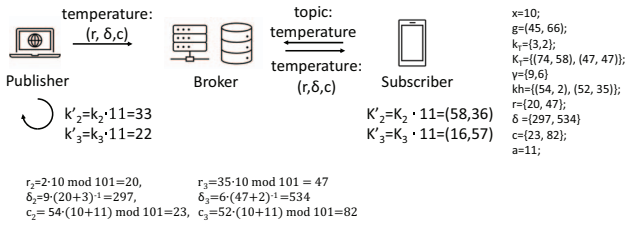


Figure 2: Running Example of TSSC

the process, we assume that the keys are the x and y coordinates of the kh (e.g., $kh_1 = 73$ and $kh_2 = 36$ in the second time instant). The hash function for computing r and c is multiplication, and the combination of m and a is addition. Therefore, the final results are $\sigma_2 = (r_2, \delta_2, c_2) = (20, 297, 23)$ and $\sigma_3 = (r_3, \delta_3, c_3) = (47, 534, 82)$.

Upon message reception, the subscriber can unsigncrypt them. He will recover the kh based on Eq. 7 if he verifies the signcryption at the second time slot. Then, he can decrypt the c_2 and check the message integrity. Subsequently, the subscriber proceeds with the key update process based on Eq. 8, which yields the new keys as follows: $((k'_2, K'_2) = (33, (58, 36)), (k'_3, K'_3) = (22, (16, 57)))$.

4.2 Dynamic Time-specific Signature

This subsection will present the Time-specific Signature Algorithm with Dynamic Secret Sharing (DTSS). A time-specific signature scheme is applied to ensure the timed integrity. Moreover, the DSS can ensure the subscriber validity. In the traditional DSS scheme, t participants can cooperatively recover the secret according to their secret shadows. The larger the t is, the more entities should participate in recovering the secret. This situation is quite different in our system. We suppose each subscriber has t shadows to recover the secret by himself. Consequently, t can be small since the number of shadows does not affect the security level [35]. When the publisher needs to update the key, he will update the public bulletin. Subsequently, the subscriber can update the key according to the information in the bulletin. This scheme generally has five steps.

(1) **Setup.**(1^λ) The Setup algorithm is a probabilistic algorithm that takes a parameter λ and the time domain T, and outputs the private and public keys for (k_T, K_T) for each time instant.
(2) **Setup**$_{\text{DSS}}(1^\lambda)$ The **Setup**$_{\text{DSS}}$ algorithm is a probabilistic algorithm that takes the private shadows $\mathbf{b} = \{b_1, b_2, \ldots, b_t\}$ and outputs $\mathbf{L} = \{L_1, L_2, \ldots, L_m\}$, $\mathbf{y} = \{y_1, y_2, \ldots, y_{n-t+1}\}$ and the public shadows $\mathbf{B} = \{B_1, B_2, \ldots, B_t\}$.
(3) **Sign(k_T, x, r).** For the publisher, it takes the private key k_T, the message x, and a random value r and outputs the signature $\sigma_T = \{r_{x_T}, \delta_T\}$.
(4) **Verify(K_T, σ_T, x).** For the subscriber, when he receives the message in the predetermined time interval, he takes the public key K_T, and the corresponding signcryption σ_T as input, and outputs a bit b. If $b = 1$, the integrity of the incoming message can be guaranteed in this verification time instant.
(5) **KeyUpd.** For the publisher, he takes a and the (k_T, K_T) as input, and outputs the new key pairs (k'_T, K'_T). For the Subscriber, he uses the public shadows $\mathbf{B}$, public parameters $(\mathbf{L}, \mathbf{y})$, and outputs the new key pairs (k'_T, K'_T). Subsequently, the publish updates the a and generates a new group of $(\mathbf{L}, \mathbf{y})$.

Subsequently, we give a specific introduction to illustrate how DTSS works. In the Setup step, the key provider generates private and public key pairs (k_T, K_T) for each time interval and stores them in the binary tree.

In the Setup$_{\text{DSS}}$ step, the key provider will prepare the parameters for each subscriber as follows.

(1) The key provider generates a t-order Homogeneous Linear Recursive (HLR) equation.

$$(x - \alpha)^t \bmod p = x^t + a_1 x^{t-1} + \ldots + a_t \bmod p_D \tag{10}$$

(2) Each subscriber selects t different integers $\mathbf{b}$ and a time interval $[T_1, T_2]$ during which the message integrity should be guaranteed. Subsequently, he sends them to the key provider as the registration. Then, the key provider generates $\mathbf{B} = \mathbf{b}g_D$ for the subscribers. In the meantime, the key provider selects a random number b_0 and computes $\mathbf{R} = b_0\mathbf{B}$ and $B_0 = b_0 g_D$.
(3) Subsequently, the key provider generates μ_i according to Eq. 11.

$$\begin{cases} \mu_i = R_i & i \le t \\ \mu_{i+t-1} + a_1\mu_{i+t-1} + \ldots + a_t\mu_i = 0 \bmod q & i > t \end{cases} \tag{11}$$

(4) Then the key provider computes $\mathbf{y}$ and $\mathbf{L}$ according to Eq. 12 and Eq. 13

$$y_i = R_i - \mu_{i-1} \text{ where } t < i \le n \tag{12}$$

$$L_i = (a - \mu_n) \oplus R_{i1} \oplus R_{i2} \oplus \ldots \oplus R_{it} \text{ where } 1 \le i \le m \tag{13}$$

The $\mathbf{L}$ and $\mathbf{y}$ are public known.

When the subscriber claims the predetermined time interval, the key provider selects corresponding (k_T, K_T) and sends them to the publisher and subscriber.

After the subscriber registers and submits the topic of the message. The publisher generates the signature $\sigma = (r_x, \delta)$ according to Eq. 14.

(1) The publisher chooses a random integer r, and computes $(r_x, r_y) = rg$. If $r_x \bmod N = 0$, then, he should change r.
(2) The publisher generates the signature according to Eq. 14. As the k_T is generated based on the time intervals, the signing process iteratively repeats.

$$\delta = r^{-1} \cdot (H(x) + k_T r_x) \tag{14}$$

(3) The publisher sends (r_x, δ) to the broker.

When the subscriber receives the message from the broker, he verifies the incoming message. Specifically, the subscriber chooses a key K_T so that $T \in [T_1, T_2]$. Then, he verifies the determined σ_T according to the following process.

(1) He computes u_1 and u_2 according to Eq. 15 and Eq. 16.

$$u_1 = H(x) \cdot \delta_T^{-1} \tag{15}$$

$$u_2 = r_x \cdot \delta_T^{-1} \tag{16}$$

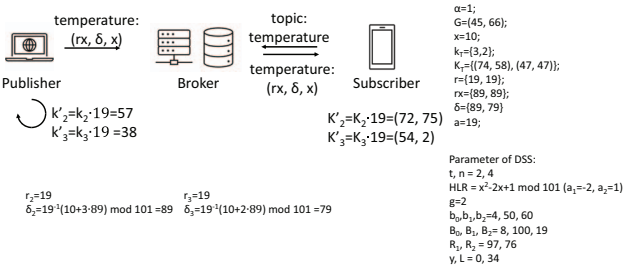


Figure 3: Running Example of DTSS

(2) The subscriber computes $r'_x = u_1 g + K_T u_2$. If r'_x is equal to r_x, the message integrity is guaranteed.

After transmitting the message, the publisher will change the main secret. Subsequently, he modifies the **L** and **y** according to Eq. 13 and Eq. 12. Since the subscriber has all the shadows, when the subscriber verifies the message, he can recover the main secret according to his shadows.

(1) Based on their private shadows, the subscriber computes $R_i = s_i B_0$. Then, he can compute μ_i according to Eq. 17.

$$\begin{cases} \mu_i = R_i & i \leq t \\ \mu_i = R_i - y_i & i > t \end{cases} \tag{17}$$

(2) According to the Lagrange interpolation, t points $(i, \mu_i/\alpha^i)$ can determinate a $t-1$ degree polynomial

$$A(x) = A_0 + A_1 x + \ldots + A_{t-1} x^{t-1} \bmod p \tag{18}$$

(3) With $A(x)$, the subscriber can get μ_i for any $i \geq 1$. Then, he can recover the secret according to Eq. 19.

$$a = L_i \oplus R_{i1} \oplus R_{i2} \oplus \ldots \oplus R_{it} + \mu_{i+t-1} \tag{19}$$

(4) When the subscriber recovers a, he can update the K_T by computing $K_T^{\text{new}} = K_T \cdot a$. The publisher can update the key by computing $k_T^{\text{new}} = k_T \cdot a$.

(5) When the publisher updates the keys, or the subscriber is out of the service, the publisher will generate a new a and update the keys **L** and **y** according to Eq. 13 and Eq. 12.

4.2.1 Concrete Example of DTSS. To further illustrate how DTSS works, we give a running example with Fig. 3.

Example 4.2. *We apply a simple curve (e.g., $y^2 = x^3 + x + 1 \bmod 101$), and a $(2,4)-$DSS. The topic is temperature. Suppose the time interval is $[2,3]$, the private and public keys DTSS are $((k_2, K_2) = (3, (74, 58))$ and $(k_3, K_3) = (2, (47, 47)))$. In the preparation phase, the key provider will generate the parameters for DSS. Specifically, the subscriber selects $b_1 = 50$ and $b_2 = 60$ as the private shadows. Then, the public shadows are $B_1 = 100$ and $B_2 = 19$. Since $b_0 = 4$, we can compute $B_0 = 8$, $R_1 = 97$, and $R_2 = 76$. Suppose the $\alpha = 1$, the HLR is equal to $x^2 - 2x + 1$. Thus we can derive the $\mu_1 = 97, \mu_2 = 76, \mu_3 = 55$, and $\mu_4 = 34$. Since there is only one subscriber, $y = 0$ and $L = 21$. According to Eq. 14, if $r = 19$, we can compute $(r_{x_2}, \delta_2) = (89, 89)$ and $(r_{x_3}, \delta_2) = (89, 79)$.*

When the subscriber receives the message, he can compute u_1 and u_2 according to Eq. 15 and Eq. 16, and verify the signature. Subsequently, the subscriber initiates a key update procedure utilizing the DSS recovery mechanism. Specifically, the subscriber computes $\mu_1 = 97$ and $\mu_2 = 76$. Therefore, he can recover $A(x) = -21x + 97 \bmod 101$. Based on Eq.19, the subscriber can compute a and update the key.

In summary, the improvement of DTSS is outlined below. First, the TSSC primarily supports offline verification. He can update the key upon unsigncrypting the received message. On the contrary, the DTSS scheme can be both online and offline. In the offline model, the publisher should send the public information (e.g., **L**) with the message to the subscriber. In the online model, the subscriber can update the key on the public bulletin. Therefore, the time consumption can be smaller than that of TSSC. When the message transmission is terminated due to a network problem, the online verification can restart since the subscriber can update the key according to the information on the public bulletin. Second, once the subscriber is no longer in the service, his key will be deactivated once the publisher stops transmitting the message in TSSC. Meanwhile, the key provider cancels the key by deleting the information on the public bulletin in the DTSS. Therefore, the publisher can actively make the subscribers out of the service. However, the TSSC still has some advantages in practical scenarios. DTSS can only protect message integrity, while the TSSC can also concurrently protect message privacy. In other words, TSSC can support time-specific privacy.

5 COMMUNICATION EFFICIENCY TIME-SPECIFIC INTEGRITY SERVICE

In this section, we introduce an improved DTSS scheme to save communication costs. Within the context of the IoT scenario, the communication channel is limited. Therefore, the signature size should be minimized. Concurrently, MQTT can work with several physical-level protocols, such as Wifi [25] and LoRa [37]. Both of them enable long-range transmission, which is costly[4]. Therefore, retransmissions should be avoided to mitigate costs.

The security level of the signature should be guaranteed during the transmission. Several factors, such as the length of the key and signature, determine the security level of the signature. For instance, different security levels in ECDSA are produced by different elliptic curves. A 64-bit security level can be achieved when secp128r1 while the 128-bit security level needs secp256k1. Communication costs are also important. Currently, there are two solutions to decrease the communication cost: aggregated signature [12] and truncated signature [29]. In aggregated signature, there will be one signature for several messages.

Definition 4. ***Aggregated Signature*** *[5]: The signer generates the signature σ_i for each messages (x_i). Then, he aggregates them and computes the final signature $\sigma = \sum \sigma_i$.*

The truncated signature means that the size of the signature will be decreased after generation.

Definition 5. ***Truncated Signature*** *[10]: The signer generates the signature $\sigma = (b_{255} b_{254} \ldots b_1 b_0)$ for each message $(b_i = \{0,1\})$. Then, he truncates the signature to $\sigma' = (b_{127} b_{126} \ldots b_1 b_0)$.*

Both schemes have limitations in the MQTT protocol. On one hand, the length of the truncated signature is small. Thus, the security level will decline. On the other hand, the delivery rate

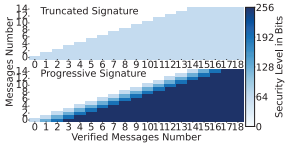


Figure 4: Security Level under Verified Messages Number

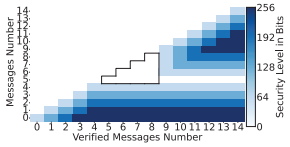


Figure 5: Resynchronization in Progressive Signature

cannot be 100% in the MQTT scenario. A single corrupted message should be viewed as a regular event [23]. Once one of the messages is missing due to a network problem, the aggregated signature cannot be verified.

Based on that, a novel progressive signature scheme is proposed. Conventionally, the signature is generated based on the private key and the message. However, the progressive signature contains a state ($S = x_{|S|}, x_{|S|-1}, \ldots, x_1$) like Eq. 20. Insides, $|S|$ is the state size. With the messages verified, the security level will progressively increase and reach the standard. Fig. 4 shows the difference between truncated and progressive signatures.

$$\sigma = Sign(sk, F(S), x) \tag{20}$$

Insides, F is a pseudorandom function. There are two main advantages in the progressive signature scheme.

(1) **Gradually Increasing Security:** The security level of the signature will reach the predetermined level when the subscriber verifies a number of messages.
(2) **Resynchronization:** It means that even if the message is missing due to a network problem, it can be restarted to verify after $|S|$ messages.

For instance, Fig. 4 shows the security level in bits of progressive signature and truncated signature. In both schemes, the key length is 128 bits so that the security level can be at least 64. In the truncated signature, each message is independently verified. Thus, the security level is 64 bits all the time. However, with the number of verified messages increasing, the security level of progressive signatures grows. When the state size becomes larger, the key for the signature becomes longer. Therefore, the security level will increase.

Fig. 5 shows the resynchronization process. In the fifth time slot, the message is missing due to the network problem. Since the state size is 4, the message verification can restart in the ninth time slot. Furthermore, the security level of the signature will progressively increase from the primary level.

Based on that, we propose an improved scheme called Pro-DTSS, which has the following steps.

- **Setup$_{\text{TS}}$** This is a determined algorithm that takes the security parameter λ and time-space T as input and outputs the master public key (MPK) and the master secret key (MSK).
- **Setup$_{\text{DSS}}$** This is a determined algorithm that takes the public shadows (**B**), the main secret (a), random number (b_0) as input, and outputs the public parameters B_0, **y** and **L**.
- **TIK-Ext** This is the algorithm that takes as input the MPK, MSK, and time interval $t \in T$, and outputs the Time Instant key (TIK) k_T^1 and k_T^2 for the time interval.
- **KeyGen** This is the algorithm that takes k_T^1 as input, and outputs the public key (K_T).
- **StateGen** This is the algorithm that takes the previous $|S|-1$ messages and a symmetric key (k_T^2) as input, and outputs the state S.
- **Sign** This is the algorithm that takes the k_T^1, a message m, S and a verifying time interval $[T_0, T_1] \subseteq T$ as input, and outputs the signature (σ).
- **Verify** This is the algorithm that takes the MPK, m, S, signature (σ) and a key k_T as input, and outputs $b = 1$ if the message has not been changed. Then it takes the private shadows **b**, the public parameters to recover the main secret
- **KeyUpd** This algorithm takes the recovered secret, the K_T, m, and S as input, and outputs the new K_t' and new state S'.

In the Signing Phase of Pro-DTSS, the publisher computes δ according to the following equation,

$$\delta = r^{-1} \cdot (H(x)F(k_T^2, S) + k_T r_x) \tag{21}$$

Insides, $F(k_T^2, S)$ is the state generation, which follows Eq. 22. Specifically, the initial state follows like S_0. When the publisher sends the message x_1, he deletes the last cells in the state and inserts the message inside so that S_1 is generated.

$$\begin{aligned} S_0 &= |x_0, x_0, \ldots, x_0| \\ S_1 &= |x_1, x_0, \ldots, x_0| \\ &\vdots \\ S_n &= |x_n, x_{n-1}, \ldots, x_{n-|S|+1}| \end{aligned} \tag{22}$$

When the signature is sent to the subscriber, he should compute u_1 according to Eq. 23. In the meantime, he should reconstruct the state, which also follows Eq. 22.

$$u_1 = H(x)F(k_T^2, S) \cdot \delta_T^{-1} \tag{23}$$

Subsequently, the rest of the steps are the same as those in DTSS. A state generation process based on a pseudorandom function can decrease the size of the signature compared with DTSS. Subscribers and publishers should share an initial state S_0 before the message transmission. A trusted third party can provide this service.

6 THEORETICAL ANALYSIS

In this section, we give a theoretical analysis of TSSC, DTSS, and Pro-DTSS, which contains time consumption and security analysis.

6.1 Time Consumption

We define modular multiplication, modular inversion, and squaring operations in elliptic curve groups as functions, MUL, INV, and SQR, and the hash function as HASH [27]. As a result, the time spent signing and verifying in DTSS is as follows:

$$t_{\text{sign}} = T((6n+2)t_{\text{MUL}} + t_{\text{INV}} + 5nt_{\text{SQR}} + t_{\text{HASH}}) \tag{24}$$

$$t_{\text{verify}} = T((12n+2)t_{\text{MUL}} + t_{\text{INV}} + 10nt_{\text{SQR}} + t_{\text{HASH}}) \tag{25}$$

Insides, n is the bit-length of multiplication operands. Apparently, we can derive the time consumption on the TSSC and Elliptic Curve

Cryptographic (ECC) scheme.

$$t_{\text{signcrypt}} = T((6n+1)t_{\text{MUL}} + t_{\text{INV}} + 5nt_{\text{SQR}} + 2t_{\text{HASH}}) \quad (26)$$

$$t_{\text{unsigncrypt}} = T((6n+6)t_{\text{MUL}} + (5n+1)t_{\text{SQR}} + 2t_{\text{HASH}}) \quad (27)$$

$$t_{\text{ECCenc}} = T((12n+2)t_{\text{MUL}} + (10n+4)t_{\text{SQR}}) \quad (28)$$

$$T_{\text{ECCdec}} = T((6n+2)t_{\text{MUL}} + (5n+4)t_{\text{SQR}}) \quad (29)$$

Therefore, we can derive that $t_{\text{sign}} < t_{\text{signcrypt}} < t_{\text{ECCenc}} + t_{\text{sign}}$ and $t_{\text{verify}} < t_{\text{unsigncrypt}} < t_{\text{ECCdec}} + t_{\text{verify}}$.

Since the key recovery is based on dynamic secret sharing, the time consumption of key update in DTSS is $O(t \log^2(t))$. t is the number of shadows. Therefore, a small value of t can minimize the time consumption of key updating, and security will not be affected.

6.2 Security Analysis

6.2.1 Unforgeability. First, we will define the unforgeability of DTSS. Since the publisher signs the message according to the different k_T, we only need to discuss the unforgeability of one of them.

Definition 6. ***(Unforgeability of DTSS)**. DTSS(Setup, Sign, Verify) is unforgeable under adaptive chosen attacks if for any efficient algorithm $\mathcal{A}$ that the experiment $Unforgeability_{\mathcal{A}}^{DTSS}$ evaluates to 1 is negligible.*

Experiment $Unforgeability_{\mathcal{A}}^{DTSS}$
-$(pk_{sign}, sk_{sign}) \leftarrow Setup(1^{\lambda})$
-$\sigma \leftarrow \mathcal{A}^{Sign(sk_{sign},\cdot)}(pk_{sign})$
-for $i = 1, 2, ..., q$, *denoted by* σ_i, *the queries to the oracle*(O) *Sign return 1 iff* $Verify(pk_{sign}, \sigma) = 1$

Specifically, the adversary can run a probabilistic algorithm in at most t steps and make no more than q queries to the Sign algorithm to forge the DTSS with the probability of success smaller than ϵ on problem instances of size k. Therefore, we have the theorem that,

Theorem 1. *The DTSS is $(\epsilon, k, q, t)-$unforgeable in the random oracle model.*

Proof. There are two situations in which the adversary can find efficient solutions to forge the ECDSA. First, he can update the key without validity shadows. Second, he can easily forge signature schemes.

In the first case, a new adversary C is built for the dynamic secret sharing recovery. In the beginning, C generates the public shadows (**B**). We set the original polynomial as $f(x)$ According to these properties, C will reconstruct a different polynomial $b(x)$ with $t-1$ degree, through which n shadows $(x_1, b(x_1)),(x_2, b(x_2)) \ldots (x_n, b(x_n))$ can interpolate. Insides, x_i are the private shadows they have. Based on $f(x)$ and $b(x)$, C can compute $c(x) = f(x) - b(x)$, which has two properties,

(1) $c(x)$ is at most $n-1$ degree polynomial and non-zero polynomial.
(2) $c(x)$ has n root.

However, these two properties are in contradiction unless $f(x)$ and $b(x)$ are equal.

In the second case, the ECDSA is a classic solution, which has strong integrity [34]. □

Concurrently, we can define the unforgeability of TSSC.

Definition 7. ***(Unforgeability of TSSC)**. DTSS(Setup, Signcrypt, Unsigncrypt) is unforgeable under adaptive chosen attacks if for any efficient algorithm $\mathcal{A}$ that the experiment $Unforgeability_{\mathcal{A}}^{TSSC}$ evaluates to 1 is negligible.*

Experiment $Unforgeability_{\mathcal{A}}^{TSSC}$
-$(pk_{signcrypt}, sk_{signcrypt}) \leftarrow Setup(1^{\lambda})$
-$\sigma \leftarrow \mathcal{A}^{Signcrypt(sk_{signcrypt},\cdot)}(pk_{signcrypt})$
-for $i = 1, 2, ..., q$, *denoted by* σ_i, *the queries to the oracle*(O)
Signcrypt return 1 iff $Unsigncrypt(pk_{signcrypt}, \sigma) = 1$

There is an important definition called Gap Diffie-Hellman (GDH).

Definition 8. ***Gap Diffie-Hellman (GDH).** Given a triple (g, g^a, g^b), find the element $C = g^{ab}$ with the help of Decision Diffie-Hellman Oracle. In other words, it can decide whether $c = ab \bmod p$ so that $g^c = C$.*

Theorem 2. *The TSSC is computationally infeasible for an adaptive attacker*

Proof. Let's prove that in contradiction. An adversary C can forge the signcryption.

Suppose C has made t number of signcryptions query $(m^{(i)})$ to the signcryption oracle. Suppose $\sigma^{(i)} = (c^{(i)}, r^{(i)}, \delta^{(i)})$ is the ith query signcryption result. Therefore, C can obtain t proof of knowledge on the input message.

Once one of the queries satisfies

$$\exists m^{(i)} \neq m, \quad \sigma^{(i)} = \sigma \quad (30)$$

C can compute $Q^{\gamma} = K_T^{q\delta^{(i)}} g^{qr\delta^{(i)}}$, since q and Q are public known, he can derive that

$$g^{q\gamma} = K_T^{q\delta^{(i)}} g^{qr\delta^{(i)}} \quad (31)$$

Subsequently, C recovers k_T so that he can construct an algorithm A which can solve the GDH problem. However, this violates the GDH assumption. □

6.2.2 Replay Attack. Formally, we give a definition of a replay attack.

Definition 9. ***Replay Attack**. A replay attack is a network attack in which valid data transmission is maliciously or fraudulently repeated or delayed.*

In this case, the theorem states that

Theorem 3. *DTSS and TSSC cannot be maliciously or fraudulently repeated or delayed.*

Proof. A new adversary $\mathcal{D}$ is built for eavesdropping on the communication cost. All the signatures generated or transmitted will be intercepted by $\mathcal{D}$. Then $\mathcal{D}$ can replay the received signatures and messages to unwanted verifiers at the other time interval.

In DTSS and TSSC, we assume k_T is relevant to the predetermined time interval, which means time is an element of the σ.

Therefore, the subscriber cannot verify the message if the message does not arrive in the predetermined time interval. Even if the replay messages can be received in a predetermined time interval, the key for signing and verifying has been changed when the public information on the bulletin changes. Therefore, the keys are invalid. □

6.2.3 Progressive Increase Security in Pro-DTSS. In Pro-DTSS, the forgery will happen in two situations.

(1) The adversary can produce a state collision.
(2) The adversary can produce a signature collision.

The second situation has been explained in Sec. 6.2.1. Consequently, we focus on the first situation. Suppose the state is generated by a $(q, t, \epsilon)-$pseudorandom function. We define a Sign-queries Game (G) for state collision. Since the security level is gradually increased based on the number of verified messages, we suppose the probability that the adversary $\mathcal{A}$ can forge the signature for the first message as follows.

$$Pr(\mathcal{A} \text{ wins G}) = c_0 + \frac{1 - c_0}{2^{|\sigma|}} \tag{32}$$

Insides, $c_i = \frac{q+i}{2^\tau}$, and τ is the state length. When the second message comes, there are still two kinds of collision. Therefore, we can derive the probability that $\mathcal{A}$ forge the signature.

$$Pr(\mathcal{A} \text{ wins G}) = c_0 + \frac{1 - c_0}{2^{|\sigma|}} \cdot (c_1 + \frac{1 - c1}{2^{|\sigma|}}) \tag{33}$$

According to the state size, we can define the forgery probability as

$$\begin{aligned} Pr(\mathcal{A} \text{ wins}) &= c_0 + \frac{c_1}{2^{|\sigma|}} + \frac{c_2}{2^{2|\sigma|}} + \ldots + \frac{c_{|S|-1}}{2^{(|S|-1)|\sigma|}} + \frac{1}{2^{|S||\sigma|}} \\ &= \sum_{i=0}^{|S|-1} \frac{c_i}{2^{i \cdot |\sigma|}} + (\frac{1}{2^\sigma})^{|S|} \end{aligned} \tag{34}$$

To find out the upper-bound of the probability, we can derive that

$$\sum_{i=0}^{|S|-1} \frac{c_i}{2^{i \cdot |\sigma|}} \leq c_0 + \frac{c_0}{2} + \frac{c_0}{2^2} + \ldots + \frac{c_0}{2^{|S|-1}} = \frac{c_0}{2(1 - (\frac{1}{2})^{|S|})} < 2c_0 \tag{35}$$

Therefore, we can derive that

$$Pr(\mathcal{A} \text{ wins}) < \frac{2q}{2^\tau} + (\frac{1}{2^{|\sigma|}})^{|S|} \tag{36}$$

which is negligible.

7 EXPERIMENTS

In this section, we will give some experiments on TSSC, DTSS, and Pro-DTSS. We use Raspberry Pi 3B with 4 CPU cores and 1GB RAM to simulate the publishers and subscribers in the practical network scenario. We choose EMQX to provide the MQTT service. The broker is 'broker.emqx.io', which is a public broker for experiments, and the port number is 1883. The code is implemented in python3.

We compare the proposed schemes with the sign-then-encrypt scheme, which is the combination of ECC [18] and ECDSA [15]. The key length is 160 bits. We use the perfect binary tree, which has 1024 leaf nodes. In DTSS, the number of shadows for key updating is 2. All the parameters, including key length, prime, and the random value

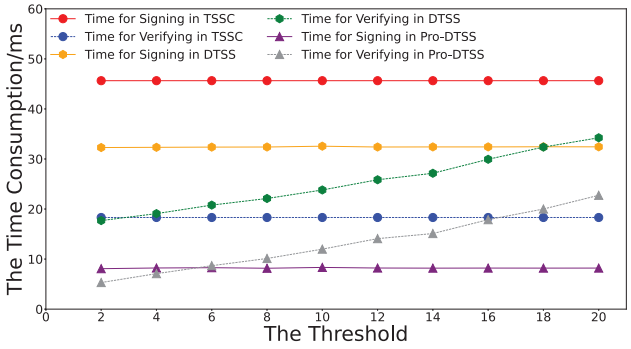


Figure 6: Time Consumption under Different Threshold

for the key update for TSSC and DTSS are 160 bits. In the HLR, α is equal to 1. There are two different datasets in the experiment. One is the synthetic dataset. The number of messages is 1000, which are randomly generated and not correlated. The other is the practical dataset, "Real-world IoT data for environmental analysis," from Kaggle. The dataset contains about 390000 messages gathered by an ESP32 microcontroller. The dataset has different topics, such as temperature and humidity, which can be directly applied to the MQTT protocol. We use the Raspberry Pi to simulate the IoT devices as the subscribers in sleeping mode. Consequently, the message integrity is essential. In sleeping mode, the IoT devices will not give feedback and verify the message. On one hand, the publisher should guarantee the message integrity during the predetermined time. On the other hand, if the subscriber does not gather the messages in time, his key cannot be used to verify the messages.

7.1 The Impact of Threshold on TSSC, DTSS, and Pro-DTSS

In Fig. 6, we compare the time for signing and verifying when the threshold increases from 2 to 20. The y-axis is the CPU time consumption. We apply secp256k1 as the curve for TSSC and DTSS, and secp128r1 for Pro-DTSS. Since TSSC does not have shadows, there is no effect on that. According to Sec. 6.1, we mentioned that the time consumption for key updating is $O(t \log^2 t)$, which is monopoly increasing with the t grows up. Since the publisher can directly generate the secret value for key updating, the time for signing will not change. The time consumption of Pro-DTSS in signing and verifying is smaller than that in DTSS. As we mentioned in Sec. 4.2, the security level will not be affected since all the shadows are obtained by the subscriber. We use 2 as the threshold in DTSS and Pro-DTSS in the rest of the experiments.

7.2 The Quantitative Analysis of TSSC, DTSS, and Pro-DTSS

We will show the quantitative analysis of the three schemes in Raspberry Pi in 5 minutes.

According to Table 2, the maximized CPU utility of the publisher and subscriber occurs in the key preparation phase. Specifically, the maximized CPU utility and the memory occupation of publisher and subscriber in TSSC are smaller than the other two schemes. Besides the key preparation for signing and verifying, DTSS and

Table 2: The Quantitative Analysis of Publisher and Subscriber

Scheme	CPU Utility (Max)		CPU Utility (Avg)		Memory Occupation/Mb	
	Publisher	Subscriber	Publisher	Subscriber	Publisher	Subscriber
TSSC	25.0%	27.5%	10.1%	4.5%	22.6	25.2
DTSS	29.0%	30.3%	9.4%	4.5%	43.7	47.1
Pro-DTSS	27.6%	27.6%	8.0%	3.5%	43	55.4

Table 3: The Replay and Delay Attack Detection of Subscriber

Scheme	Replay Attack Times	Replay Attack Detection	Delay Attack Times	Delay Attack Detection
TSSC	8	8	12	6
DTSS	11	11	8	5
Pro-DTSS	14	14	9	6

Pro-DTSS should also load the shadows for DSS. According to Sec. 6.1, the DTSS is more lightweight than TSSC. Therefore, the average CPU utility in DTSS becomes smaller than that in TSSC.

7.3 The Replay and Delay Attack Detection of TSSC, DTSS, and Pro-DTSS

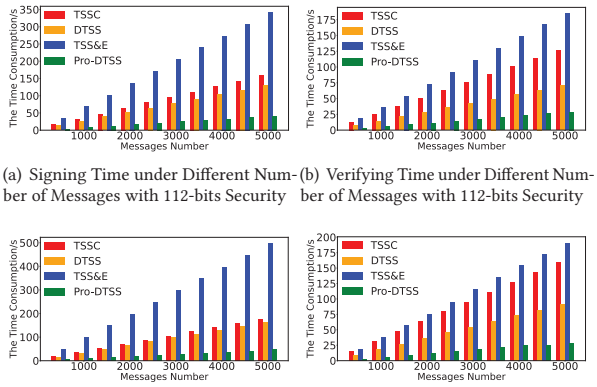
We show how our proposed schemes can detect the replay and delay attack. Suppose there are 10000 messages, and the possibility of a replay attack and delay attack is 0.1%. The predetermined time interval is $[2, 12]$.

Table 3 shows the number of replay and delay attacks during the message transmission and the number of attack detections. For instance, there are 8 replay attacks in TSSC schemes, and all of them can be detected. For the replay attack, the key will update upon receiving the message. It can play a role as the timestamp. If replay attacks occur, the key cannot be applied to verify the following incoming message. Therefore, the replay attack can be detected. Furthermore, the TSSC can only verify the rest of the messages if the subscriber registers again. The reason is that the parameters for key updating are transmitted together with the message. If the replay attack happens, the subscriber cannot receive the latest parameters for key updating.

On the contrary, there will be two situations when the delay attack happens. First, if the delay attacks do not make the message exceed the predetermined time interval, the delay attack cannot be detected. For example, the subscriber receives the message at 10th. time-instant after the delay attack. The reason is that the subscriber can still own the key for verification. Second, if the delay attacks make the message exceed the predetermined time interval (e.g., $[2, 12]$), the message cannot be verified, and the delay attack can be detected. For example, the subscriber receives the message at 14th. time-instant after the delay attack. The reason is that the key for verification is outdated. Therefore, our proposed schemes can solve the replay attack and prevent the delay attack to some degree.

7.4 The Performance of Four Schemes in EMQX

We apply our proposed schemes to EMQX, a practical MQTT service provider in real-world scenarios with the practical datasets. The experiments contain one-to-one, one-to-many, and many-to-many scenarios. Without further mentioning, we use secp256k1 for the



(a) Signing Time under Different Number of Messages with 112-bits Security
(b) Verifying Time under Different Number of Messages with 112-bits Security
(c) Signing Time under Different Number of Messages with 128-bits Security
(d) Verifying Time under Different Number of Messages with 128-bits Security

Figure 7: Time Consumption of Publisher and Subscriber under Different Curves and The Different Number of Messages

TSSC, DTSS, and TSS&E, and secp128r1 for Pro-DTSS since they provide the same security level.

7.4.1 The Time Consumption under Different Curves and The Number of Messages. In Fig. 7, we compare four schemes for the publisher and subscriber practically. The x-axis is the number of messages sent and received by the publisher and subscriber, increasing from 500 to 5000, and the y-axis is the CPU time consumption. To ensure the same security level, we employ secp128r1 and secp112r1 for Pro-DTSS and secp256k1 and secp224k1 for the rest three schemes. Furthermore, the publisher sends the message every 0.2 seconds.

It is evident that the utilization of the sign-then-encrypt approach in both publisher and subscriber has a larger time consumption than our proposed schemes. Specifically, TSSC reduces about 55% of time in the secp224k1 curve and about 15% of time for the subscriber secp256k1 curve. In the meantime, DTSS can save 62% of time in the secp224k1 curve and 52% of time in the secp256k1 curve. Similarly, the TSSC and DTSS can save 55% and 61% of time for the publisher in the secp224k1 curve and save 65% and 68% of time in the secp256k1 curve. In the case of TSSC, the time for signcryption is shorter than sign-then-encrypt. Therefore, the time of signing can decrease. Conversely, the DTSS only encompasses key updating and signing processes. Hence, the time is shorter than the other two schemes. From the subscriber's perspective, the time consumption for unsigncryption is less than decrypt-then-verify, thereby saving more time. On the contrary, the number of shadows is small in DTSS schemes; thus, the time consumption for secret recovery is small. The result can prove the deduction in Sec. 6.1 is accurate. Furthermore, since the Pro-DTSS applies a lightweight curve to reach the same security level, the time consumption is much smaller than TSS&E, which is about 90% on the publisher and 85% on the subscriber.

7.4.2 The Throughput under The Time Consumption. In this experiment, we compare the system's throughput (The number of messages received by the subscriber) from 60 seconds to 600 seconds. According to Fig. 8, the throughput of TSSC and DTSS are

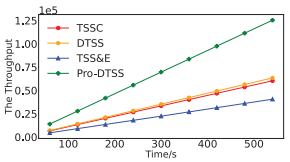


Figure 8: Throughput

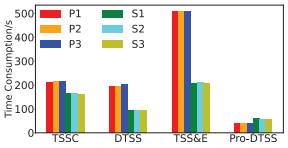
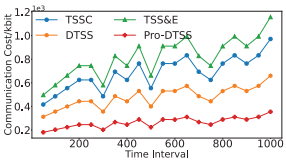
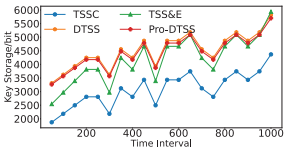


Figure 9: Many-to-Many Scenario



(a) Communication Cost



(b) Key Storage

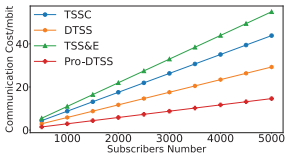
Figure 10: Communication Cost and Key Storage under Time Range

1.46 and 1.56 times larger than that in TSS&E. Furthermore, the throughput of the Pro-DTSS is the largest among all the schemes, which is two times larger than that in DTSS. The reason is that the time for signing and verifying is the smallest among the four schemes. Therefore, within the same time interval, the throughput of Pro-DTSS is the largest, and the DTSS is the second largest one.

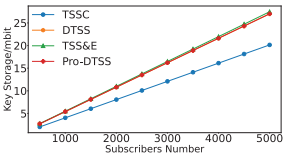
7.4.3 The Time Consumption and Communication Cost under many-to-many scenario. We will show the performance of the four schemes in the many-to-many scenarios. Since the dataset contains three different topics, we assume that there are three subscribers (e.g., $S1$, $S2$, $S3$) who need 5000 messages with distinct topics from three publishers (e.g., $P1$, $P2$, $P3$).

In Fig. 9, all our proposed schemes consume less time than TSS&E, and the time consumption of the Pro-DTSS is the smallest among the four schemes. According to Table 2, the average CPU usage is small after the key generation. Consequently, the three publishers' and subscribers' time consumption is similar.

7.4.4 The Communication Cost and Key Storage under The Time Interval. In this experiment, we compare the four schemes when the predetermined time interval increases. The number of messages is 100. The x-axis is the predetermined time interval for guaranteeing the message integrity. We separate the time domain into 1024 time slots, and the starting time-instant is 20. According to Fig. 10(a) and Fig. 10(b), communication cost and key storage will increase when the time interval becomes large. Specifically, when the time interval becomes 300, which is $[20, 320]$, there will be 7 k_T, (e.g., $k_{[20,23]}$, $k_{[24,31]}$, $k_{[32,63]}$, $k_{[64,127]}$, $k_{[128,255]}$, $k_{[256,319]}$ and k_{320}). However, when the time interval is 250, the number of k_T and K_T is 9 according to Sec. 3.2. Therefore, the communication cost and key storage decrease. The communication of the DTSS is smaller than that of the TSSC and TSS&E. Furthermore, the Pro-DTSS can save communication costs, which proves our assumption. According to Sec. 4.1, except the signature, there is some ciphertext whose number is equal to the number of k_T. While in DTSS and Pro-DTSS, there is only one message with several signatures.



(a) Communication Cost



(b) Key Storage

Figure 11: Communication Cost and Key Storage under Subscribers Number

On the contrary, DTSS and Pro-DTSS should store the key for signature and the associated shadows for key updating. TSSC and TSS&E update the key directly when the subscriber receives the message. Therefore, the key storage is small. Additionally, it is noteworthy that the key length in Pro-DTSS is generally smaller than that in DTSS. Thus, the key storage for Pro-DTSS is small.

7.4.5 The Communication Cost and Key Storage on One-to-Many Scenario. We show the performance of a one-to-many scenario. The number of subscribers increases from 500 to 5000.

According to Fig. 11, both communication cost and the key storage are linearly increasing when the number of subscribers rises. The reason is that each subscriber should have different keys to verify the message. Generally, the communication costs of TSSC and TSS&E are 1.5 and 1.87 times larger than DTSS. In the meantime, the communication cost of DTSS is about two times larger than that of Pro-DTSS. As for the key storage, TSSC can save about 26% storage. Compared with TSSC, both DTSS and Pro-DTSS should store shadows for DSS. Therefore, when the number of subscribers becomes large, the number of shadows in the system will increase. Although each subscriber only needs to store his own shadows, the publisher should contain all the shadows.

In summary, DTSS and Pro-DTSS offer the advantage of reducing communication costs while enlarging the key storage size. The shadows for key updating remain constant, while the key for verifying is subject to change. Although our proposed schemes can be directly applied in one-to-many and many-to-many scenarios, they confront the bottleneck in key storage. However, the trade-off is acceptable within practical contexts.

8 RELATED WORK

In this section, we provide an overview of about the existing works, which contains the privacy and integrity, and the timed cryptography.

8.1 Privacy and Integrity

During message transmission, both message privacy and integrity are essential. Most of the works focus on either of them. Encryption is the most common scheme to ensure privacy. To solve the integrity problem, the message authentication code (MAC) [39] and digital signatures [22]. In [2], the authors proposed a block-chaining message authentication code, which divided the message into several blocks and generated MAC for each of them. In [39], the authors provided a partial verification message authentication

code so that the verifier only needs one MAC to verify any subset of the message. Compared with MAC schemes, the signature can provide some extra properties. For instance, [19] provided a 3-move undeniable signature scheme that is secure against active and concurrent attacks. [17] provided a blind signature, which supports boosts security. Therefore, the verifier cannot know or derive the identity of the signer. However, traditional MACs or signature schemes did not consider the privacy problems.

In [13], the authors provided a Prefix-verifiable Message Authentication Code (PMAC) scheme for location privacy and integrity. Subsequently, the authors proposed a Scalable Prefix-verifiable Message Authentication Code (SPMAC) to achieve the target in the IoT scenario in [40]. On one hand, the message authentication code naturally assured integrity. On the other hand, it can protect some secret information from being disclosed by the unauthorized party. However, these works did not consider the key cancellation problem. Another solution to concurrently solve privacy and integrity is signcryption, which was proposed in [43], based on ECDSA. Compared with encrypt-then-sign or sign-then-encrypt, the signcryption scheme had better efficiency. Based on that, [24] proposed a new signcryption scheme for the CSUAC-IoT scenario. In [38], the authors provided a novel signcryption scheme for edge computing with blockchain. The identity-based signcryption scheme is proposed in [42]. Currently, there is no work focused on the timed integrity problem in the MQTT protocol.

8.2 Timed Cryptography

Timed Cryptography is an important cryptographic primitive first proposed in [32]. Based on that, [7] proposed the timed-released encryption, which allowed the receiver to decrypt the message after a determined time T. Moreover, the time-specific encryption [26] allowed the receiver to decrypt the ciphertext in the time range $[T_1, T_2]$. Moreover, in [16], the authors provided a time-specific signature so the signer can sign the message in the determined time range. Besides, other schemes can ensure timed security before a predetermined time T. In [3][36], the authors provided a verifiable delay function (VDF), which is hard to compute while easy to verify. Even a parallel computing paradigm cannot speed up the computation process. Therefore, VDF can be used as the proof of replication. Specifically, the participant should spend a predetermined time computing the function. Therefore, if he can give the feedback in time, he will have no more time to conduct the same function. In [1], the authors used the function to prevent the denial of service attack. [36] propose an efficient trapdoor VDF so that a participant with the trapdoor key can speed up the computation process. For the other participants, he still needs to spend the predetermined time to compute the result. In [20], a latticed-based sequential function leads to a simple and efficient proof of sequential work. However, the existing schemes did not consider the key cancellation problem.

9 CONCLUSION

In this paper, we propose a novel problem in the MQTT scenario called timed integrity with key cancellation, which has two components: message integrity and validity, as well as subscriber validity. To solve the problem, we provide one baseline scheme called Time-specific Signcryption and one advanced scheme called Dynamic Time-specific Signature. Moreover, a communication efficient scheme called Pro-DTSS is proposed to save communication costs in the time stream scenario.

Besides being applied on the subscriber side, our scheme can also be applied on the publisher side. In this situation, the subscriber can ensure the publisher signs the messages in the predetermined time interval. One of the applications is the anonymous questionnaire. For instance, a company conducts a survey on target production across different age groups, and the participants can sign the message using the age-appropriate time-instant key. Moreover, our schemes can also be applied in one-to-many scenarios, which means several subscribers request the message concurrently. If the message topics are different, the publisher should sign them separately. On the contrary, unless all the subscribers' working time does not overlap, the time for signing in the publisher will not be proportional to the number of subscribers. Moreover, our scheme can also support the many-to-many network.

In the future, we want to solve the problem of decreasing the number of keys in TSSC and DTSS in many-to-many scenarios. When our schemes are directly applied in many-to-many scenarios, each publisher and subscriber will share a group of keys, which leads to significant storage costs. Since IoT devices do not have large memory for storage, the current solution may confront the bottleneck. How to organize the key distribution is a great problem we want to solve.

ACKNOWLEDGEMENT

This work was supported by the National Natural Science Foundation of China (Grant No: 92270123, 62072390 and 62102334), and the Research Grants Council, Hong Kong SAR, China (Grant No: 15218919, 15203120 and 15209922).

REFERENCES

[1] Vidal Attias, Luigi Vigneri, and Vassil Dimitrov. 2020. Preventing denial of service attacks in IoT networks through verifiable delay functions. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*. IEEE, 1–6.

[2] Mihir Bellare, Joe Kilian, and Phillip Rogaway. 2000. The security of the cipher block chaining message authentication code. *J. Comput. System Sci.* 61, 3 (2000), 362–399.

[3] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. 2018. Verifiable delay functions. In *Advances in Cryptology–CRYPTO 2018: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I*. Springer, 757–788.

[4] Gonglong Chen, Yihui Wang, Huikang Li, and Wei Dong. 2019. TinyNET: a lightweight, modular, and unified network architecture for the internet of things. In *Proceedings of the ACM SIGCOMM 2019 conference posters and demos*. 9–11.

[5] Xue Chen, Shiyuan Xu, Yunhua He, Yu Cui, Jiahuan He, and Shang Gao. 2022. LFS-AS: lightweight forward secure aggregate signature for e-health scenarios. In *ICC 2022-IEEE International Conference on Communications*. IEEE, 1239–1244.

[6] Yuanyi Chen, Mingxuan Zhou, Zengwei Zheng, and Dan Chen. 2019. Time-aware smart object recommendation in social internet of things. *IEEE Internet of Things Journal* 7, 3 (2019), 2014–2027.

[7] Sherman SM Chow, Volker Roth, and Eleanor G Rieffel. 2008. General certificateless encryption and timed-release encryption. In *Security and Cryptography for Networks: 6th International Conference, SCN 2008, Amalfi, Italy, September 10-12, 2008. Proceedings 6*. Springer, 126–143.

[8] Flaviu Cristian and Christof Fetzer. 1999. The timed asynchronous distributed system model. *IEEE Transactions on Parallel and Distributed systems* 10, 6 (1999), 642–657.

[9] Jesús E Díaz-Verdejo, Rafael Estepa Alonso, Antonio Estepa Alonso, and German Madinabeitia. 2022. A critical review of the techniques used for anomaly detection of HTTP-based attacks: taxonomy, limitations and open challenges. *Computers & Security* (2022), 102997.

[10] Adeline Fermanian. 2022. Functional linear regression with truncated signatures. *Journal of Multivariate Analysis* 192 (2022), 105031.

[11] Xiang Gong and Tao Feng. 2022. Lightweight anonymous authentication and key agreement protocol based on CoAP of Internet of Things. *Sensors* 22, 19 (2022), 7191.

[12] Rishab Goyal and Vinod Vaikuntanathan. 2022. Locally verifiable signature and key aggregation. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*. Springer, 761–791.

[13] Haibo Hu, Qian Chen, Jianliang Xu, and Byron Choi. 2017. Assuring spatio-temporal integrity on mobile devices with minimum location disclosure. *IEEE Transactions on Mobile Computing* 16, 11 (2017), 3000–3013.

[14] Åsmund Hugo, Brice Morin, and Karl Svantorp. 2020. Bridging MQTT and Kafka to support C-ITS: A feasibility study. In *2020 21st IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 371–376.

[15] Sangwon Hyun, Peng Ning, An Liu, and Wenliang Du. 2008. Seluge: Secure and dos-resistant code dissemination in wireless sensor networks. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*. IEEE, 445–456.

[16] Masahito Ishizaka and Shinsaku Kiyomoto. 2020. Time-specific signatures. In *Information Security: 23rd International Conference, ISC 2020, Bali, Indonesia, December 16–18, 2020, Proceedings*. Springer, 20–38.

[17] Jonathan Katz, Julian Loss, and Michael Rosenberg. 2021. Boosting the security of blind signature schemes. In *Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part IV 27*. Springer, 468–492.

[18] Pardeep Kumar, Andrei Gurtov, and Phuong H Ha. 2016. An efficient authentication model in smart grid networks. In *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 1–2.

[19] Kaoru Kurosawa and Swee-Huay Heng. 2005. 3-move undeniable signature scheme. In *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*. Springer, 181–197.

[20] Russell WF Lai and Giulio Malavolta. 2023. Lattice-Based Timed Cryptography. In *Annual International Cryptology Conference*. Springer, 782–804.

[21] Loukas Lazos, Radha Poovendran, and Srdjan Capkun. 2005. ROPE: Robust position estimation in wireless sensor networks. In *IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005.* IEEE, 324–331.

[22] An Liu and Peng Ning. 2008. TinyECC: A configurable library for elliptic curve cryptography in wireless sensor networks. In *2008 International Conference on Information Processing in Sensor Networks (ipsn 2008)*. IEEE, 245–256.

[23] Jorge E Luzuriaga, Miguel Perez, Pablo Boronat, Juan Carlos Cano, Carlos Calafate, and Pietro Manzoni. 2016. Improving mqtt data delivery in mobile scenarios: Results from a realistic testbed. *Mobile Information Systems* 2016 (2016).

[24] Shobhan Mandal, Basudeb Bera, Anil Kumar Sutrala, Ashok Kumar Das, Kim-Kwang Raymond Choo, and Youngho Park. 2020. Certificateless-signcryption-based three-factor user access control scheme for IoT environment. *IEEE Internet of Things Journal* 7, 4 (2020), 3184–3197.

[25] John Miller, Elahe Soltanaghai, Raewyn Duvall, Jeff Chen, Vikram Bhat, Nuno Pereira, and Anthony Rowe. 2022. Cappella: Establishing Multi-User Augmented Reality Sessions Using Inertial Estimates and Peer-to-Peer Ranging. In *2022 21st ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 428–440.

[26] Kenneth G Paterson and Elizabeth A Quaglia. 2010. Time-specific encryption. In *Security and Cryptography for Networks: 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings 7*. Springer, 1–16.

[27] Jonathan Petit. 2009. Analysis of ecdsa authentication processing in vanets. In *2009 3rd international conference on new technologies, mobility and security*. IEEE, 1–5.

[28] Julien Ponge, Boualem Benatallah, Fabio Casati, and Farouk Toumani. 2010. Analysis and applications of timed service protocols. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 19, 4 (2010), 1–38.

[29] Thomas Pornin. 2022. Truncated EdDSA/ECDSA Signatures. *Cryptology ePrint Archive* (2022).

[30] Marco Rasori, Michele La Manna, Pericle Perazzo, and Gianluca Dini. 2022. A survey on attribute-based encryption schemes suitable for the internet of things. *IEEE Internet of Things Journal* (2022).

[31] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[32] C Timothy. 1993. May. *Time-release crypto* (1993).

[33] Roman Trüb, Daniel Moser, Matthias Schäfer, Rui Pinheiro, and Vincent Lenders. 2018. Monitoring meteorological parameters with crowdsourced air traffic control data. In *2018 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 25–36.

[34] Serge Vaudenay. 2002. The security of DSA and ECDSA: Bypassing the standard elliptic curve certification scheme. In *Public Key Cryptography—PKC 2003: 6th International Workshop on Practice and Theory in Public Key Cryptography Miami, FL, USA, January 6–8, 2003 Proceedings 6*. Springer, 309–323.

[35] Na Wang, Yuanyuan Cai, Junsong Fu, and Xiqi Chen. 2020. Information privacy protection based on verifiable (t, n)-Threshold multi-secret sharing scheme. *IEEE Access* 8 (2020), 20799–20804.

[36] Benjamin Wesolowski. 2019. Efficient verifiable delay functions. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part III 38*. Springer, 379–407.

[37] Lei Xie, Bing Xie, and Yuming Qi. 2022. Research on remote intelligent monitoring system based on MQTT and LoRa communication for the five-axis NC machine tool. In *2022 IEEE Conference on Telecommunications, Optics and Computer Science (TOCS)*. 1462–1466. https://doi.org/10.1109/TOCS56154.2022.10016059

[38] Guangxia Xu, Jingnan Dong, Chuang Ma, Jun Liu, and Uchani Gutierrez Omar Cliff. 2022. A certificateless signcryption mechanism based on blockchain for edge computing. *IEEE Internet of Things Journal* (2022).

[39] Haotian Yan, Haibo Hu, and Qingqing Ye. 2023. Partial Message Verification in Fog-based Industrial Internet of Things. *Computers & Security* (2023), 103530.

[40] Haotian Yan, Haibo Hu, Qingqing Ye, and Li Tang. 2022. SPMAC: Scalable Prefix Verifiable Message Authentication Code for Internet of Things. *IEEE Transactions on Network and Service Management* 19, 3 (2022), 3453–3464.

[41] Jiangtao Yuan and Lixiang Li. 2019. A fully dynamic secret sharing scheme. *Information Sciences* 496 (2019), 42–52.

[42] Yanan Zhao, Yunpeng Wang, Yuhao Liang, Haiyang Yu, and Yilong Ren. 2022. Identity-based broadcast signcryption scheme for vehicular platoon communication. *IEEE Transactions on Industrial Informatics* (2022).

[43] Yuliang Zheng. 1997. Digital signcryption or how to achieve cost (signature & encryption) cost (signature)+ cost (encryption). In *Advances in Cryptology—CRYPTO'97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17–21, 1997 Proceedings 17*. Springer, 165–179.