

# Formalization and Verification of MQTT-SN Communication Using CSP

Wei Lin , Sini Chen, Huibiao Zhu<sup>[0000-0002-0214-8565]</sup> (✉)

Shanghai Key Laboratory of Trustworthy Computing,  
East China Normal University, Shanghai, China  
`hbzhu@sei.ecnu.edu.cn`

**Abstract.** The MQTT-SN protocol is a lightweight version of the MQTT protocol and is customized for Wireless Sensor Networks (WSN). It removes the need for the underlying protocol to provide ordered and reliable connections during transmission, making it ideal for sensors in WSN with extremely limited computing power and resources. Due to the widespread use of WSN in various areas, the MQTT-SN protocol has promising application prospects. Furthermore, security is crucial for MQTT-SN, as sensor nodes applying this protocol are often deployed in uncontrolled wireless environments and are vulnerable to a variety of external security threats.

To ensure the security of the MQTT-SN protocol without compromising its simplicity, we introduce the ChaCha20-Poly1305 cryptographic authentication algorithm. In this paper, we formally model the MQTT-SN communication system using Communicating Sequential Process (CSP) and then verify seven properties of this model using Process Analysis Toolkit (PAT), including deadlock freedom, divergence freedom, data reachability, client security, gateway security, broker security, and data leakage. According to the verification results in PAT, our model satisfies all the properties above. Therefore, we can conclude that the MQTT-SN protocol is secure with the introduction of ChaCha20-Poly1305.

**Keywords:** MQTT-SN Protocol · Communicating Sequential Process (CSP) · Formal Methods · Modeling · Verification.

## 1 Introduction

Wireless sensor nodes in wireless sensor networks (WSN) are characterized by their small size, ease of deployment, and low cost, making WSN widely used in various fields, such as real-time intelligent monitoring and hazardous zone operations [1, 2]. MQTT-SN protocol (Message Queuing Telemetry Transport for Wireless Sensor Networks) is a topic-based publish-subscribe message transmission protocol designed by IBM specifically for WSN [3]. It is a lightweight and resource-efficient version of the MQTT protocol. Compared to the MQTT protocol, which requires the underlying protocol to provide ordered and reliable connections during data transmission, the MQTT-SN protocol eliminates these

requirements. As a result, it is more suitable for sensor nodes in WSN with extremely limited energy, computing capacity, storage capacity, and bandwidth.

In addition, wireless sensor nodes are usually deployed in uncontrollable and open wireless environments where external security threats are inevitable [4]. At the same time, sensitive data that is not intended to be accessed by outsiders is often transmitted between wireless sensor nodes. Therefore, it is important to investigate whether MQTT-SN communication can meet the reliability and security requirements for data transmission in WSN.

Several studies have analyzed and tested the communication mechanism of the MQTT-SN protocol. For example, Park et al. [5] standardized the generation, distribution, and registration of security certificates. They then proposed a secure MQTT-SN protocol communication architecture and tested the performance of this architecture by building a simulation scenario. Roldán-Gómez et al. [6] constructed an MQTT-SN protocol communication network and simulated a series of attack behaviors to test the security of the protocol, comparing it with communication in an environment without attacks. Diwan et al. [7] used Event-B to propose an abstract model for the MQTT, MQTT-SN, and CoAP, and verified their common properties.

It can be seen that most studies have conducted experiments by building environments to simulate actual application scenarios or attacks, collecting experimental data, and analyzing the security of communication mechanisms based on the data. However, experiments may be affected by many external factors. Potential security issues in MQTT-SN communication may exist but have not been discovered.

As the MQTT-SN protocol itself does not specify any security mechanism to maintain its simplicity, this paper introduces a lightweight encryption and authentication algorithm, ChaCha20-Poly1305 [8–10], as a security guarantee for MQTT-SN. In this paper, we adopt a classical formal method, Communicating Sequential Process (CSP) [11], to construct models for entities involved in MQTT-SN communication. We also introduce the intruder which can intercept and fake messages to simulate real-world attacks. After that, we use the model checking tool Process Analysis Toolkit (PAT) [12, 13] to verify seven properties with the interference of the intruder, including deadlock freedom, divergence freedom, data reachability, client security, gateway security, broker security, and data leakage. The verification results show that the reliability and security of the MQTT-SN protocol communication are ensured with the introduction of ChaCha20-Poly1305.

The structure of this paper is organized as follows. Section 2 provides a brief introduction to the communication mechanism of the MQTT-SN protocol, the process algebra CSP, and the verification tool PAT. In section 3, we present the detailed modeling process for the main entities in our model. In section 4, we implement the constructed model using PAT and verify seven properties. Section 5 concludes the paper and gives a discussion about further improvement.

## 2 Background

In this section, we start with the MQTT-SN architecture and a brief explanation of its communication mechanism. We also give a brief introduction to CSP and PAT.

### 2.1 MQTT-SN Architecture

The communication architecture of the MQTT-SN protocol is shown in Fig.1. There are four main entities in MQTT-SN communication: clients, gateways, forwarders, and brokers [3].

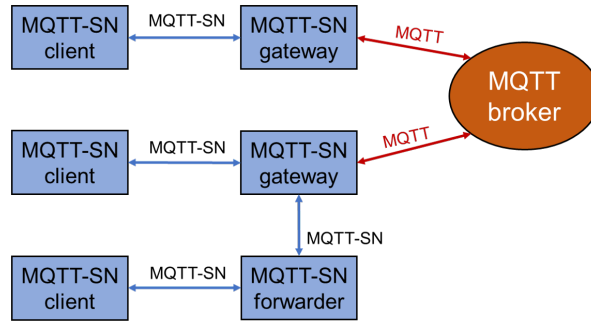


Fig. 1. MQTT-SN Architecture

- **Clients:** Clients can be divided into two types of roles, namely publishers and subscribers. Clients who publish messages with topics are called publishers, while clients who subscribe to topics are called subscribers.
- **Gateways:** The communication between clients and gateways follows the MQTT-SN protocol, while the communication between gateways and brokers adopts the MQTT protocol. The main function of the gateway is to adjust the format of data packets and forward them after protocol conversion between MQTT-SN and MQTT. The gateway may be integrated into the broker server or may exist independently. As the function of the gateway is independent of that of the broker, the gateway is assumed to be an independent module in the subsequent modeling part.
- **Forwarders:** When the gateway cannot directly connect to the network where the client is located, a forwarder is needed. The forwarder functions basically the same as the gateway. Therefore, it is omitted for simplicity in the subsequent modeling part.
- **Brokers:** All clients need to be connected to the broker via a gateway to achieve topic-based message exchange, rather than communicating with each other directly. The main function of brokers is to receive messages from publishers and distribute these messages to the appropriate subscribers.

## 2.2 Communication Mechanism of the MQTT-SN Protocol

The MQTT-SN protocol itself does not specify security mechanisms in order to maintain its lightweight characteristics. Since a majority of the MQTT-SN clients do not possess the ability to process and store complex data, there are higher efficiency requirements. Therefore, this paper adopts ChaCha20-Poly1305 [8–10], an efficient and lightweight cryptographic authentication algorithm, to ensure the security of MQTT-SN communication.

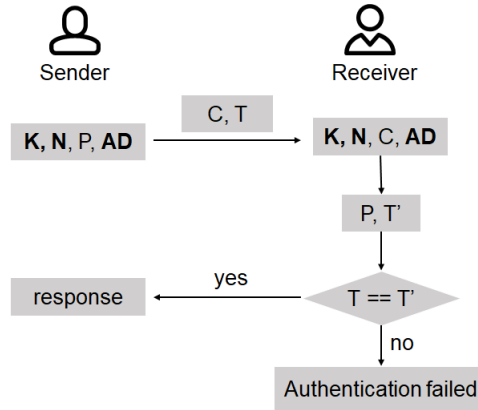


Fig. 2. Mechanism of ChaCha20-Poly1305

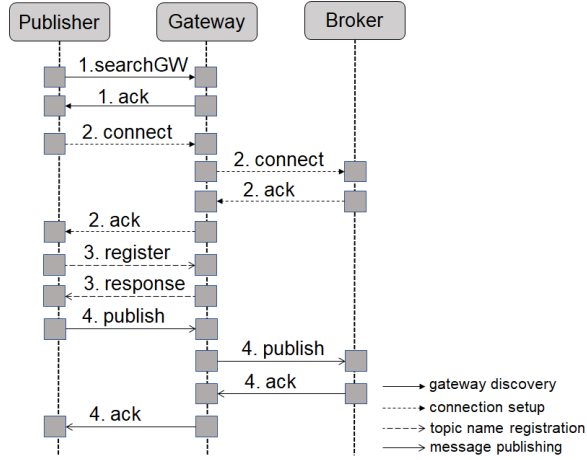
The workflows of ChaCha20-Poly1305 are shown in Fig.2. The algorithm requires the secret key  $K$ , the interference term  $N$ , the plaintext  $P$ , and the associated data  $AD$  as input. As an assumption of this paper, the sender and the receiver need to generate the shared  $K$ ,  $N$ , and  $AD$  that are kept confidential from others using a secure exchange algorithm. The sender computes the ciphertext  $C$  and the authentication tag  $T$  with the following two steps and sends them to the receiver:

- Use  $K$  and  $N$  to produce a stream of bytes that is XORed with  $P$ . The result of the XOR operation is  $C$ .
- Hash  $P$  with  $K$ , and then combine the hash value with  $N$ . The result is  $T$ .

After receiving these messages, the receiver needs to compute a new plaintext  $P$  and a new authentication tag  $T'$ . Only when the  $T$  and the  $T'$  are equal, the receiver considers that the sender can pass the identity authentication.

The two most important entity behaviors in MQTT-SN communication are publishing data with topics by publishers and subscribing to topics by subscribers. Because these two behaviors result in similar interaction behaviors, we will introduce the communication mechanism of MQTT-SN using the example of a publisher publishing data with topics. The whole process of a publisher publishing data with topics mainly consists of four stages, as illustrated in Fig.3,

including searching for a gateway, establishing a connection, registering a topic name and publishing a message.



**Fig. 3.** MQTT-SN Communication Mechanism

In the first step, the publisher needs to find a gateway to assist in forwarding its requests and messages. This involves the following steps:

1. The publisher broadcasts an encrypted control packet to all other devices in the network to search for a gateway.
2. When a gateway receives the packet, it verifies the legitimacy of the publisher. If the publisher is legitimate, it replies to the publisher with a control packet containing its own information to inform the client of its address.
3. After receiving the packet containing the gateway's information, the client also needs to confirm the legitimacy of the gateway's identity. If the gateway is legitimate, the publisher stops searching for a gateway.

MQTT-SN is based on the publish-subscribe pattern and requires a broker to coordinate data between publishers and subscribers. In the second step, the publisher needs to establish a connection with the broker through the gateway to publish messages. The steps to establish a connection are as follows:

1. The client sends an encrypted packet requesting to set up a connection.
2. When the gateway receives the request, it verifies the legitimacy of the client. If the client is legitimate, it forwards the request to the broker.
3. When the broker receives the request, it also needs to verify the legitimacy of the gateway's identity. If the gateway is legitimate, it replies with a response message indicating that the broker agrees to establish a connection.
4. After the client successfully receives the response message agreeing to establish a connection, the connection is established successfully.

In the third step, the publisher needs to initiate a registration process with the gateway to obtain the *TopicId* corresponding to the topic name. When the publisher publishes a message, it needs to use a fixed-length 2-byte topic id (represented by **TopicId**) field to tell the broker which topic the message wants to be published to. Using a shorter *TopicId* to represent the topic instead of the topic name aims to reduce the length of the message, which is one of the adjustments made by MQTT-SN. Here are the steps to follow:

1. The publisher sends an encrypted registration request packet to the gateway.
2. When the gateway receives the request, it first checks the legitimacy of the publisher's identity. If legitimate, it assigns a unique *TopicId* and includes this *TopicId* in the response message to inform the client. The gateway ensures that different topic names have different *TopicIds*.
3. If registration fails due to network or other unexpected reasons, the publisher can initiate registration again.

The fourth step is that the publisher can send encrypted data to the gateway using the *TopicId* successfully registered in the third step. The message is then successfully published to the broker via the gateway. After the broker receives the message, it replies with a confirmation packet. When the publisher receives this confirmation packet, one successful publishing is complete.

### 2.3 CSP

Process algebra is a formal method that characterizes the communication between processes in concurrent systems. Communicating Sequential Process (CSP), proposed by C.A.R. Hoare [11], is a type of process algebra, which has been successfully applied to verify many parallel systems [14] and communication protocols [15, 16]. Therefore, this paper uses CSP as the method to analyze and verify the security of the MQTT-SN protocol communication mechanism.

The syntax and definitions of CSP statements are briefly introduced below, where  $P$  and  $Q$  represent the processes,  $a$  means the atomic actions (also called events),  $b$  stands for a boolean expression and  $c$  denotes the name of channel:

$$P, Q ::= SKIP \mid a \rightarrow P \mid c? v \rightarrow P \mid c! x \rightarrow P \mid \\ P \triangleleft b \triangleright Q \mid P \square Q \mid P \parallel Q \mid P ; Q$$

- $SKIP$  represents that the process terminates successfully.
- $a \rightarrow P$  represents that the process performs the atomic action  $a$  first and then executes the process  $P$ .
- $c?v \rightarrow P$  represents that the process first receives a value  $a$  through the channel  $c$ , then assigns the value to the variable  $v$ , and finally continues to execute the process  $P$ .
- $c!x \rightarrow P$  represents that the process first sends a value  $x$  through the channel  $c$  to another process, and then continues to execute the process  $P$ .
- $P \triangleleft b \triangleright Q$  represents that if  $b$  is true, process  $P$  is executed. Otherwise, process  $Q$  is executed.

- $P \square Q$  represents that it is uncertain whether process  $P$  or process  $Q$  is executed and the choice is made by the external environment.
- $P \parallel Q$  represents that processes  $P$  and  $Q$  are executed concurrently.
- $P; Q$  represents that processes  $P$  and  $Q$  are executed in sequence.
- $P[[a \leftarrow b]]$  represents that the atomic event  $a$  in process  $P$  is replaced by another atomic event  $b$ .
- $P[[c]]Q$  represents that processes  $P$  and  $Q$  are executed in parallel through the channels defined in set  $c$ .

### 3 Modeling MQTT-SN Communication

In this section, we formalize the MQTT-SN communication model based on the mechanism presented in the previous section 2.2.

#### 3.1 Sets, Messages and Channels

In order to formalize the behaviors of different entities in MQTT-SN, we first need to define the sets, messages, and channels used in our model.

First, we give the definition of the sets. **Entity** set contains all entities during message transmission, including the publishers, subscribers, brokers and gateways. **Req** set denotes all request messages during the communication process, such as topic registration requests, connection setup requests, etc. **Prk** set represents the set of private keys involved in communication transmission for implementing encryption and authentication algorithms. **Data** set is composed of plaintext data during communication. **Content** set contains all the other messages, which includes the **Ack** set for feedback messages, the **Tag** set for identity authentication and the **Identifier** set for various identifiers.

Next, we define the following messages based on the sets described above:

$$\begin{aligned}
 MSG &= MSG_{req} \cup MSG_{data} \cup MSG_{ack} \\
 MSG_{req} &= \{msg_{req}a.b.E(k, t, req) \mid a, b \in Entity, k \in Prk, \\
 &\quad t \in Tag, req \in Req\} \\
 MSG_{data} &= \{msg_{data}a.b.req.E(k, t, d) \mid a, b \in Entity, k \in Prk, \\
 &\quad t \in Tag, req \in Req, d \in Data\} \\
 MSG_{ack} &= \{msg_{ack}a.b.E(k, t, ack), msg_{ack1}a.b.E(k, t, ack).id \mid \\
 &\quad a, b \in Entity, ack \in Ack, id \in Identifier\}
 \end{aligned}$$

$MSG_{req}$  represents the set of request messages used in the interaction process for clients to find gateways and establish connections.  $MSG_{data}$  represents the set of messages involving topic registration, data publishing, topic subscribing, and data updating.  $MSG_{ack}$  represents all the confirmations and response messages.  $MSG$  includes all the messages above.

Take one message  $msg_{ack1}a.b.E(k, t, ack).id$  as an example. It means that entity  $a$  sends an acknowledgment message to entity  $b$  with its identifier  $id$ .

$E(k,t,ack)$  indicates that  $ack$  is encrypted with the shared private key  $k$  by applying ChaCha20-Poly1305 algorithm and  $t$  is the generated tag value for identity authentication.

Finally, we define two sets of channels to simulate communications between entities. The set of channels used when there is no intruder is described as  $COM\_PATH$  and it contains  $ComPP$ ,  $ComBP$ ,  $ComSS$  and  $ComBS$ . The set of channels denoted as  $INTRUDER\_PATH$  is used when an intruder is present and it contains  $FakeA$ ,  $FakeB$ ,  $FakeC$ ,  $FakeD$  and  $FakeE$ .

### 3.2 Overall Modeling

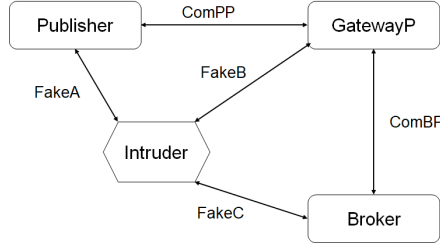


Fig. 4. Publisher Model

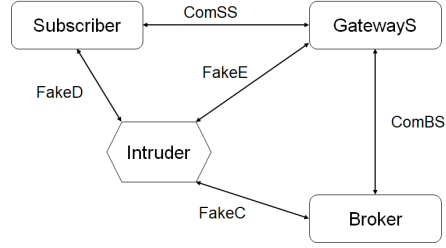


Fig. 5. Subscriber Model

Based on the communication model diagram for the publisher in Fig.4 and for the subscriber in Fig.5, we define two models  $SystemP$  and  $SystemS$ . We divide gateways into two categories:  $GatewayP$  to interact with the publishers and  $GatewayS$  to interact with the subscribers.

$$\begin{aligned}
 SystemP &= Pub \llbracket COM\_PATH \rrbracket GatewayP \llbracket COM\_PATH \rrbracket Broker \\
 SystemS &= Sub \llbracket COM\_PATH \rrbracket GatewayS \llbracket COM\_PATH \rrbracket Broker \\
 System0 &= SystemP \llbracket COM\_PATH \rrbracket SystemS \\
 SYSTEM &= System0 \llbracket INTRUDER\_PATH \rrbracket Intruder
 \end{aligned}$$

Then we formalize the MQTT-SN communication system without intruders  $System0$  by combining  $SystemP$  and  $SystemS$ .  $SYSTEM$  stands for the complete system with the presence of intruders. In this paper, we assume that the intruders are able to eavesdrop on messages sent through normal channels, and are also capable of forging messages and sending them to other entities.

### 3.3 Publisher Modeling

It is the responsibility of the publishers to utilize gateways to transmit messages to the broker so that the broker can forward messages to other clients (i.e. subscribers). As we mentioned before in section 2.2, the process of a publisher



publishing data can be broken down into four main steps: searching for a gateway, establishing a connection, registering a topic id for each topic name and publishing messages. Based on this, we give the following definition:

$$Pub_0 = FindGWP; Connect; TopicReg; MsgPub; Pub_0$$

We define a recursive process  $Pub_0$  that executes  $FindGWP$ ,  $Connect$ ,  $TopicReg$  and  $MsgPub$  in sequence, and finally executes itself. This definition indicates that  $Pub_0$  can repeatedly execute itself so  $FindGWP$ ,  $Connect$ ,  $TopicReg$  and  $MsgPub$  will be continuously executed, which is in line with real-world scenarios where publishers can disconnect and reconnect to resume message publishing.

We define two special events to denote exception handling. We use **fail** to indicate that the request was not processed successfully due to various reasons while **drop** is adopted to represent that the response was sent by some invalid entities and should be discarded.

$$\begin{aligned}
 FindGWP &= ComPP!msg_{req}A.B.E(k, t, search) \rightarrow \\
 &ComPP?msg_{ack1}B.A.E(k, t, ack).gwId \rightarrow \\
 &\left( \begin{array}{l}
 (\mathbf{SKIP} \triangleleft (ack == true) \triangleright (\mathbf{fail} \rightarrow FindGWP)) \\
 \triangleleft (DV(k, t', E(k, t, ack))) \triangleright \\
 (\mathbf{drop} \rightarrow FindGWP)
 \end{array} \right) \\
 Connect &= ComPP!msg_{req}A.B.E(k, t, connect) \rightarrow \dots \\
 TopicReg &= ComPP!msg_{data}A.B.reg.E(k, t, name) \rightarrow \dots \\
 MsgPub &= ComPP!msg_{data}A.B.pub.E(k, t, data) \rightarrow \\
 &ComPP?msg_{ack}B.A.E(k, t, ack) \rightarrow \\
 &\left( \begin{array}{l}
 (\mathbf{MsgPub} \triangleleft (ack == true) \triangleright (fail \rightarrow MsgPub)) \\
 \triangleleft (DV(k, t', E(k, t, ack))) \triangleright \\
 (drop \rightarrow MsgPub)
 \end{array} \right) \\
 &\square ComPP!msg_{req}A.B.E(k, t, disconnect) \rightarrow \\
 &ComPP?msg_{ack}B.A.E(k, t, ack) \rightarrow \dots
 \end{aligned}$$

For instance, in the subprocess  $FindGWP$ , the publisher (**represented by A**) first sends an encrypted  $search$  request and then waits for a response from the gateway (**represented by B**). After that, it verifies whether the response is sent by a legal gateway entity using function **DV** (means **D**ecryption and **V**erification). The function **DV** will determine the legitimacy of the entity by comparing the tag values (i.e.  $t$  and  $t'$ ). If the response is sent by an illegal entity, the publisher will drop the response (denoted as **drop**), resend the search request, and repeat the above steps. Otherwise, the publisher will judge whether the gateway allows a connection according to the value of the  $ack$ . If  $ack$  is true, it means that the gateway accepts communication from the publisher. The publisher can then stop searching for a gateway, which is denoted as **SKIP**, so the process  $Pub_0$  can execute the next subprocess  $Connect$  to establish a

connection with the broker. However, the *search* request failed (denoted as **fail**) when *ack* is false.

Subprocesses *Connect*, *TopicReg* and *MsgPub* are similar to *FindGWP*, so we will only provide partial definitions for them. Especially, *MsgPub* will call itself instead of executing *SKIP* to continue the recursion when a *pub* request is successfully ended. This means that multiple messages can be published in a single connection until a *disconnect* request is made. The general choice symbol  $\square$  splits the handling of different requests, which are frequently used later.

The  $Pub_0$  model above does not consider the presence of intruders. Based on the  $Pub_0$  model, we define *Pub* model that takes intruders into account by the following renaming operations:

$$\begin{aligned} Pub = Pub_0 [ & [ComPP! \{|ComPP|\} \leftarrow ComPP! \{|ComPP|\}, \\ & ComPP! \{|ComPP|\} \leftarrow FakeB! \{|ComPP|\}, \\ & ComPP? \{|ComPP|\} \leftarrow ComPP? \{|ComPP|\}, \\ & ComPP? \{|ComPP|\} \leftarrow FakeA? \{|ComPP|\}] ] \end{aligned}$$

Here,  $|c|$  represents the set of messages that can be transmitted on channel  $c$ .

The first two lines of the above *Pub* definition indicate that when  $Pub_0$  sends messages on channel *ComPP*, *Pub* can also send the same messages on channel *FakeB*. This is to simulate the behavior of intruders faking messages. The last two lines indicate that when  $Pub_0$  receives messages on channel *ComPP*, *Pub* can also receive the same messages on channel *FakeA*. This is to achieve the behavior of intruders eavesdropping on messages.

### 3.4 Gateway Modeling

The main function of gateways is to respond to search gateway requests and topic registration requests from clients. Gateways also need to forward other requests between clients and the broker. We divide and formalize the behaviors of gateways into two processes: *GatewayP* which interacts with publishers and *GatewayS* which interacts with subscribers.

The detail of modeling *GatewayP* is presented below. We omit the detail of modeling *GatewayS* due to their similarity. We set a variable **TpcTable** for *GatewayS* and *GatewayP* respectively, which records the corresponding relationship between each topic name and its *topicId*.

$$\begin{aligned} GatewayP_0(\mathbf{TpcTable}) = & ComPP?msg_{req}A.B.E(k, t, search) \rightarrow \\ & \left( ComPP!msg_{ack1}B.A.E(k, t, true).gwId \right) \rightarrow \\ & \left( \triangleleft (DV(k, t', E(k, t, search))) \triangleright \right) \rightarrow \\ & \left( ComPP!msg_{ack1}B.A.E(k, t, false).none \right) \rightarrow \\ & GatewayP_0(TpcTable) \\ \square ComPP?msg_{data}A.B.reg.E(k, t, name) \rightarrow & \end{aligned}$$

$$\left( \left( \left( \begin{array}{l} \text{SKIP} \\ \triangleleft (\exists \text{entry} \in \text{TpcTable} \cdot \text{entry.key} == \text{name}) \triangleright \end{array} \right); \right. \right. \\
 \left. \left. \begin{array}{l} \text{AddEntry} \\ \text{ComPP!msg}_{ack1} B.A.E(k, t, \text{true}). \mathbf{TpcTable}[\text{name}] \\ \rightarrow \text{GatewayP}_0(\text{TpcTable}) \end{array} \right) \right) \\
 \triangleleft (DV(k, t', E(k, t, \text{name}))) \triangleright \\
 \left( \begin{array}{l} \text{ComPP!msg}_{ack1} B.A.E(k, t, \text{false}). \text{none} \\ \rightarrow \text{GatewayP}_0(\text{TpcTable}) \end{array} \right) \\
 \square \text{ComPP?msg}_{req} A.B.E(k, t, \text{connect}) \rightarrow \dots \\
 \square \text{ComPP?msg}_{data} A.B.\text{pub}.E(k, t, \text{data}) \rightarrow \dots \\
 \square \text{ComPP?msg}_{req} A.B.E(k, t, \text{disconnect}) \rightarrow \dots$$

In the first part, we describe how to handle gateway searching requests. Upon receiving a *search* request from a client, the gateway first uses the *DV* function to verify the legitimacy of the client. When the client is legal, the gateway replies with a message containing a value of *true* for *ack* and its own information represented by **gwId**, indicating its agreement to help the client forward messages. The reply messages also need to be encrypted.

In the second part, we describe how to handle topic registration requests. After receiving a registration request *reg* from the publisher, the gateway still needs to verify the legitimacy of the publisher's identity. Only when the publisher is a legal entity can the following steps be taken. The gateway searches in *TpcTable* for the topic *name*. If the *name* exists, it means that the *name* has been registered before and there is no need to allocate a new id. Otherwise, a unique id is assigned to the *name* and this record is added to *TpcTable*, which is marked as a function named **AddEntry**. Finally, we retrieve the corresponding id for *name* using **TpcTable[name]** and send it to the related publisher.

Since the definitions for the other requests are similar to the first part, their detailed modeling is omitted here.

As with the process *Pub*, we can define the process *GatewayP* that takes into account the existence of intruders by renaming operations based on the current process *GatewayP*<sub>0</sub>.

### 3.5 Broker Modeling

The broker is mainly responsible for two functions: first, to respond to various requests from the gateway, and second, to coordinate messages from different topics. If the data related to a certain topic subscribed by a subscriber changes, the broker should push the updated data to the subscriber.

We formalize the model of the broker as below:

$$\begin{aligned}
 \text{Broker}_0 = & \text{ComBS!msg}_{data} C.D.\text{update}.E(k, t, \text{data}) \rightarrow \\
 & \text{ComBS?msg}_{ack} D.C.E(k, t, \text{ack}) \rightarrow
 \end{aligned}$$

$$\begin{aligned}
& \left( \begin{array}{l} (Broker_0 \triangleleft (ack == true) \triangleright (fail \rightarrow Broker_0)) \\ \triangleleft (DV(k, t', E(k, t, ack))) \triangleright \\ (drop \rightarrow Broker_0) \end{array} \right) \\
& \square ComBP?msg_{req}B.C.E(k, t, connect) \rightarrow \\
& \left( \begin{array}{l} ComBP!msg_{ack}C.B.E(k, t, true) \\ \triangleleft (DV(k, t', E(k, t, connect))) \triangleright \\ ComBP!msg_{ack}C.B.E(k, t, false) \end{array} \right) \rightarrow Broker_0 \\
& \square ComBP?msg_{data}B.C.pub.E(k, t, data) \rightarrow \dots \\
& \square ComBS?msg_{data}D.C.sub.E(k, t, topic) \rightarrow \dots \\
& \square \dots
\end{aligned}$$

In the first part, the broker (**represented by C**) sends an *update* request to the gateway (**represented by D**) interacting with the subscriber, in order to notify the subscriber that the data of topics he has subscribed to have changed. Then the broker waits for the response and uses the *DV* function to test the legitimacy of the gateway. After authentication, if the value of *ack* is true, the request is processed successfully, and vice versa. Failure to pass authentication means that the response is from an illegal entity. For simplicity, we allow the broker to send *update* requests at any time.

In the second part, the broker receives a connection setup request from the gateway. After verifying that the gateway is legitimate, the broker gives an answer with an *ack* value of *true*, indicating that the connection is allowed to be set up. The rest parts are similar to the second part, so we omit the details here.

As with the process *Pub*, we can define the process *Broker* under the existence of intruders by renaming *Broker0*.

### 3.6 Intruder Modeling

In this paper, we assume that intruders can intercept or fake messages via normal communication channels *ComPP*, *ComBP*, *ComSS* and *ComBS*.

First, we define a set **Fact** as below, which includes all the facts the intruder can learn at its initial state. The intruder can know all the entities in the system, the intruder's own private key  $prk_i$  and its own tag value  $tag_i$ , as well as all the encrypted messages *MSG* during communication.

$$\mathbf{Fact} = Entity \cup \{prk_i, tag_i\} \cup MSG$$

In addition, the intruder can deduce new facts based on the set of facts that it has already learned. And the specific deduction rules are as follows:

$$\begin{aligned}
& \{k, d\} \rightarrow E(k, d) \\
& \{sk, E(sk, d)\} \rightarrow d \\
& (F \rightarrow f) \wedge (F \subseteq F') \rightarrow (F' \implies f)
\end{aligned}$$

The first rule states that the intruder can get the encrypted message  $E(k,d)$  if it has the encryption key  $k$  and the data  $d$ . The second rule states that the intruder can get the plaintext  $d$  if it has the decryption key  $sk$  and the encrypted message  $E(sk,d)$ . The third rule states that if the fact  $f$  can be deduced from the fact set  $F$ , and  $F$  is a subset of  $F'$ , then the intruder can also deduce  $f$  from the bigger set  $F'$ .

Then, we define the function  $Info(msg)$  to describe the facts that the intruder can deduce from the different types of messages that are defined previously.

$$\begin{aligned} Info(msg_{req}a.b.E(k,t,req)) &= \{a,b,E(k,t,req)\} \\ Info(msg_{data}a.b.req.E(k,t,d)) &= \{a,b,req,E(k,t,d)\} \\ Info(msg_{ack}a.b.E(k,t,ack)) &= \{a,b,E(k,t,ack)\} \\ Info(msg_{ack1}a.b.E(k,t,ack).id) &= \{a,b,E(k,t,ack),id\} \end{aligned}$$

The first rule indicates that the intruder can deduce from this kind of message that it was sent from entity  $a$  to entity  $b$ . Also, the message content is an encrypted packet  $E(k,t,req)$ . The remaining rules are similar to the first one.

Therefore, we introduce a channel called  $Deduce$  for the intruder process to deduce new facts through this channel, which is defined as follows:

$$Channel \text{ Deduce} : Fact.P(Fact)$$

Based on all the deduction rules and the channel definition above, the model of the intruder can be defined as follows:

$$\begin{aligned} Intruder_0(F) &= m \in MSG \text{ Fake?} m \rightarrow Intruder_0(F \cup Info(m)) \\ &\square \square_{m \in MSG \cap Info(m) \subseteq F} \text{ Fake!} m \rightarrow Intruder_0(F) \\ &\square \square_{f \in Fact, f \notin F, F \rightarrow f} \text{ Init} \{Data\_Leakage\_Success = false\} \\ &\rightarrow Deduce.f.F \\ &\rightarrow \left( \begin{array}{l} \left( \begin{array}{l} Data\_Leakage\_Success = true \\ \rightarrow Intruder_0(F \cup \{f\}) \end{array} \right) \\ \triangleleft (f == data) \triangleright \\ Intruder_0(F) \end{array} \right) \end{aligned}$$

In the above definition,  $Fake$  represents the integrated set of all channels contained in  $INTRUDER\_PATH$ , and  $F$  is a set that contains the intruder's current known messages. The first line states that the intruder can eavesdrop on messages through all the channels in the  $Fake$  set and add the deduced content to its known fact set  $F$ . The second line states that the intruder can fake messages based on known facts and send them to other entities. The remaining lines state that the intruder can deduce new facts based on known messages through the  $Deduce$  channel, and then add these deduced new facts to its known fact set  $F$ . If the intruder can deduce the plaintext of a message (defined as  $f==data$ ), it indicates a data leakage scenario.

Finally, we give the complete definition of the intruder process as below, where  $IF$  represents the set of facts the intruder can get initially:

$$\begin{aligned} \text{Intruder} &= \text{Intruder}_0(IF) \\ IF &= \{A, B, C, D, E, \text{prk}_i, \text{tag}_i\} \end{aligned}$$

## 4 Implementation and Verification

In this section, we use the model checking tool PAT to implement the model we constructed in section 3 and then verify seven properties of the model. The verification results are shown at the end of this section.

### 4.1 Implementation

First, we give a brief introduction of the syntax and definitions of PAT as below:

- **#define goal value == 1**; It defines a proposition named *goal* that evaluates to true only when the variable named *value* is equal to 1.
- **var a = 1**; It defines a global variable named *a* and assigns it the value 1.
- **enum{a, b}**; It defines two enumeration constants named *a* and *b*.
- **channel c 0**; It defines a channel named *c* with a buffer size of 0, indicating that it is for synchronous communication. The buffer size of the channel must be greater than or equal to 0.
- **#assert P() deadlockfree**; It defines an assertion to check whether the process *P* will go into a deadlock state with a built-in primitive in PAT.
- **#assert P() reaches goal**; It defines an assertion to check whether the process *P* will go into a state, where the property named *goal* is satisfied.
- **#assert P() | = [] ! F**; It defines an assertion to check whether the process *P* can never reach a state where the property *F* holds.

### 4.2 Properties Verification

#### Property 1: Deadlock Freedom

The deadlock state refers to the situation where the system is continuously blocked and unable to perform any actions. We can use the verification primitive provided by PAT to check this property. The verification primitive is as follows:

```
# assert SYSTEM deadlockfree;
```

#### Property 2: Divergence Freedom

Divergence refers to the system being trapped in an infinite loop and continuously consuming resources secretly. We also use the primitive provided by PAT to check this property. The verification primitive is as follows:

```
# assert SYSTEM divergencefree;
```

**Property 3: Data Reachability**

*Data Reachability* refers to the ability that all the messages published by the clients and all the requests sent by the clients can be successfully received and processed by the broker server. We define a state with a variable called *Data\_Reachability\_Success* to indicate that this property is satisfied and then use assert to check whether the model can reach this state.

```
#define Data_Reachability_Success data_reachability == true;
#assert SYSTEM reaches Data_Reachability_Success;
```

**Property 4: Client Security**

*Client Security* refers to the situation where intruders cannot impersonate publishers or subscribers to communicate with other entities in the system. We define a state called *Client\_Fake\_Success* to indicate that the system is in a state where intruders can successfully impersonate clients. Then we use an assert statement with the always symbol  $\square$  defined in PAT to check whether the model can ever reach this state.

```
#define client_fake_success (pub_fake_success || sub_fake_success);
#define Client_Fake_Success client_fake_success == true;
#assert SYSTEM | =  $\square$ ! Client_Fake_Success;
```

**Property 5: Gateway Security**

*Gateway Security* refers to the state in which intruders are unable to pretend to be gateways. Similarly, we define a state called *Gateway\_Fake\_Success* and then check whether the system will never enter into this state.

```
#define gateway_fake_success (gwp_fake_success || gws_fake_success);
#define Gateway_Fake_Success gateway_fake_success == true;
#assert SYSTEM | =  $\square$ ! Gateway_Fake_Success;
```

**Property 6: Broker Security**

*Broker Security* stands for the situation where intruders cannot impersonate brokers to communicate in the system. Similarly, we define a state called *Broker\_Fake\_Success* and check by using an assert statement.

```
#define Broker_Fake_Success broker_fake_success == true;
#assert SYSTEM | =  $\square$ ! Broker_Fake_Success;
```

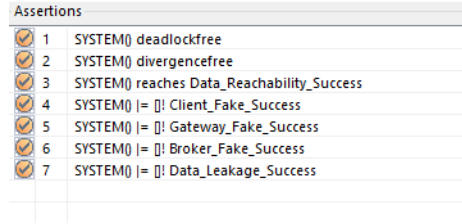
**Property 7: Data Leakage**

Data leakage refers to the situation where intruders can obtain, use or share plaintext data during the communication process, which is not allowed in a safe system. Protecting data privacy and confidentiality is an important issue for WSN. We define a state called *Data\_Leakage\_Success* to indicate the state where intruders can access the plaintext data.

```
#define Data_Leakage_Success data_leakage_success == true;
#assert SYSTEM | =  $\square$ ! Data_Leakage_Success;
```

### 4.3 Verification Results

According to our definitions and assertions of different properties, we verify our model in PAT. The verification results are shown in Fig.6. We can see that the seven properties are all valid.



Assertions		
1	SYSTEM()	deadlockfree
2	SYSTEM()	divergencefree
3	SYSTEM()	reaches Data_Reachability_Success
4	SYSTEM() != []	Client_Fake_Success
5	SYSTEM() != []	Gateway_Fake_Success
6	SYSTEM() != []	Broker_Fake_Success
7	SYSTEM() != []	Data_Leakage_Success

**Fig. 6.** Verification Results in PAT

This means that our system will never run into a deadlock or divergence state and all the clients can get the data they want. In addition, it indicates that the intruder cannot pretend to be a normal entity during communication and there is no risk of leaking data in our system.

## 5 Conclusion and Future Work

This paper analyzes and formalizes the main components of MQTT-SN communication. The MQTT-SN protocol does not specify security measures to maintain lightweight, so this paper introduces the ChaCha20-Poly1305 algorithm as a security guarantee. Then, we list seven properties that need to be verified including deadlock freedom, divergence freedom, data reachability, client security, gateway security, broker security, and data leakage. Moreover, we use the model checker PAT to verify the above properties. According to the verification results, we can summarize that all these properties are satisfied in our model.

When modeling the MQTT-SN protocol communication system, this paper considers the possible attack behaviors that may be encountered in the real environment, such as eavesdropping and forgery. However, in practical applications, there are more types of attacks that may weaken the security of the MQTT-SN communication, such as DDoS attacks and sinkhole attacks [4, 17]. In the future, more attack behaviors can be introduced to enrich the intruder process.

**Acknowledgements.** This work was partially supported by the National Key Research and Development Program of China (No. 2022YFB3305102), the National Natural Science Foundation of China (Grant No. 62032024), the “Digital Silk Road” Shanghai International Joint Lab of Trustworthy Intelligent Software (No. 22510750100), and Shanghai Trusted Industry Internet Software Collaborative Innovation Center.



## References

1. Kandris, D., Nakas, C., Vomvas, D., Koulouras, G.: Applications of Wireless Sensor Networks: An Up-to-Date Survey. *Applied System Innovation* **3**(1) (2020)
2. Sharma, S., Kaur, A.: Survey on Wireless Sensor Network, Its Applications and Issues. *Journal of Physics: Conference series* **1969**(1), 12042 (2021)
3. Stanford-Clark, A., Truong, H.L.: MQTT for Sensor Networks (MQTT-SN) Protocol Specification. International business machines (IBM) Corporation version 1(2), 1–28 (2013)
4. Avila, K., Sanmartin, P., Jabba, D., Gómez, J.: An analytical Survey of Attack Scenario Parameters on the Techniques of Attack Mitigation in WSN. *Wireless Personal Communications* **122**, 3687–3718 (2022)
5. Park, C.S., Nam, H.M.: Security Architecture and Protocols for Secure MQTT-SN. *IEEE Access* **8**, 226422–226436 (2020)
6. Roldán-Gómez, J., Carrillo-Mondéjar, J., Castelo Gómez, J.M., Ruiz-Villafranca, S.: Security Analysis of the MQTT-SN Protocol for the Internet of Things. *Applied Sciences* **12**(21), 10991 (2022)
7. Diwan, M., D’Souza, M.: A Framework for Modeling and Verifying IoT Communication Protocols. In: International Symposium on Dependable Software Engineering: Theories, Tools, and Applications. pp. 266–280. Springer (2017)
8. Sadio, O., Ngom, I., Lishou, C.: Lightweight Security Scheme for MQTT/MQTT-SN Protocol. In: 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS). pp. 119–123. IEEE (2019)
9. Kao, T., Wang, H., Li, J.: Safe MQTT-SN: A Lightweight Secure Encrypted Communication in IoT. In: *Journal of Physics: Conference Series*. p. 012044. IOP Publishing (2021)
10. De Santis, F., Schauer, A., Sigl, G.: ChaCha20-Poly1305 Authenticated Encryption for High-speed Embedded IoT Applications. In: Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 692–697. IEEE (2017)
11. Hoare, C.A.R.: *Communicating Sequential Processes*. Prentice Hall International (1985)
12. National University of Singapore: PAT: Process Analysis Toolkit (2007), <https://pat.comp.nus.edu.sg/>
13. Sun, J., Liu, Y., Dong, J.S.: Model Checking CSP Revisited: Introducing a Process Analysis Toolkit. In: International symposium on leveraging applications of formal methods, verification and validation. pp. 307–322. Springer (2008)
14. Xu, J., Yin, J., Zhu, H., Xiao, L.: Modeling and Verifying Producer-consumer Communication in Kafka Using CSP. In: 7th Conference on the Engineering of Computer Based Systems. pp. 1–10. ACM (2021)
15. Lowe, G., Roscoe, B.: Using CSP to Detect Errors in the TMN Protocol. *IEEE Transactions on Software Engineering* **23**(10), 659–669 (1997)
16. Chen, S., Li, R., Zhu, H.: Formalization and Verification of Group Communication CoAP Using CSP. In: International Conference on Parallel and Distributed Computing: Applications and Technologies. pp. 616–628. Springer (2021)
17. Abidoye, A.P., Obagbuwa, I.C.: DDoS Attacks in WSNs: Detection and Countermeasures. *IET Wireless Sensor Systems* **8**(2), 52–59 (2018)