TERAKREDITASI RISTEKDIKTI No. 36b/E/KPT/2016

Jurnal Rekayasa Elektrika

VOLUME 15 NOMOR 2 Agustus 2019

JRE	Vol. 15	No. 2	Hal 75–156	Banda Aceh, Agustus 2019	ISSN. 1412-4785 e-ISSN. 2252-620X

Jurnal Rekayasa Elektrika Vol. 15, No. 2, Agustus 2019, hal. 125-133
ISSN. 1412-4785; e-ISSN. 2252-620X, Terakreditasi RISTEKDIKTI No. 36b/E/KPT/2016
DOI: 10.17529/jre.v15i2.13963

Implementation of Event-Based Dynamic Authentication on MQTT Protocol

Rizka Reza Pahlevi, Parman Sukarno, and Bayu Erfianto
School of Computing, Telkom University
Bandung, Indonesia, 40257
e-mail: rizkarezapahlevi@student.telkomuniversity.ac.id

***Abstrak*—Artikel ini mengusulkan mekanisme otentikasi pada protokol MQ *Telemetry Transport* (MQTT). Pertukaran data dalam sistem IoT menjadi aktivitas penting. Pertukaran data ini dilakukan dengan komunikasi antar perangkat *Internet of Things* (IoT). Protokol MQTT adalah protokol komunikasi yang cepat dan ringan untuk IoT. Protokol MQTT menggunakan broker sebagai peladen untuk *publish/subscribe*. Salah satu masalah pada protokol MQTT adalah tidak adanya mekanisme keamanan pada pengaturan awal. Tahap registrasi client memiliki kerentanan terhadap serangan *client* palsu karena tidak adanya mekanisme otentikasi. Mekanisme otentikasi telah dibuat sebelumnya menggunakan *Transport Layer Security* (TLS). Namun, mekanisme TLS mengkonsumsi lebih dari 100 KB memori data dan tidak bersahabat untuk perangkat yang memiliki batasan. Oleh karena permasalahan tersebut diperlukan mekanisme otentikasi yang cocok untuk perangkat terbatas. Artikel ini mengusulkan protokol untuk mekanisme otentikasi berbasis kejadian dan dinamis untuk protokol MQTT. Penggunaan berbasis kejadian diajukan untuk mengurangi beban komputasi perangkat terbatas. Penggunaan dinamis ditujukan untuk memberikan properti otentikasi yang berbeda pada setiap sesi sehingga dapat meningkatkan keamanan otentikasi. Dari hasil evaluasi, protokol otentikasi dinamis berbasis kejadian berhasil diterapkan kepada perangkat terbatas mikrokontroler dan broker. Mikrokontroler sebagai *client* mampu melakukan proses untuk protokol yang diajukan. *Broker* mampu memilah *client* yang otentik dan perangkat terbatas mampu melakukan komputasi untuk melakukan proses mutual otentikasi kepada *client*. Mikrokontroler menggunakan 52% memori untuk protokol yang diajukan dan hanya mengkonsumsi 2% lebih tinggi dari protokol tanpa keamanan. *Broker* mampu memilah *client* yang otentik dan perangkat terbatas mampu melakukan komputasi untuk melakukan proses mutual otentikasi kepada *client*. *Broker* menggunakan real memori maksimum sebesar 4,3 MB dan penggunaan CPU maksimum 3,7%..**

Kata kunci: ***IoT, otentikasi, dinamis, berbasis kejadiaan, MQTT***

***Abstract*—This paper proposes an authentication mechanism on the MQ Telemetry Transport (MQTT) protocol. The exchange of data in the IoT system became an important activity. The MQTT protocol is a fast and lightweight communication protocol for IoT. One of the problems with the MQTT protocol is that there is no security mechanism in the initial setup. One security attack may occur during the client registration phase. The client registration phase has a vulnerability to accept false clients due to the absence of an authentication mechanism. An authentication mechanism has been previously made using Transport Layer Security (TLS). However, the TLS mechanism consumes more than 100 KB of data memory and is not suitable for devices that have limitations. Therefore, a suitable authentication mechanism for constraint devices is required. This paper proposes a protocol for authentication mechanisms using dynamic and event-based authentication for the MQTT protocol. The event-based is used to reduce the computing burden of constraint devices. Dynamic usage is intended to provide different authentication properties for each session so that it can improve authentication security. As results, the applied of the event-based dynamic authentication protocol was successful in the constraint devices of microcontrollers and broker. The microcontroller, as a client, is able to process the proposed protocol. The client uses 52% of the memory for the proposed protocol and only consumes 2% higher than the protocol without security. The broker can find authentic clients and constraint devices capable of computing to carry out mutual authentication processes to clients. The broker uses a maximum of 4.3 MB of real memory and a maximum CPU usage of 3.7%.**

Keywords: ***IoT, authentication, dynamic, event-based, MQTT***

I. INTRODUCTION

Since the establishment of the internet, the development of hardware and software had increased. The increasing quality of internet services today creates innovations. One of the innovation is the Internet of Things (IoT) [1]–[3]. The IoT makes physical devices in the environment (such as medical devices, vehicles, sensors) become part of computer networks and operate without human intervention [4], [5]. The architecture of

Received 01 July 2019; Revised 31 July 2019; Accepted 12 August 2019

the IoT consists of four-layer: Sensing layer, Network layer, Service layer, and Applications layer [6]. The Sensing layer generally contains constraint devices as in RFC72278 [7]. The constraint device is used to convert environmental parameters into digital data. The use of constraint devices to the IoT is intended to be one of them for device mobilization and resource requirements.

The data from the sensing layer was sent to another layer using the communication protocol. The network layer provides a communication protocol between nodes. One of the communication protocol for the IoT is Message Queuing Telemetry Transport (MQTT). The MQTT protocol is an open standard protocol that is issued by the Organization for the Advancement of Structured Information Standards (OASIS) that can be used with devices that have limited memory, low data rates and are transmitted over the network [8]–[10]. The MQTT protocol works with the published/subscribed model. Figure 1 shows a scheme of communication between the client and the broker. That figure shows the MQTT protocol scheme which consists of publisher P1, P2, ..., Pn and subscriber S1, S2, ..., Sn. The task of the publisher is to send data to the MQTT broker through networks, and the subscriber is to receive messages from brokers through networks on certain topics. The clients use the topic to publish and subscribe.

However, from the advantages possessed by the MQTT protocol, the MQTT protocol has a threat [8], [9], especially at MQTT brokers. The activities such as registering and publishing are using the MQTT broker. The MQTT protocol does not have a security mechanism in the default setting [11], [8], [9]. Since the absences of the security mechanism in the MQTT protocol causes several problems, one of them is the authentication mechanism [6], [8], [9], [11]. The authentication mechanism is a method to prove the authenticity of the client, whether it is legitimate or not to the broker. Because the broker cannot verify the client, the broker can accept the connection message from any client. This phase has a vulnerability to the broker receiving a fake client. Unauthenticated clients can cause unauthorized access [6] that results in fake messages, malfunctions, or collapses.

Authentication issues in the MQTT protocol have been previously addressed. One mechanism is to use Transport Layer Security (TLS). The TLS is a security standard for communication between two parties. The TLS is providing symmetric, asymmetric, and hash encryption to accommodate integrity, confidentiality, and authentication.

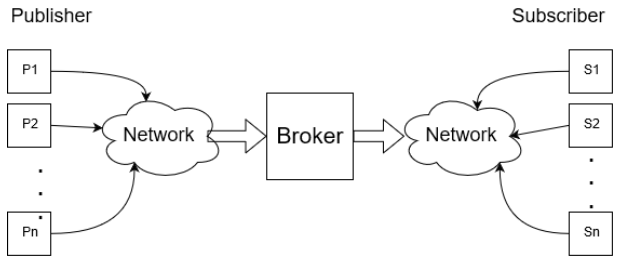

Figure 1. Existing MQTT Scheme [8], [6]

However, Sorcha Nolan [12] states that TLS added costs to the process, so it was not suitable for constraint devices. Gil Reiter [13] states that TLS used more than 100 KB of memory consumption, which is certainly not a problem for smartphone, but it is a problem for constraint devices to RFC7228. Miranda P. et al. [14] states that there was overhead energy for TLS. To deal with the problem of energy use in the TLS, Tae Ho Cho et al. [15] proposed the method. However, Tae Ho Cho et al. proposed method cannot handle the problem of memory usage [12].

The overconnected handling needed to overcome the authentication problems. Overconnected is caused by an end device that cannot prove the authenticity of the client. This research proposes an event-based authentication protocol mechanism. The authentication mechanism proposed in this research consists of two pillars, namely event-based and dynamic property pillars. The use of event-based control systems pillar performed when triggers occur, which potentially reduce computational and communication burdens and do not require additional synchronization to the server [16]. The dynamic pillar makes security attributes on each event change so that it can improve security. In this research succeeded in implementing event-based dynamic authentication on microcontroller devices, and brokers were able to authenticate clients.

The contribution of this research is to improve security in the MQTT protocol by using dynamic event-based authentication. In this research, it introduced a dynamic authentication security mechanism using an event-based control system that had not previously been studied on the IoT. This study proposes a security mechanism that simultaneously considers computing and memory on constraint devices such as microcontrollers. Previous research has tried to solve this problem, but as far as we know, it does not consider computing devices that have limitations either. Besides, previous research has not discussed authentication at the client level on the IoT system. Therefore, this study took part to improve authentication security.

II. Literature Review

The MQTT protocol is a communication protocol that uses the publish/subscribe mechanism to requests/responses [17] [18] [11]. There are four properties at the MQTT protocol: IP, Payload, Topic, and Port [8]. The MQTT protocol, there are three agents: Publisher, Subscriber, and Broker. The Publisher works by connecting the client to the broker and sending messages to the broker. The subscriber works by connecting the client to the broker and receives messages on the desired topic. The broker work to exchange messages, receive messages and forward messages. The agents have relationships and are attached to certain attributes. The MQTT protocol, the topic is an inherent property that must exist.

Budi R. et al. [8] reviewed the security in the MQTT protocol. The Security required in MQTT includes

authentication, authorization, and access control. However, the MQTT protocol does not have it all. The MQTT protocol only provides authentication without encryption. The authentication process on the MQTT generally uses a username and password. The sending of the authentication message is not encrypted. Budi R. et al. [8] provides a demonstration of how the MQTT attacks are carried out on the network by obtaining an MQTT packet from the network. The attacker registers himself to the broker with the content from the obtained header data. Furthermore, the attacker gain access to the resources at the broker. S. N. Firdous et al. [9] provides categories of attacks and effects of attacks. The attack categories describes the denial of service, identity spoofing, information disclosure, elevation of privileges, and tampering data also explains how broker are attacked with unauthorized access.

The Transport Layer Security (TLS) security mechanism has been established to provide security mechanisms. TLS provides a secure communication channel through two hosts. TLS uses symmetric and asymmetric functions and hashes to provide authentication, encryption for data, and data integrity. TLS provides security for authentication, authorization, and encryption of data sent. TLS has become standard security for communication over the network. TLS in the MQTT protocol has been used to secure data and maintain the authenticity of the broker from the message received. However, P. Miranda et al. provides exposure to how TLS has overhead in computing [14]. The energy consumed depends on the amount of data to be sent. Gil Reiter explained that TLS needed at least 100 KB of memory [13]. The computation of TLS security mechanism that can be done by smart-phones and other devices that have much memory, but not for devices that have limitations such as those on RFC7228 [7].

Sorcha Nolan had provided another security mechanism at MQTT [12]. Sorcha Nolan provides an authentication mechanism on the MQTT protocol by providing encryption on the payload. This study aims to provide low computing and memory usage for the authentication process on the MQTT protocol payload. Sorcha Nolan succeeded in providing an authentication mechanism on MQTT content regarding clients that have limitations. However, this study does not pay attention to client authentication mechanism, where it is used to validate the clients that want to uses the broker resources.

The control system, there are two types, event-based and time-based. An event-based control system is a control system that updates its information when a particular event triggers the function. This control system computation carried out if certain events trigger the system. This control scheme is a solution where the device has a limit power usage [16]. An event-based control system performed when state changes occur, or there are triggers from other operations that potentially reduce the computing burden. In the case of IoT, because not all the IoT devices have the same delivery time, event-based control system more efficient. Also, the eventbased control scheme saves bandwidth because it does not require synchronization to the server to update the property.

III. Methodology and System Design

A. Proposed Protocol Concept

The design of the proposed protocol for dynamic event-based authentication divided into two: event id and encryption. Pseudo-random number generator is used to create the event id. The pseudo-random number generator has a concept where the seed becomes a generator of current value. Each client owns a seed (each one), and the broker owns the entire list of client seeds. The event id is formed by clients who want to register and brokers to mutually authenticate. Moreover, the event id formed by the client is called *Evk* and the broker is called *Evb*. The encryption process, the value of the event id is added with random characters, and then processed with symmetric encryption using the Fisher Yates Shuffle. The Fisher-Yates shuffle uses key *R* as an individual scrambler. The Fisher-Yates was chosen because it fulfills unbiased permutation rules. It only requires a portion of time for a number of individuals, and does not require additional space [19] [20]. Moreover, the results of encryption from the client are called *Ec*(*Evk*), and the broker is called *Ec*(*Evb*). The *Ec*(*Evk*) is used by the client to register to the broker, and *Ec*(*Evb*) is used by brokers to provide mutual authentication to the client. Because the event id built with a pseudo-random number generator that is affected by the seed, the seed must be updated to ensure the dynamic event id. The update of the event id on the client done after the client sends a message to the broker. On the broker's side, the broker updates the related event id when the broker declares to accept the relevant client. Moreover, the entire system is described as in Figure 2.

B. Authentication Protocol Design

The proposed protocol, the client who register to the broker passes through the secure channel, is as shown in Figure 3. A security channel is formed to process client registering process to the broker. The client sends a registration message to the broker along with the *Ec*(*Evk*) security attribute through the CONNECT header. The MQTT broker receives CONNECT header from the clients and validates security properties *Ec*(*Evk*) from the clients. If the property *Evk* (from the *Ec*(*Evk*) decryption) from the client is accepted, then the broker gives a reply in the form of the security property *Ec*(*Evb*) to the client. The client receives a reply message *Ec*(*Evb*) from the broker and validates the broker security properties. If the client accepts *Evb* (from the *Ec*(*Evb*) decryption) from the broker, then the client believes that the sender is a legitimate broker. This scheme is used to validate the two-way broker and client authentication.

In the registration process, the client must be able to prove its authenticity, and the broker must also be able to verify its authenticity. To complete the registration

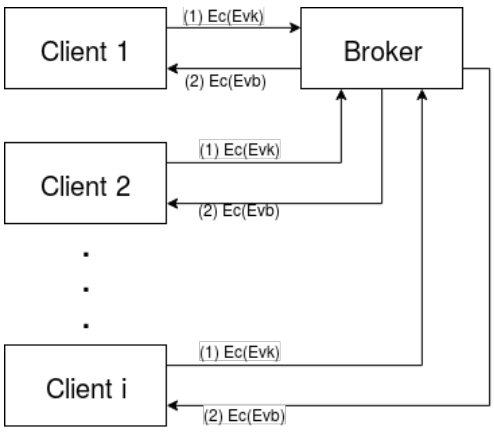

Figure 2. Proposed design protocol for MQTT

process, the client must form the event id *Evk*. The random characters added to the event id that built before. The random character addition is needed to increase the security strength of the Fisher-Yates Shuffle. Adding random characters to the event id is done until it meets the length *m*. After the event id has been added with random characters to the length *m*, it is processed with the Fisher-Yates shuffle so that it becomes *Ec(Evk)*. The client sends this result from *Ec(Evk)* to the broker. The broker who has received *Ec(Evk)* decodes to get *Evk*. If *Evk* is accepted, then the broker forming *Evb*, then random characters are added to have the length *m* and processed with the Fisher-Yates shuffle to form *Ec(Evb),* then the *Ec(Evb)* is sent to the corresponding client. The related client that receives *Ec(Evb)* decode to gets *Evb*. If *Evb* is accepted, then the client has received a message from a legitimate broker.

C. System Design

1. *Client Authentication*: The system design for the proposed protocol consists of two major parts, namely the formation of event id and encryption. The formation of event id based on the rules of the pseudo-random number generator [21]. All seeds used by the client for the pseudo-random number function are registered with the broker. Then the client forms the event id. The event id that has been formed by the client is added by random characters to the length m then encrypted with the Fisher-Yates Shuffle. Then the system design is depicted in Figure 4. The event id formed by the client goes through functions. The functions have properties that are only known by legitimate broker and clients. The product of these functions is a number that is influenced by the initial seed and functions properties. After the event id is done, a random character is added until it has a length of *m*. Both parties have agreed with this length of *m*. This *m* length is needed to improve security when performing the Fisher-Yates shuffle. The Fisher-Yates shuffle has the property that each has a probability *1 / m* to move. With that probability, optimizing *m* can increase the reliability of individual locations. The event id that has been added with this random character is processed Fisher-Yates shuffle with key *R* for the random

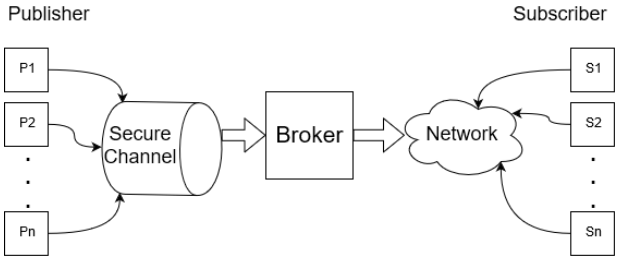

Figure 3. Proposed secure protocol for MQTT

factor. The key *R* consists of an array along *m* where each array has a value with a range of *1 <m-1*. The product of event id and random characters that have been subject to Fisher-Yates are called *Ec(Evk)*. The product *Ec(Evk)* is embedded in the CONNECT header along with the ID in the client registration process. The ID does not store any sensitive information.

2. *Broker Authentication:* After the broker receives a registration request from the client, the broker checks the provisions of the registration form. After the broker receives the CONNECT packet from the client, the broker checks whether there is a registration form in the header. If there is a registration form as specified, then the broker decodes the message. The broker decoding *Ec(Evk)* are pinned on the header using the key *R*. The product decoding *Ec(Evk)* produces event id and random characters. The broker sorts out event id and random characters and leaves only the event id *Evk*. This event id *Evk* is used by the broker to find out the authenticity of the client. If the relevant event id *Evk* is registered to the broker, then the broker processes the mutual authentication. The mutual authentication process carried out by the broker is useful to inform the client that the sender is a legitimate broker. After the broker receives an event id from a valid client, the broker updates the seed of the relevant event id. The broker was forming the event id *Evb* from the new seed. The event id that is added random characters to the length *m* and processed with Fisher-Yates shuffle with key *R*. The result of the process is called *Ec(Evb)*. The *Ec(Evb)* result sent to the related client to the PUBLISH header from the MQTT protocol. The system design is depicted in Figure 4.

3. *Mutual Authentication:* After the client receives a publish from the broker, the client checks the provisions of the authentication form. After the client receives the PUBLISH packet from the broker, the client checks whether there is an authentication form in the header. If there is an authentication form as specified, then the client decodes the message. After the client receives *Ec(Evb)* from the broker, the client decodes *Ec(Evb)* with key *R*. The result of the *Ec(Evb)* decoding process leaves the event id *Evb* and random characters. Then, the client sort the event id and random characters. After the event id *Evb* is found, the client matches the event id that the client-owned. If the event id *Evb* that is sent matches, then the client believes that the sender is a legitimate broker. If

the client states that he has a legitimate broker, then the client updates the previous seed. The event id that uses to register later becomes dynamic by updating the previous seed. The system design is illustrated in Figure 4

D. CPU and Real Memory Usage

The use of CPU and real memory is done to measure how much cost must be paid by the MQTT broker in performing the proposed and the existing protocols and how much additional memory is given to the constraint devices. The CPU and real memory measurements on the MQTT broker are measured by monitoring the process identifier (PID). This activity records the real memory and CPU usage of the PID used by the proposed and the existing protocols. After obtained the CPU and real memory usage data at the MQTT broker, the data is presented in a graph. Data samples taken were for 600 seconds (ten minutes) in each session. Measurement of additional memory on the constraint devices is done by compiling the code on the proposed and the existing protocols. The compiler used to compile the code in the Arduino IDE. The result from the compilation is the bytes size of the code being embedded in the constraint device. The subtraction in code size between the proposed protocol and the existing protocol give results in the amount of additional memory needed in the proposed protocol for the constraint device. The measurement of CPU and real memory usage on the MQTT broker and the addition of memory on the constraint devices provide information on costs to be paid by the proposed protocol.

IV. Result and Discussion

A. Proposed Protocol Algorithm

An algorithm is needed as a builder to build the proposed protocol on client and broker. The algorithm 1 and algorithm 2 described is used on clients and broker. In the client, the end result of this algorithm is *Ec(Evk)*, and the broker is *Ec(Evb)*. These algorithms are divided into two parts. The first part is an algorithm to build a pseudo-random number generator called *Pr*. The formation of the *Pr* refers to the seed. The clients use existing seeds to forming the *Pr* and the broker forming the *Pr* from the client seeds that owned. The second part is the formation of *Ec*. The formation of the *Ec* on the client is called *Ec(Evk)* then the broker is called *Ec(Evb)*.

Algorithm 1: The function used to form *Pr* from the previous seed

Client		*Broker*
Init Seedi		*Init Seedi*
compute *Pri*		compute Pr_i
compute *Evki*		compute Evb_i
rc ← random characters		
Evki ← concat(Evki,rc)		
Ec(Evki) ←{Evki,R}		
	CONNECT(ID,Ec(Evk$_i$)) →	
		Evki ←{Ec(Evki),R}
		validate *Evki* with *Evbi*
compute new $Seed_{i+1}$		compute new *Seedi+1* for related client
compute Pr_{i+1}		compute Pr_{i+1}
compute Evk_{i+1}		compute Evb_{i+1}
		rc ← random characters
		Evbi+1 ← concat(Evb_{i+1},rc)
		Ec(Evbi+1) ←{Evb_{i+1},R}
	PUBLISH(Ec(Evb_{i+1})) ←	
	CONACK ←	
Evbi+1 ←{Ec(Evbi+1),R}		
validate *Evbi+1 with Evki+1*		
compute new *Seedi+2*		compute new $Seed_{i+2}$ for related client

Figure 4. System design of the proposed protocol

function ProducePr (seed,q,qroot,n);
Input : seed, q, qroot, n
Output: Pr_i
Y_i *: integer,* T_i *: integer,* Pr_i *: integer;*
$Y_i \leftarrow power(seed,qroot)\ modulo\ q;$
$T_i \leftarrow power(Y_i,(qroot \times n))\ modulo\ q;$
$Pr_i \leftarrow T_i \times (qroot \times (q - n))\ modulo\ (Y_i \times q)$;
return Pr_i;

The algorithm 1 provides exposure to pseudo-code to form of the *Pr*. The value *q* is the prime number, and *qroot* is the prime root of *q*, and the *n* is the integer. The first row forms the Y_i, then the T_i, and the last is the *Pr*. The last product in this algorithm is to return *Pr*. The product of the algorithm 1 it is combined with the algorithm 2 to forming *Ec(Evk)* (for the client) or *Ec(Evk)* (for the broker).

The algorithm 2 forms the last product in the registration process. The length of *Pr* is calculated, which used to specify the length of the random characters *rc* where is get from the length of *m* subtract the length of *Pr*. After the random character *rc* is formed, the random character *rc* to be combined with *Pr*. After merging is done, the function forming the value of key *R* as long *m* as the key to do Fisher-Yates. After getting the key *R*, the result of the merger between *Pr* and *rc* as *Ev* are subjected to the Fisher-Yates Shuffle by key *R*, which then produces *Ec*.

Algorithm 2: The procedure used to form *Ec* procedure

ProduceEvConnect ();
Input: seed, q, qroot, n, m
Output : *Ec*(*Evk*)
Pr : integer, lengthPr : integer, lengthrc : integer, rc : char, Ev : char , R : list of integer, Ec : char ;
$Pr \leftarrow ProducePr(seed,q,qroot,n)$;
$lengthPr \leftarrow lengthof(Pr)$;
$lengthrc \leftarrow m - lengthPr$;
$rc \leftarrow randomstring(rc,lengthrc)$;
$Ev \leftarrow concat(Pr,rc)$;
$R \leftarrow getkeyrandom()$;
$Ec \leftarrow randomize(Ev,R)$;

B. Authentication Testing

Protocol testing is carried out in two stages, namely client authentication to the broker and mutual authentication broker to the client.

1. *Client Authentication:* on testing the client to the broker, the attention is the validity of the client to forming *Ec(Evk)*. The test using *m* as long as 35. Figure 5 indicates that the client can form *Ec(Evk)*. The first line of the figure shows the event id *Evk* that was formed. Then the second line indicates the random character is formed, and the third line shows the merger between *Evk* with random characters. The fifth line indicates the results of the third row is subjected to the Fisher-Yates Shuffle. The sixth line

```
1245800
4kP3g&RKIa34n9Gz&BThU6&5vl1
1245800&4kP3g&RKIa34n9Gz&BThU6&5vl1
35
80P364K5ha9&lnvlT&Uk4Gg32054&RI&lBz
Dev 1
80P364K5ha9&lnvlT&Uk4Gg32054&RI&lBz
```

Figure 5. Microcontroller process the proposed protocol

```
Loading Data Success
1559378439: Squanix Broker version 1.5.4  from mosquitto starting
1559378439: This Broker built by Squanix fot Dynamic Event Based
1559378439: Using default config. (No Security Mechanism)
1559378439: Opening ipv4 listen socket on port 1883.
1559378439: Opening ipv6 listen socket on port 1883.
1559378443: New Client found 192.168.230.104 on port 1883.
1559378443: Client Send Connect
1559378444: Save Data
1559378444: Succes Insert User
1559378444: Produce Publick Key
1559378444: New Client 192.168.230.104 (c1, k10, u'Dev 1') with password (null).
1559378444: No will message specified.
1559378444: Sending CONNACK to ID 80P364K5ha9&lnvlT&Uk4Gg32054&RI&lBz (0, 0)
```

Figure 6. Broker accept the client registe

```
▾ MQ Telemetry Transport Protocol, Publish Message
  ▸ Header Flags: 0x30, Message Type: Publish Message,
    Msg Len: 49
    Topic Length: 12
    Topic: Publish_key/
    Message: 5&&1p4&qZ0034Tsae4j04527788&d71uoP3
```

Figure 7. MQTT packet mutual authentication from broker

```
Received From broker :5&&lp4&qZ0034Tsae4j04527788&d71uoP3
After Decoding :474858&440&1237&107&T053dPeZjpuqsao
Evb :474858
Evk :474858
```

Figure 8. The client receives mutual authentication

is the ID, and the seventh line is *Ec(Evk)* sent. Figure 5 indicates that the event id is hidden by using Fisher-Yates shuffle. After the broker receives a registration message, the broker performs the validation. Figure 6 indicates that the broker can receive the registration message from the client. This acceptance can be seen in Figure 6 on the last line. On this line states that sending CONACK with the message (0,0). Message (0,0) in the MQTT manual states that 0x00 states the acceptance of the client [17]. From this process, it is stated that the broker can validate the event id sent by the client.

2. *Broker Authentication*: in the broker to the client testing, what the consideration is the client's ability to authenticate the broker. The broker formed an event id *Evb* from a new seed from the related client to carry out a mutual authentication process. Furthermore, *Evb* from the broker added random character so that it has a length of 35 and is subject to the Fisher-Yates shuffle. Figure 7 shows the final result of *Ec(Evb)* from the broker that sent to the relevant client. Figure 7 in the last row indicates that there is *Ec(Evb)*. After the corresponding client receives *Ec(Evb)*, the client proves the event id that was received. Figure 8 shows that the client can receive mutual authentication messages from the broker. The first line indicates the reception of *Ec(Evb)*. The second line indicates the decoding results of *Ec(Evb)*. The third and

```
1559378580: Squanix Broker version 1.5.4  from mosquitto starting
1559378580: This Broker built by Squanix fot Dynamic Event Based
1559378580: Using default config. (No Security Mechanism)
1559378580: Opening ipv4 listen socket on port 1883.
1559378580: Opening ipv6 listen socket on port 1883.
1559378582: New Client found 192.168.230.105 on port 1883.
1559378582: Client Send Connect
1559378582: Auth Failed, Drop Device From IP 192.168.230.105
1559378582: Not Authenticate User
1559378582: Sending CONNACK to Address 192.168.230.105 (0, 5)
```

Figure 9. The broker reject unauthenticated client

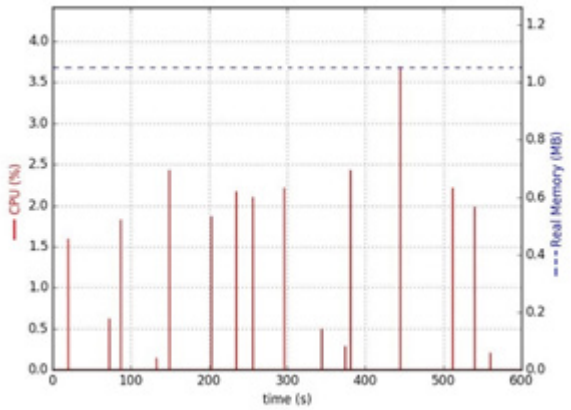

Figure 10. The broker memory and CPU usage in existing MQTT protocol

fourth line is the event id of the broker (*Evb*) and the event id of the client (*Evk*). Because *Evb* and *Evk* are the same, the client receives a broker. From this process stated that the client could validate the event id sent by the broker.

C. *Proposed Protocol Security Test*

In this test, there are two invalid clients. This invalid client tries to register to the broker by using a random event id. Figure 9 indicates that the broker receives a registration message from the client. The figure indicates that the broker receives a registration request from the client. Then the broker performs a decoding process on the registration message sent. The results of the decoding process, the broker did not find a match between the event id that sent with the event id that owned. Furthermore, the broker gives an *"Auth Failed"* signal, which states the authentication failed to the relevant client. After it is declared not accepted, the broker sends a CONACK message to the related client with message content (0,5). As in the manual MQTT manual, the message (0,5) states that the contents of 0x05 inform the client that it is not authenticated [17].

D. *Client Memory Usage*

In this test, it is intended to see the validity of the protocol built on the constraint device. This test uses ESP8266-12E as a client. This test uses the MQTT as the MQTT library for the ESP8266-12E and BigNumber as the library to support the mathematical process. The MQTT library accommodates all headers and rules on the MQTT protocol. The BigNumber library is required to provide pseudo-random numbers and other mathematical calculation processes. Using the Arduino IDE

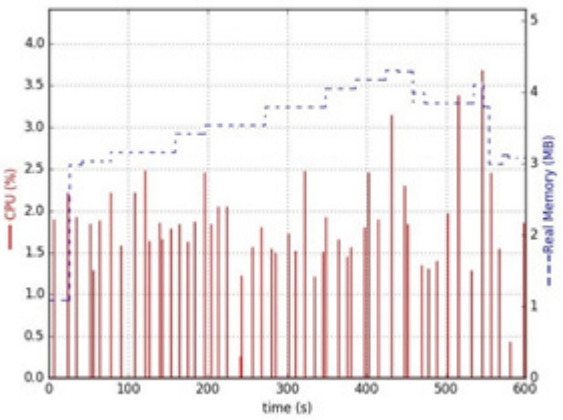

Figure 11. The broker real memory and CPU usage in the proposed MQTT protocol

, the compilation results that inform how much memory usage is done. From the results of the compilation carried out, the existing MQTT protocol uses 252.584 bytes or about 50% of the total memory of ESP8266-12E. Furthermore, the proposed protocol uses 262.404 bytes or about 52% of the full memory of ESP8266-12E. The difference in the amount of memory usage is 9.820 bytes or 2%. This memory usage is 2% greater than the MQTT protocol without security protocols. Thus, constraint devices still have more memory space to store other programs. In addition, with a difference of 2% the memory usage can reduce the amount of power used to generate memory. However, it should be noted that ESP8266-12E uses a 32-bit architecture with an instruction length of 16-bit. Thus, the compilation of code entered into ESP8266-12E is not only the size of the proposed protocol, but also includes the operating system, instructions, and address memory. Although ESP8266-12E has a flash of 512

Kb, ESP8266-12E actually only has a small space of around 32 Kb for code space (the rest is for operating system space, address, instructions).

E. *Broker Real Memory and Cpu Usage*

The specification for the MQTT broker that used is Intel i5 7200U with 4 GB RAM and using the Ubuntu OS 16.04. Memory usage is allocated to 256 MB to measure the use of real memory in the proposed and existing protocols. Tolerated memory usage of the proposed protocol is less than 30% of the allocated memory. Analysis of real memory and CPU usage on the MQTT broker have been done. This analysis is intended to identify the additional costs must be paid in implementing the proposed protocol compared to existing MQTT protocols.

Non-Attack CPU and real memory usage: The scenario of testing real memory and CPU usage in the existing MQTT protocol is with three clients who continually connect at once every five seconds. Figure 10 shows a graph of the results of real memory and CPU usage in the existing MQTT protocol taken within ten minutes. Figure 10 indicates that the maximum real memory usage is around 1.01 MB in the existing protocol. The maximum

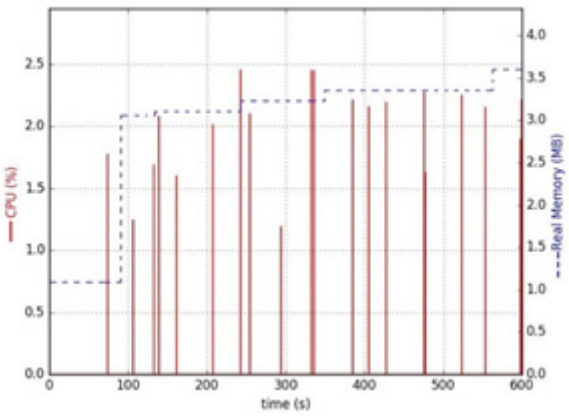

Figure 12. The broker real memory and CPU usage in an attack scenario

CPU usage used in the existing MQTT protocol is around 3.6%. Figure 10 also shows the CPU usage interval of the existing MQTT protocol. Furthermore, Figure 11 indicates a graph of the results of the real memory and CPU usage on the proposed MQTT protocol taken within ten minutes. The scenario for testing the real memory and CPU usage on the proposed MQTT protocol is that three clients are continually making connections at once every five seconds. Figure 11 indicates that the real memory usage in the proposed protocol is dynamic, and the maximum is 4.3 MB. The maximum CPU usage for the proposed MQTT protocol is around 3.7%. Figure 11 also indicates the CPU usage interval on the proposed MQTT protocol. From the results of Figure 10 and Figure 11 shows differences in the real memory and CPU usage in the existing MQTT protocol and the proposed MQTT protocol. From the results of these tests indicates that the proposed MQTT protocol uses more real memory, which is 4.3 MB (1.67% from allocated memory) compared to the existing MQTT protocol use 1.01 MB (0.39% from allocated memory). The proposed protocol uses 3.29 MB (1.28% of allocated memory) more than the existing protocol, which is still within the tolerance level (less than 30% of allocated memory). The excessive real memory usage in the proposed protocol is because MQTT brokers need additional space in calculating to verify the incoming clients. On the other hand, maximum CPU usage on the proposed MQTT protocol was close at 3.7% compared to the existing MQTT protocol, which reached 3.6%. On the other hand, CPU usage intervals on the proposed MQTT protocol are more often compared to existing protocols. The increasing CPU usage is because the CPU has an additional burden to perform calculations to validate the incoming client.

Attack CPU and real memory usage: The testing is done to see the real memory and CPU usage in the proposed protocol when the attacker tries to enter. The scenario that used is made up of two groups. The first group is a three trusted client, and the second group is two attackers. A trusted client and attacker try to register with the proposed MQTT protocol. The attackers try to attack the MQTT broker by brute force at once every two seconds. According to the previous explanation that the trusted client rules are accepted and the attackers are rejected. Figure 12 shows the real memory and CPU usage when the MQTT broker is attacked, which is taken within ten minutes. Figure 12 indicates that the real memory usage at the MQTT broker regularly increases. Maximum real memory usage is around 3.55 MB, and maximum CPU usage is 2.49%. However, when viewed from the CPU usage interval, it is lower than Figure 11. This interval difference is because the scenarios are different.

V. Conclusion

This research has proposed an authentication mechanism in the MQTT protocol. Authentication problems that occur in the MQTT protocol can be overcome using the proposed protocol. The proposed protocol uses the event based dynamic mutual authentication to accommodate the authentication mechanism for the MQTT protocol. Dynamic concepts can change the different authentication attributes for each session. The event-based concept can be applied to the constraint devices to support the authentication process. Authentication is done in two-way between the client and the broker through a secure channel. The brokers and the clients can authenticate by validating the authentication attributes that are sent. The test indicates that the proposed protocol runs on the constraint device. The brokers and the constraint devices are able to carry out the entire process in the proposed protocol. The proposed protocol can reject registration messages from the invalid clients that use brute force attacks to the broker. The implementation of the proposed protocol uses 2% memory higher than the existing MQTT protocol without authentication on the constraint device. A sample of running MQTT brokers for 10 minutes found that MQTT brokers used a maximum of 4.3 MB of real memory with maximum CPU usage at 3.7% when accepting authentic clients. The MQTT broker uses a maximum of 3.55 MB of real memory with a maximum CPU usage of 2.49% when rejecting non-authentic clients.

REFERENCES

[1] Statista, "Internet of things (iot) connected devices installed base worldwide from 2015 to 2025 (in billions)," https://www.statista.com/statistics/471264/iot-number-of-connecteddevices-worldwide/, 2016, accessed : 2019-06-30.

[2] M. Hung, "Leading the iot : Gartner insights on how to lead in a connected world," *GARTNER*, pp. 1–29, 2017.

[3] D. Evans, "The internet of things: How the next evolution of the internet is changing everything," *CISCO white paper*, vol. 1, pp. 1–11, 2011.

[4] A. F. A. Rahman, M. Daud, and M. Z. Mohamad, "Securing sensor to cloud ecosystem using internet of things (iot) security framework," in *Proceedings of the International Conference on Internet of Things and Cloud Computing*, ser. ICC '16. New York, NY, USA: ACM, 2016, pp. 79:1–79:5.

[5] F. A. Alaba, M. Othman, I. A. T. Hashem, and F. Alotaibi, "Internet of things security: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 10 – 28, 2017.

[6] S. Li and L. D. Xu, *Securing the Internet of Things*, 1st ed. Massachusetts, USA: Syngress Publishing, 2017.

[7] C. Bormann, M. Ersue, and A. Keränen, "Terminology for Constrained-Node Networks," RFC 7228, Tech. Rep. 7228, May

[8] 2014. [Online]. Available: https://rfc-editor.org/rfc/rfc7228.txt

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

[10] S. N. Firdous, Z. Baig, C. Valli, and A. Ibrahim, "Modelling and evaluation of malicious attacks against the iot mqtt protocol," in *2017 IEEE International Conference on Internet of Things (iThings) and*

[11] IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), June 2017, pp. 748–755.

[12] N. Naik, "Choice of effective messaging protocols for iot systems:

[13] Mqtt, coap, amqp and http," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, Oct 2017, pp. 1–7.

[14] P. Waher, *Learning Internet of Things*, ser. Community experience distilled. Birmingham B3 2PB, UK: Packt Publishing Ltd., 2015.

[15] S. Nolan, *Authenticated Payload Encryption Scheme for Internet of Things Systems over the MQTT Protocol*. Dublin, Ireland: Trinity Collage Dublin, The University of Dublin, 2018.

[16] G. Reiter. (2015) Securing all devices in the internet of things. https://www.ecnmag.com/article/2015/06/securing-all-devicesinternet-things. Accessed :2019-06-30.

[17] P. Miranda, M. Siekkinen, and H. Waris, "Tls and energy consumption on a mobile device: A measurement study," in *2011 IEEE Symposium on Computers and Communications (ISCC)*, June 2011, pp. 983–989.

[18] J. H. C. , Tae Ho Cho, "Adaptive energy-efficient ssl/tls method using fuzzy logic for the mqtt-based internet of things," *International Journal of Engineering and Computer Science*, vol. 5, no. 12, Nov. 2016. [Online]. Available: http://www.ijecs.in/index.php/ijecs/article/view/3229

[19] D. Ding, Z. Wang, G. Wei, and F. E. Alsaadi, "Event-based security control for discrete-time stochastic systems," *IET Control Theory Applications*, vol. 10, no. 15, pp. 1808–1815, 2016.

[20] A. Banks and R. Gupta, *MQTT Version 3.1.1*, OASIS Standard, 2014. [18] B. Russell and D. Van Duren, *Practical Internet of Things Security*. Livery place 35, Birmingham b3 2pb, UK: Packt Publishing, 2016.

[21] F. Y. Sir Ronald A. Fisher, *Statistical tables for biological, agricultural and medical research, edited by R.A. Fisher and F. Yates. 6th ed.* Edinburgh, Scotland: Oliver and Boyd, 1963, no. Ed. 6.

[22] T. K. Hazra, R. Ghosh, S. Kumar, S. Dutta, and A. K. Chakraborty, "File encryption using fisher-yates shuffle," in *2015 International Conference and Workshop on Computing and Communication (IEMCON)*, Oct 2015, pp. 1–7.

[23] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 6th ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014.

Penerbit:
Jurusan Teknik Elektro, Fakultas Teknik, Universitas Syiah Kuala
Jl. Tgk. Syech Abdurrauf No. 7, Banda Aceh 23111
website: http://jurnal.unsyiah.ac.id/JRE
email: rekayasa.elektrika@unsyiah.net
Telp/Fax: (0651) 7554336