Egyptian Informatics Journal 21 (2020) 169–182

Contents lists available at ScienceDirect

Egyptian Informatics Journal

journal homepage: www.sciencedirect.com

Formal verification for a PMQTT protocol

Eman Elemam [a,*], Ayman M. Bahaa-Eldin [b], Nabil H. Shaker [c], Mohamed Sobh [a]

[a] Computers and Systems Eng. Dept., Faculty of Engineering, Ain Shams University, 1 El Sarayat St., Abbasseya, Cairo, Egypt
[b] Misr International University, On Leave from Ain Shams University, 1 El Sarayat St., Abbasseya, Cairo, Egypt
[c] Electronics and Comm. Dept., Faculty of Engineering, Misr International University, Gamaiet Ahmed Orabi, Al Obour, Cairo, Egypt

ARTICLE INFO

Article history:
Received 16 October 2019
Revised 7 December 2019
Accepted 21 January 2020
Available online 13 February 2020

Keywords:
IoT
MQTT
Elliptic Curve Digital Signature Algorithm
Elliptic Curve Diffie Hellman
Formal verification
ProVerif

ABSTRACT

The future of Internet of Things (IoT) foresees a world of interconnected people with every physical object in a seamless manner. The security related aspects for the IoT world are still an open field of discussion and research. The Message Queue Telemetry Transport (MQTT) application layer protocol is widely used in IoT networks. Since, MQTT standard has no mandatory requirements regarding the security services, therefore, manipulating the security related issues is different in MQTT platforms. This paper proposes a novel security protocol. It is the Protected Message Queue Telemetry Transport (PMQTT) protocol which is based on MQTT with added cryptographic primitives to offer security services for IoT systems. Moreover, a formal verification for a PMQTT protocol is conducted using the ProVerif cryptographic automated verifier tool to prove that the PMQTT protocol satisfies the intended security properties.

1. Introduction

The Internet of Things (IoT) is a brand new wave of technology that promise to make our lives better and easier. Although the current focus in the IoT industry is on the ease of use, functional properly, and low cost, there is urgent need to define, as quickly as possible, security standards in the IoT world capable of dealing with the heterogeneous requirements of the IoT environment [1].

1.1. MQTT components

The Message Queue Telemetry Transport (MQTT) protocol is considered one of the best candidate protocols for the IoT platforms since it is a real time lightweight publisher/subscriber protocol [2]. The MQTT has five main components, they are [3]:

1. The Broker: It is the server that receives and publishes messages between clients.
2. The Message: It is the container of the data that has been sent to the broker by the publisher or has been received by the subscriber from the broker.
3. The Publisher: It is the device which sends messages to the broker to update the data of certain topic(s).
4. The Subscriber: It is the device which receives messages from the broker that carry the updated status of the broker's topic(s).
5. The Topic: It is an entity on the broker where the publisher sends messages to it and the subscriber receives messages from it.

The official MQTT standard [4] released by the Organization for the Advancement of Structured Information Standards (OASIS) does not have mandatory requirements regarding the security services like authentication, confidentiality, data integrity, and access control. Currently, solving the security related issues is a project and/or implementation specific matter and there is no specific standardization to handle these issues. This paper presents a Protected MQTT (PMQTT) protocol that is based on the MQTT in conjunction with the PMQTT formal verification using the ProVerif 2.00 cryptographic protocol verifier tool to demonstrate that it satisfies the intended security properties.

* Corresponding author.
E-mail addresses: eman.elemam@gmail.com (E. Elemam), ayman.bahaa@eng.asu.eg (A.M. Bahaa-Eldin), nabil.hamdy@miuegypt.edu.eg (N.H. Shaker), Mohamed.sobh@eng.asu.eg (M. Sobh).

Peer review under responsibility of Faculty of Computers and Information, Cairo University.

Production and hosting by Elsevier

https://doi.org/10.1016/j.eij.2020.01.001
1110-8665/

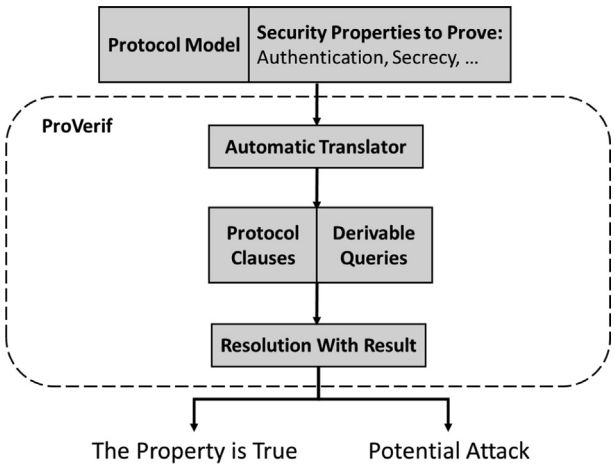

Fig. 1. The Structure of the ProVerif Tool.

1.2. Structure of ProVerif

ProVerif is an efficient automated tool used during the verification testing stage of the security protocols [5,6]. It is based on Pi calculus and it has the ability to verify the authenticity and the secrecy properties of the cryptographic security protocols. It can handle an unbounded number of sessions for the protocol under test [7]. Besides, using ProVerif, the intruder has the power to vigorously monitor the communication channel between the communicating parties where he can capture, modify, inject, and resend messages to maliciously attack the system [8]. Furthermore, ProVerif provides a tracing for the adversary attack to the system to clarify whether the protocol has security problems or not.

The inputs to the ProVerif are a model for the security protocol under verification testing in the form of Pi calculus, in conjunction with the security properties that the user needs to prove, As illustrated in Fig. 1 [7]. ProVerif takes these inputs and translates them into protocol clauses and security queries required to be verified using the automatic translator. The protocol clauses clarify the computational abilities of the adversary and the messages of the protocol. After that, ProVerif makes use of the facts – that are the initial info possessed by the attacker – to check whether the security queries are derivable from the protocol clauses or not. If the security queries can be resolved, then the protocol under test is susceptible to an attack possibility, else the modelled protocol is secure against malicious attacks.

The rest of the paper is organized in the following sections: Section 2 summarizes the previous work done to enhance the security aspects of the MQTT protocol. After that, Section 3 presents the proposed PMQTT protocol and its suggested cryptographic primitives to offer security services in IoT platforms. Then, Section 4 discusses the details of the formal verification for PMQTT to prove that it securely offers the intended security services. Finally, Section 5 summarizes the conclusions of the conducted work in conjunction with the future research directions to clarify the added values of using PMQTT in real MQTT based IoT platforms.

2. Previous work

Bhawiyuga et al. in 2017 [9] proposed a token based authentication for MQTT using a JSON Web Token (JWT) server as an authentication server [10]. They select the JWT because it has a small message size. They proposed a system architecture in which the user sends his/her username and password to the JWT authentication server. Then, the server checks its database for the validity of the user credential. If they are valid, the server sends the token to the user who saves that token in his/her local storage. When the user wants to connect to the Broker, he/she sends his/her token during the connection establishment phase to the Broker who checks the validity of the token with the JWT server. If it is valid, the Broker will allow the user to publish/subscribe to the required topics. The sequence diagram of their system is indicated in Fig. 2. It is as follows:

1. The client requests a token from the authentication server using its username and password to authenticate itself.
2. The authentication server grants token to the client after validating its credentials.
3. The client makes use of the token in the connection establishment phase with the MQTT Broker.
4. The Broker checks the validity of the token presented by the client with the authentication server.
5. The authentication server replies with the validity status of the token to the MQTT Broker.

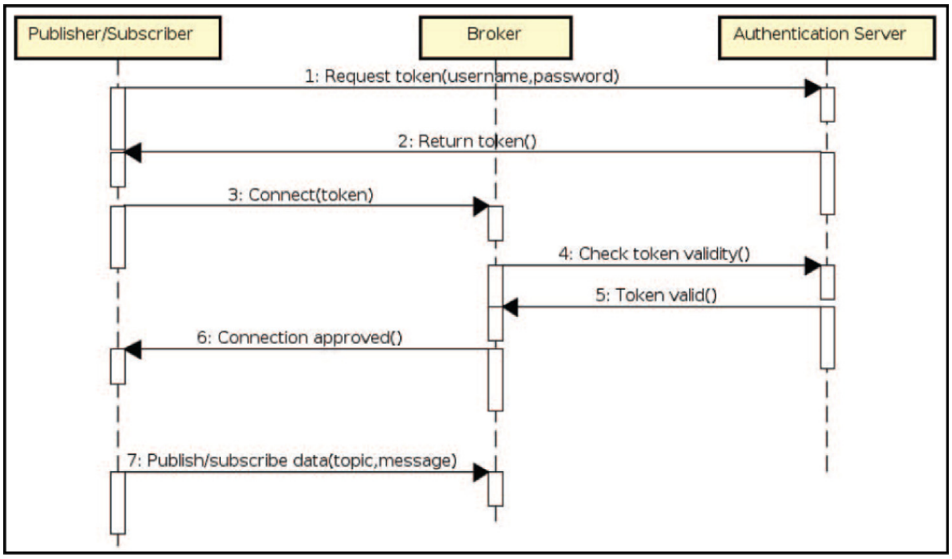

Fig. 2. The Sequence Diagram of the Token Based MQTT Publisher/Subscriber System [9].

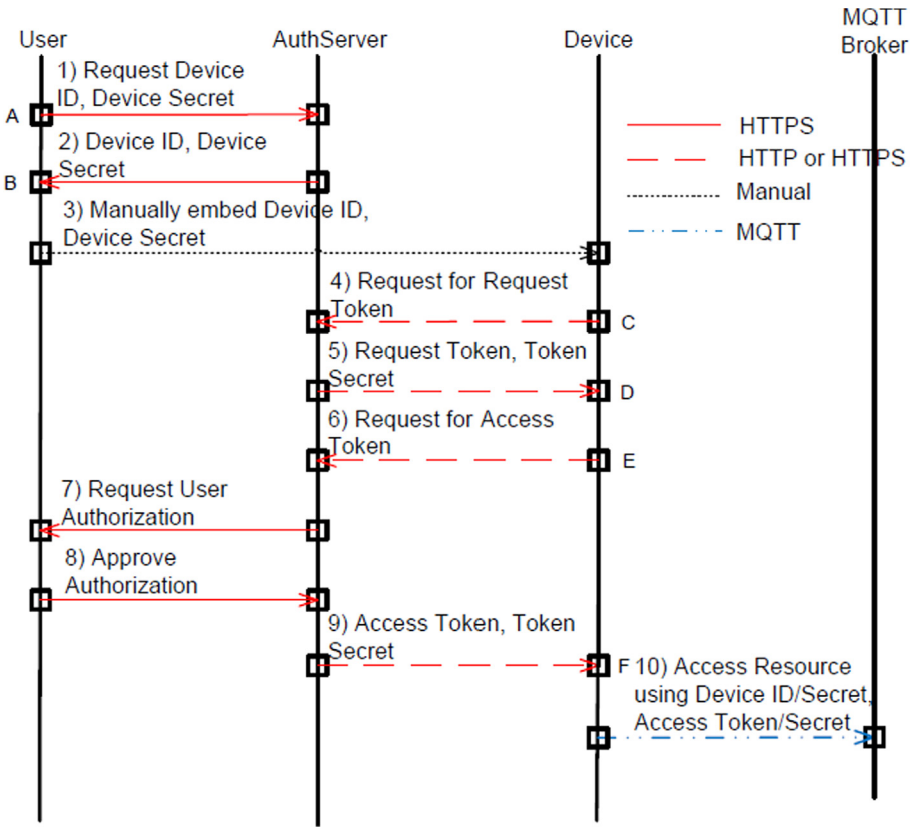

Fig. 3. Proposed Authorization Mechanism in MQTT System [3].

6. In case of valid token, the Broker approves the connection request from the client.
7. The client starts to access the topics of the Broker.

Niruntasukrat et al. [3] presented an authorization mechanism for MQTT using OAuth 1.0a [11] authorization standard. They stated that since OAuth 2.0 [12] does not support any security on top of the TLS/SSL, OAuth 1.0a is more suitable for the IoT environment than OAuth 2.0. Their idea can be summarized in that the user — who has the access credentials — will delegate some of his/her authority to some devices. Their proposed mechanism is presented in Fig. 3. It has the following steps:

1. The user sends an HTTPS message to the AuthServer to request the Device ID and its secret.
2. The AuthServer grants the device credentials to the user (Device ID and its secret).
3. The user manually embeds the device credentials into the device local memory.
4. The device sends to the AuthServer to request a Request Token. This message is digitally signed using the HMAC-SHA1 algorithm where the HKey is the Device Secret.
5. The AuthServer issues a Request Token and its secret after validating the device credentials.
6. The device sends to the AuthServer to request an Access Token. This message is digitally signed using the HMAC-SHA1 algorithm where the HKey is the Request Token Secret and the Device Secret.
7. The AuthServer requests user approval using e-mail or Short Message Service (SMS).
8. The user approves the Device ID and the access privilege scope.
9. The AuthServer grants the Access Token and its secret to the device.
10. The device can access the MQTT broker where the username will be the Device ID with concatenated timestamp and the password will be generated from the access token and the username with an HMAC-SHA1 where the HKey will be the Access Token Secret and the Device Secret.

Rahman et al. in 2018 [13] offered the usage of Key Policy/Cipher Policy Attribute Based Encryption (KP/CP ABE) using Elliptic Curve Cryptography (ECC) to get a modified MQTT protocol capable of delivering secure communication between end devices. The sequence diagram of their proposed system architecture is shown in Fig. 4. This architecture has the following stages:

1. After system initialization, both the Device and the Web Server will register in the MQTT Broker.
2. The key management phase is performed between the MQTT Broker and the IoT Device and the Web Server.
3. Both the Device and the Web Server will subscribe in the required topics of the MQTT Broker.
4. When an authorized client sends a command to the Web Server, it will encrypt this command and publish the encrypted message to the MQTT Broker.
5. The Broker will pass the encrypted message to the Device where the decryption process will be performed and the appropriate action(s) will be taken.

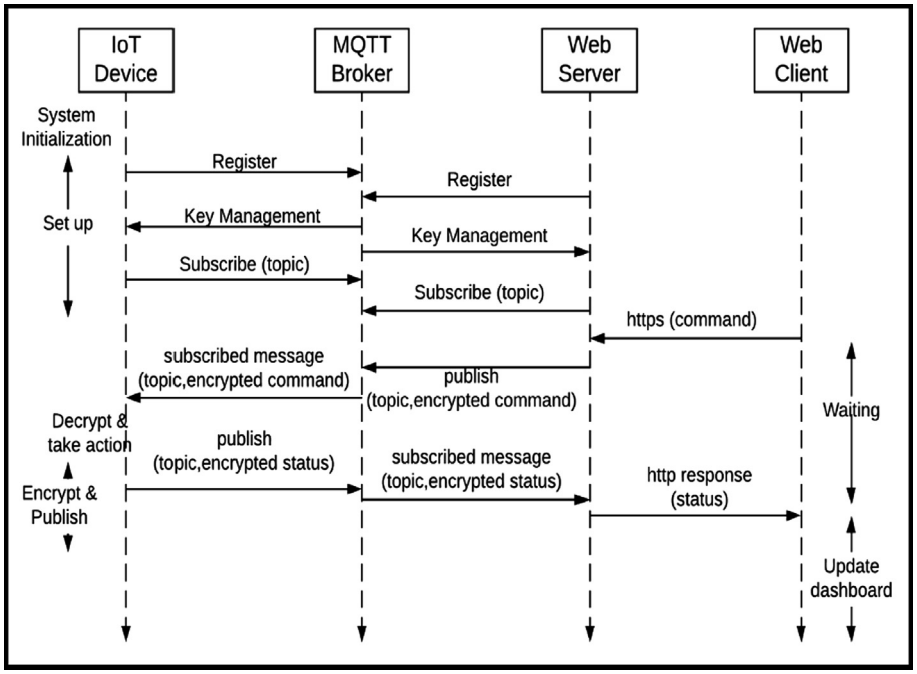

Fig. 4. The Sequence Diagram of the Proposed System Architecture [13].

6. The device will encrypt the prepared response and publish the encrypted message to the MQTT Broker.
7. The MQTT Broker will pass the encrypted response to the Web Server.
8. The Web Server will decrypt the received response.
9. The received decrypted response is delivered to the client.

Bali et al. in 2019 [14] tackled a lightweight mechanism for authentication in MQTT platforms using chaotic algorithm with block cipher. Their presented simulation model is shown in Fig. 5. They mentioned that their proposed approach depends on the high diversity of the chaotic algorithm. Also, they stated that as the diversity of the keys are high as the mission of the attacker is difficult since it will be a hard job to get back the plaintext. Thus, a more secure system is achieved. Besides, they clarified that they maintained the high diversity between the consecutive keys by properly selecting the chaotic parameters and they depends on the distance entropy when selecting these parameters.

Approaching rigorous analysis for the presented studies in this section, one can figure out that each system from the previous work covers one or two of the required security aspects in IoT environment like authentication, authorization, confidentiality, data integrity, secure key establishment and distribution, etc. Thus, there is a need for a protocol that offers a comprehensive security solution to cover the heterogeneous security requirements of the MQTT based IoT platforms.

3. Overview of the PMQTT protocol

The proposed PMQTT protocol maintains a secure communication environment for an MQTT based IoT networks. The suggested system architecture is shown in Fig. 6. It has three doctors (Dr1, Dr2, and Dr3), three Patient Controllers PC (PC1, PC2, and PC3) and a Broker. Some topics are recommended on the Broker. They are: PC1 Status, PC2 Status, PC3 Status, Dr1 Status, Dr2 Status, Dr3 Status, PC1 Data, PC2 Data, and PC3 Data.

The proposed secure PMQTT protocol has three stages as indicated in Fig. 7. The first stage is the authentication stage in which the client whether it is a publisher or a subscriber proves its identity to the Broker during the MQTT connection establishment phase. If the client fails to prove its identity to the Broker, the Broker will reject the connection request. Else, the Broker will accept the client's connection and the second stage will start. During the second phase the PC and the Dr are going to jointly establish and share a session key. Then, in the last stage, they will exchange end-to-end encrypted data using the shared session key. The flow chart of these stages is depicted in Fig. 8.

3.1. The authentication stage of the PMQTT protocol

The first stage of the proposed secured PMQTT based IoT platforms is the authentication stage where in this phase the Broker checks the claimed identity of the client whether this client was a Dr or a PC during the connection establishment phase of the MQTT protocol. If the claimed identity is true, the Broker accepts the client's connection request. Else, the Broker rejects the connection request.

The proposed authentication protocol is based on the Elliptic Curve Digital Signature Algorithm (ECDSA) [15]. Its sequence diagram is depicted in Fig. 9 and its flow chart is shown in Fig. 10. The authentication process starts at the client side with the following:

1. The client selects a random K where $1 \leq K \leq n$ and n is the order of the elliptic curve.
2. The client calculates $KG = (x_1, y_1)$ where G is the generator point of the elliptic curve, x_1 is the x-coordinate of the multiplication result, and y_1 is the y-coordinate of the multiplication result.
3. The client calculates $r = x_1 \bmod n$ to be the first part of the digital signature.
4. The client calculates $K^{-1} \bmod n$.
5. The client calculates SHA-1(MQTT_Connect_Packet) = e where the inputs to the SHA-1 are the client username and its password.
6. The client calculates $s = K^{-1}(e + dr)$ where d is the private key of the client and the resultant s is the second part of the digital signature.
7. The client attaches the digital signature (r,s) to its username to construct the modified MQTT connect packet where the client password is not added to this packet.

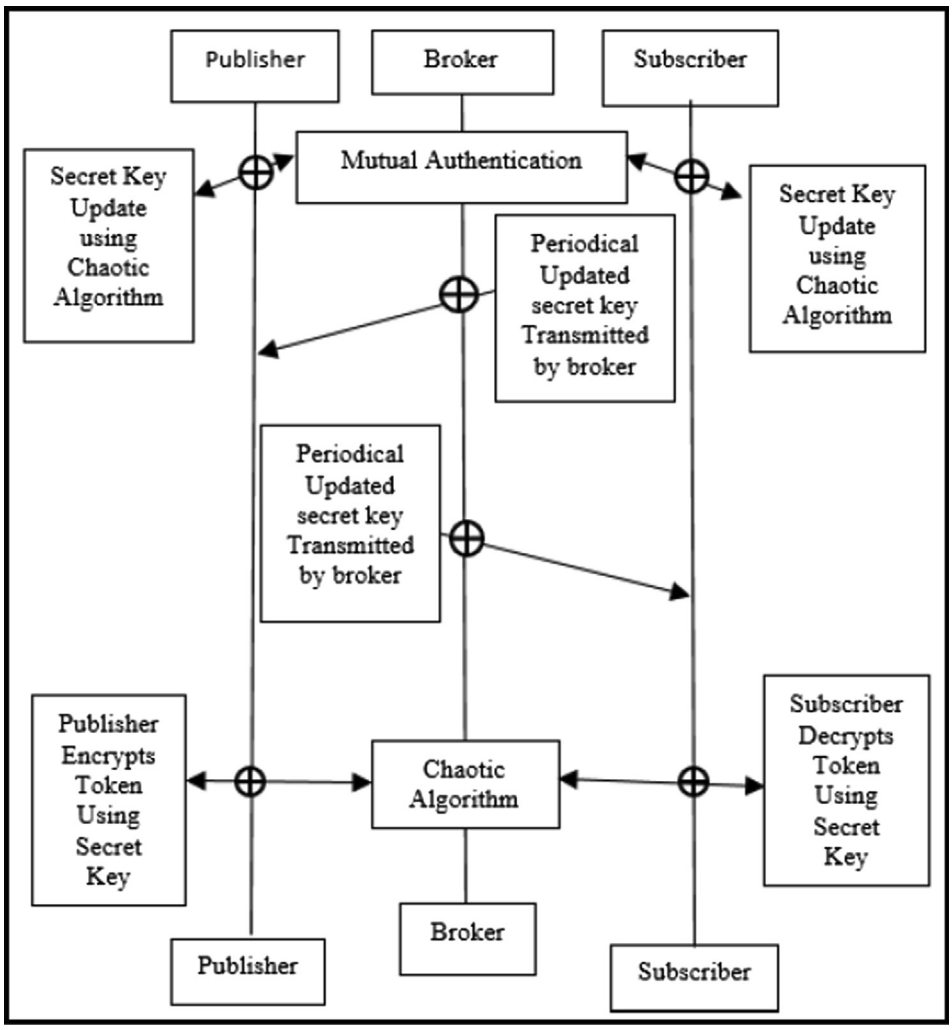

Fig. 5. The Proposed Authentication Scheme Model in the MQTT Environment [14].

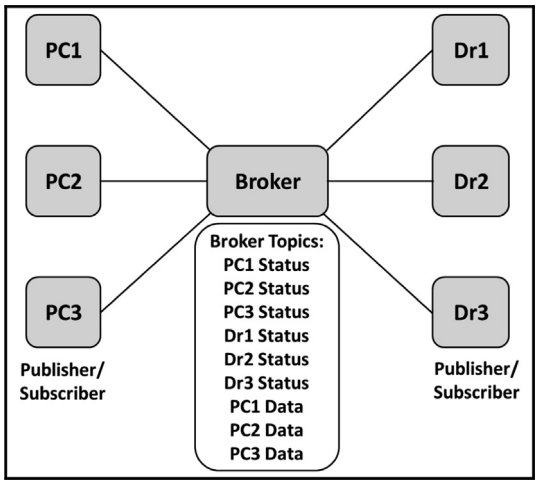

Fig. 6. The Proposed System Architecture for the PMQTT Network.

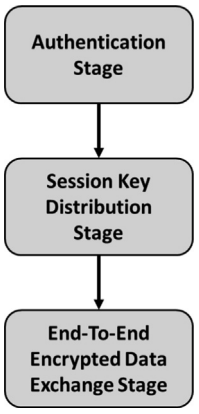

Fig. 7. The Stages of the PMQTT Based IoT Environment.

The Broker then receives the Modified_MQTT_Connect_Packet and verifies the identity of the client as follows:

8. The Broker calculates SHA-1(MQTT_Connect_Packet) = e where the payload of the MQTT_Connect_Packet is the client username and its password that is retrieved from the database of the Broker.
9. The Broker calculates $w = s^{-1} \bmod n$ where s is the second part of the digital signature that is attached to the received Modified_MQTT_Connect_Packet.
10. The Broker calculates $u_1 = ew \bmod n$ and $u_2 = rw \bmod n$ where r is the first part of the digital signature that is attached to the received Modified_MQTT_Connect_Packet.

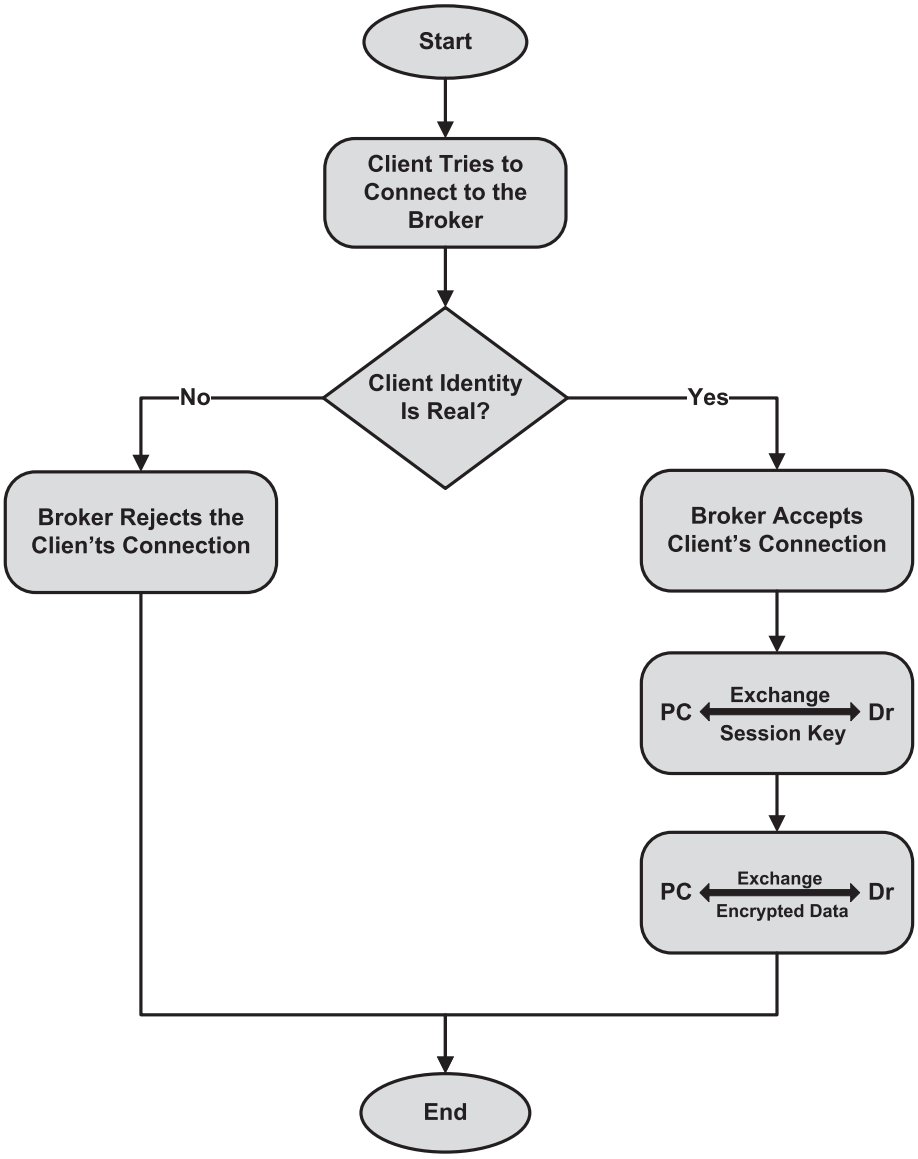

Fig. 8. Flow Chart for the Stages of the PMQTT Based IoT Platform.

11. The Broker calculates $X = (x_1, y_1) = u_1 G + u_2 Q$ where Q is the Public key of the client that is pre-embedded in the Broker.
12. The Broker checks if $v = x_1 \bmod n = r$ then the claimed client identity is true and the Broker accepts the client's connection request. Else, the Broker rejects the client's connection request.

Conducting careful investigations for the proposed authentication protocol, one can deduce that the Modified_MQTT_Connect_Packet has only the username and it has no password field. Although the password is used at the client side in the generation of the digital signature when hashing the MQTT_Connect_Packet using SHA-1 in step 5. Moreover, the client password is used at the Broker side when hashing the MQTT_Connect_Packet at step 8 where the client password is retrieved from the Broker internal database and the username is extracted from the received Modified_MQTT_Connect_Packet. Accordingly, the username may be transferred from the client to the Broker in plain text without the need for the burden of the Secure Socket Layer (SSL) in the transport layer.

3.2. The key establishment and distribution stage of the PMQTT protocol

The second stage of the proposed secured protocol for the PMQTT based IoT platforms is the key establishment and distribution stage. Through this phase, the PC and the Dr are going to share in the establishment of a symmetric secret session key based on the Elliptic Curve Diffie Hellman (ECDH) key agreement protocol [16]. This shared secret key will be used in encrypting and decrypting the specific data related to the patient to whom this PC is attached.

The suggested sequence diagram of the proposed key establishment and distribution protocol between PC1 and Dr1 is illustrated in Fig. 11. It adheres to the following steps:

1. The PC1 and the Dr1 subscribe to the Dr1 Status and the PC1 Status topics on the Broker respectively.
2. PC1 selects α and Dr1 selects β where $1 \leq \alpha \leq n\text{-}1$; $1 \leq \beta \leq n\text{-}1$ and n is the order of the elliptic curve.
3. PC1 calculates $A = \alpha G$ and Dr1 calculates $B = \beta G$ where G is the generator of the elliptic curve.

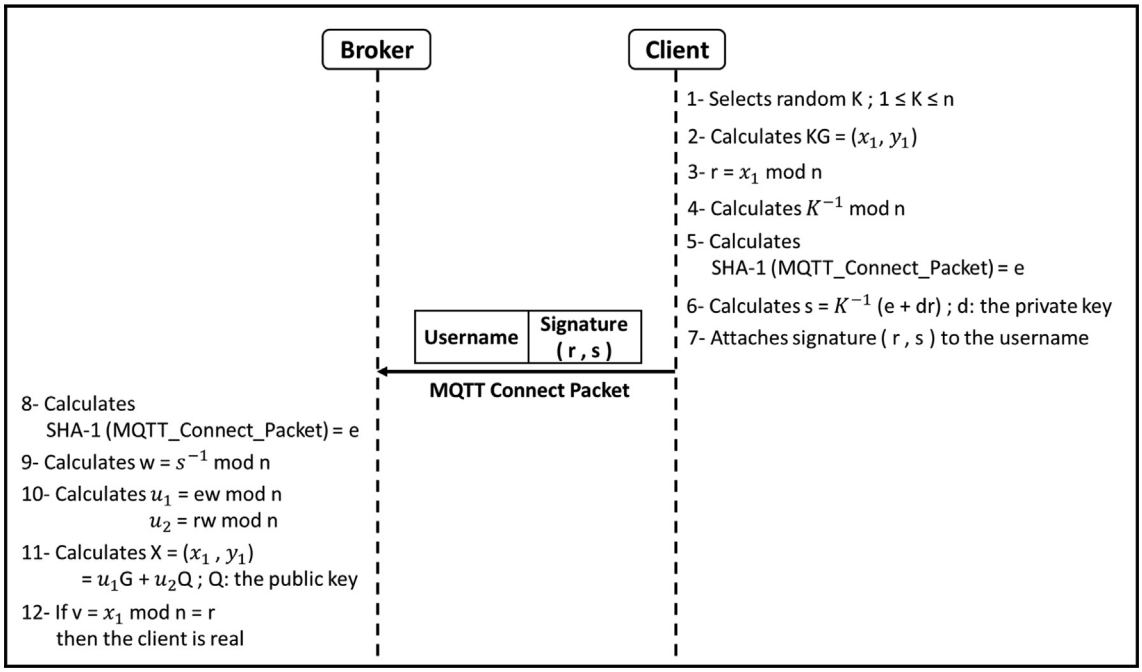

Fig. 9. The Proposed Authentication Protocol for the PMQTT Based IoT Networks.

4. PC1 publishes A to the PC1 Status topic on the Broker and Dr1 publishes B to the Dr1 Status topic on the Broker.
5. PC1 receives B and Dr1 receives A.
6. PC1 produces the shared secret key by multiplying the received B by α that is picked in step 2 and Dr1 produces the shared secret key by multiplying the received A by β that is picked in step 2.

Approaching rigorous examinations for the suggested key agreement protocol, one can conclude that, following this protocol the Dr and the PC are able to share a secret key over an insecure channel. Thanks to the elliptic curve discrete logarithm problem, the Broker cannot discover the shared secret between the PC and the Dr. Consequently, the transactions of this protocol may securely occur over untrusted network. Moreover, the Broker of the PMQTT network may reside over the cloud and no need to have it in-house at the medical center. Accordingly, the burden of the network cost can be reduced without affecting the quality of the offered service of the proposed system.

3.3. The confidentiality stage of the PMQTT protocol

The third stage of the proposed protocol for the PMQTT based IoT platforms is the end-to-end encrypted data exchange stage. During this phase, the Dr and the PC are interchanging ciphered details related to the status of the patient through the Broker. The Dr and the PC are using symmetric cipher Advanced Encryption Standard (AES) encryption scheme [17] based on the established shared secret key of the key establishment and distribution stage of the suggested secure protocol.

The suggested sequence diagram of the proposed end-to-end symmetric cipher between PC1 and Dr1 is demonstrated in Fig. 12. It passes through the following steps:

1. Both the PC1 and the Dr1 subscribe to the PC1 Data topic on the Broker.
2. The PC1 encrypts the particular data related to the status of the patient to whom PC1 is attached using the AES symmetric encryption algorithm based on the resultant shared secret key of the key establishment and distribution phase between PC1 and Dr1.
3. The PC1 publishes its encrypted data to the PC1 Data topic on the Broker.
4. The Dr1 receives the published data from the PC1 Data topic of the Broker.
5. The Dr1 decrypts and analyzes the received data using the AES symmetric decryption algorithm based on the resultant shared key of the key establishment and distribution phase between PC1 and Dr1.

Conducting attentive examinations for this system, it worth noting that, the Broker has no idea about what is the shared key between the Dr and the PC as indicated in the key establishment and distribution phase of the PMQTT protocol. Accordingly, the hospital or the medical center may safely choose to build their Broker over the cloud and the communication channel between the Dr and the PC may securely pass through untrusted network. Of course, this will have a great influence on the cost reduction of the establishment of the network infrastructure of the medical center. As a consequence, this emphasizes the practical importance of the proposed PMQTT for IoT networks.

4. Formal modeling for the PMQTT protocol using ProVerif

The formal verification of the PMQTT protocol is done using ProVerif 2.00. The protocol model verification is divided into two stages:

- The first stage is to prove that the Broker can successfully authenticate the client whether it is a PC or a Dr using ECDSA algorithm.
- The second stage is to prove:
  - The secrecy of the session key generated by the PC and the Dr using the ECDH algorithm.
  - The secrecy of the message encrypted with the generated session key using a symmetric encryption algorithm.

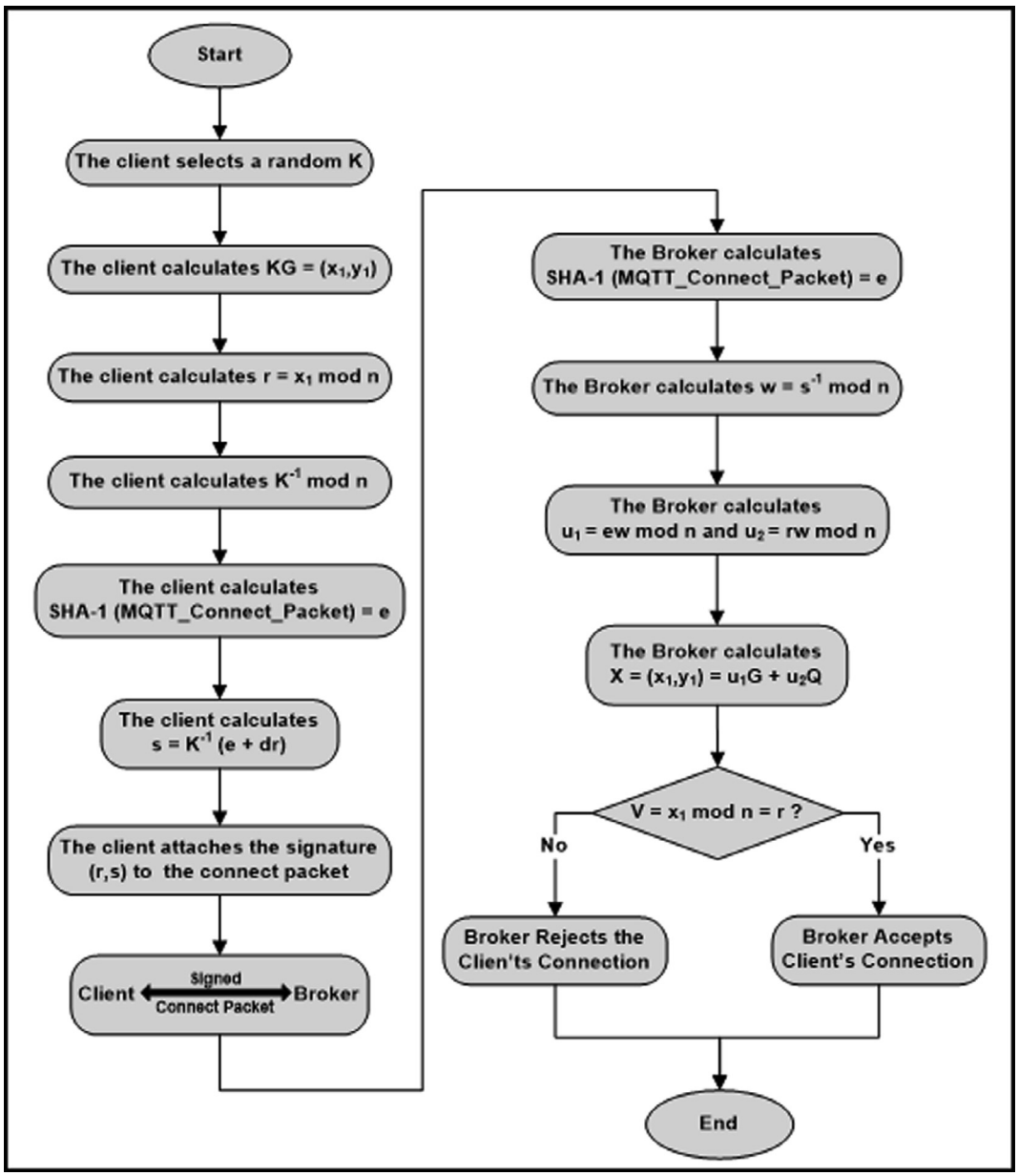

Fig. 10. The Flow Chart of the Proposed Authentication Protocol for the Telemedicine MQTT Based IoT Networks.

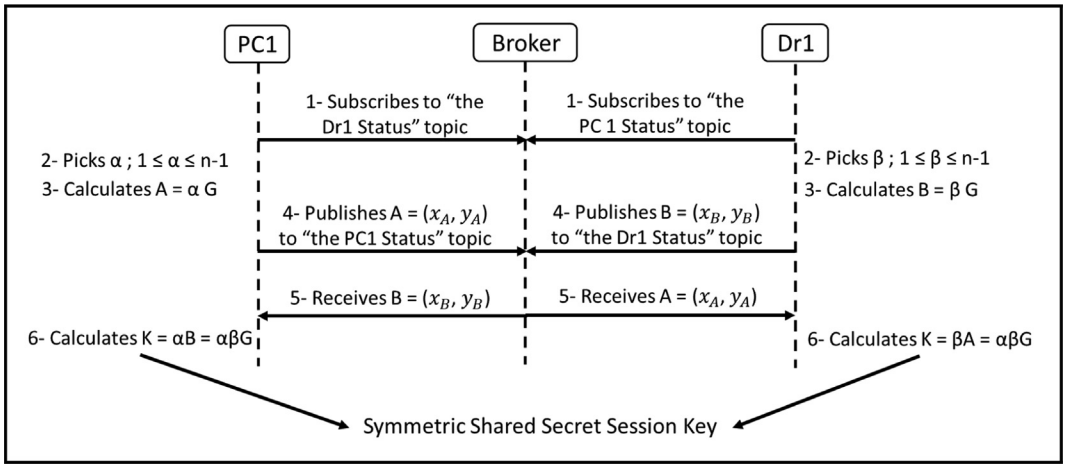

Fig. 11. The Proposed Key Establishment and Distribution Protocol for the PMQTT Based IoT Networks.

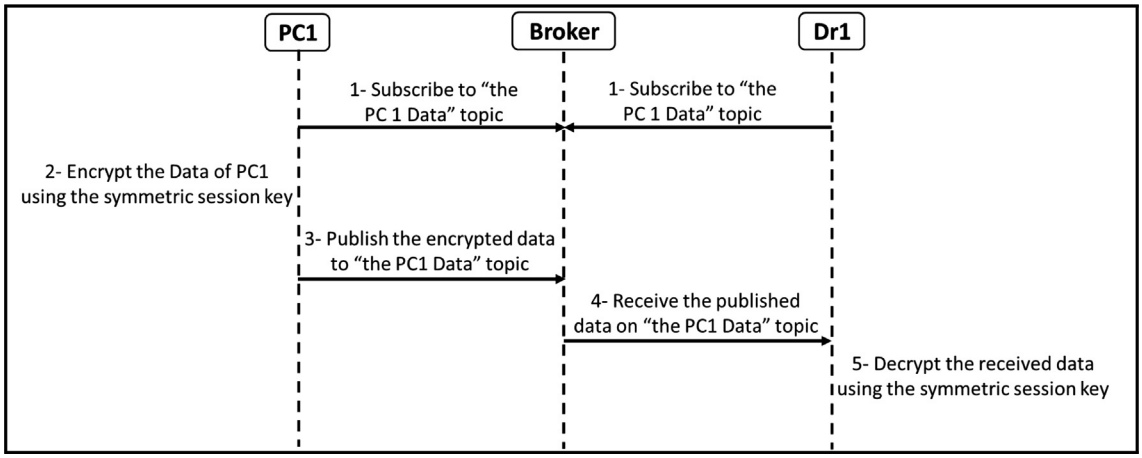

Fig. 12. The Proposed End-to-End Symmetric Cipher Protocol for the PMQTT Based IoT Platforms.

```
(*------------- Channel ------------------*)
free Client_Broker_Secure_Ch: channel [private].        (*Secure Channel*)
free Client_Broker_Public_Ch: channel.                  (*Public Channel*)

(*----------- Elliptic Curve Public Key -------------*)
type x_y_point.

fun Generate_Public_Key(bitstring): x_y_point.       (* The bitstring is the private key *)

(*------------ Constants -------------*)
const G: x_y_point.
const n: bitstring.

(*------------ Variables -------------*)
free username: bitstring.
free password: bitstring [private].

(*----------- Constructors ------------*)
fun EC_Mult(bitstring,x_y_point): x_y_point.
fun Get_X_Coordinate(x_y_point): bitstring.
fun Mod(bitstring,bitstring): bitstring.
fun Inv(bitstring,bitstring): bitstring.
fun Hash(bitstring,bitstring): bitstring.
fun Mult(bitstring,bitstring): bitstring.
fun Concat(bitstring,bitstring): bitstring.
fun EC_Concat(x_y_point, x_y_point): x_y_point.

(*---------- Events -----------*)
event beginAuthenticationCheck(bitstring).
event endAuthenticationCheck(bitstring).

(*------------ Queries -------------*)
query id: bitstring; inj-event(endAuthenticationCheck(id)) ==> inj-event(beginAuthenticationCheck(id)).
```

Fig. 13. The Declarations of the Authentication Formal Verification Stage.

The details of the conducted formal verification process is described in the following sub-sections.

4.1. The authentication verification stage

The declarations, the processes and the main process of the authentication formal verification stage are shown in Figs. 13–15 respectively. Two processes are needed to formally check the security of the client authentication in the PMQTT protocol. They are the client process and the Broker process as illustrated in Fig. 14.

In the main process of Fig. 15 the client public key is created and is used as input to unbounded number of the client process and unbounded number of the Broker process. In the client process and the Broker process the messages of the ECDSA is constructed and exchanged between the two parties using the constructors declared in Fig. 13.

The authentication query is checked to verify that the endAuthenticationCheck event is reached securely without any attack possibility after the beginAuthenticationCheck event is injected. The Proverif results to verify this query are depicted in Fig. 16. Tracing the results, one can conclude that the tested authentication query is true.

4.2. The verification stage of the session key secrecy and the encrypted message secrecy

The declarations, the processes and the main process of the formal verification for the session key secrecy and the encrypted mes-

```
(*------------ The Processes -------------*)
(*------------ The Client Process -------------*)
let Client(d:bitstring) =
    out (Client_Broker_Secure_Ch, (username, password));       (*Registration*)
    new k: bitstring;
    let z = EC_Mult(k,G) in
    let z_x = Get_X_Coordinate(z) in
    let r = Mod(z_x,n) in
    let k'= Inv(k,n) in
    let e = Hash(username, password) in
    let s = Mult(k', Concat(e, Mult(d,r))) in
    out (Client_Broker_Public_Ch, (username, r, s)).

(*------------ The Broker Process -------------*)
let Broker(Q:x_y_point) =
    in (Client_Broker_Secure_Ch, password_at_Broker: bitstring);
    in (Client_Broker_Public_Ch, (username_rx: bitstring, r_rx: bitstring, s_rx: bitstring));
    event beginAuthenticationCheck(username_rx);
    let e' = Hash(username_rx, password_at_Broker) in
    let w = Inv(s_rx, n) in
    let u1 = Mod(Mult(e',w), n) in
    let u2 = Mod(Mult(r_rx,w), n) in
    let x = EC_Concat( EC_Mult(u1,G), EC_Mult(u2,Q)) in
    let v' = Get_X_Coordinate(x) in
    let v = Mod(v', n) in
    if v = r_rx then
        event endAuthenticationCheck(username_rx)
    else 0.
```

Fig. 14. The Processes of the Authentication Formal Verification Stage.

```
(*------------ The Main Process -----------*)
process
    new client_private_key: bitstring;
    let client_public_key = Generate_Public_Key(client_private_key) in
    !Client(client_private_key) | !Broker(client_public_key)
```

Fig. 15. The Main Process of the Authentication Formal Verification Stage.

```
Process:
{1}new client_private_key: bitstring;
{2}let client_public_key: x_y_point = Generate_Public_Key(client_private_key) in
(
    {3}!
    {4}out(Client_Broker_Secure_Ch, (username,password));
    {5}new k: bitstring;
    {6}let z: x_y_point = EC_Mult(k,G) in
    {7}let z_x: bitstring = Get_X_Coordinate(z) in
    {8}let r: bitstring = Mod(z_x,n) in
    {9}let k': bitstring = Inv(k,n) in
    {10}let e: bitstring = Hash(username,password) in
    {11}let s: bitstring = Mult(k',Concat(e,Mult(client_private_key,r))) in
    {12}out(Client_Broker_Public_Ch, (username,r,s))
) | (
    {13}!
    {14}in(Client_Broker_Secure_Ch, password_at_Broker: bitstring);
    {15}in(Client_Broker_Public_Ch, (username_rx: bitstring,r_rx: bitstring,s_rx: bitstring));
    {16}event beginAuthenticationCheck(username_rx);
    {17}let e': bitstring = Hash(username_rx,password_at_Broker) in
    {18}let w: bitstring = Inv(s_rx,n) in
    {19}let u1: bitstring = Mod(Mult(e',w),n) in
    {20}let u2: bitstring = Mod(Mult(r_rx,w),n) in
    {21}let x: x_y_point = EC_Concat(EC_Mult(u1,G),EC_Mult(u2,client_public_key)) in
    {22}let v': bitstring = Get_X_Coordinate(x) in
    {23}let v_10: bitstring = Mod(v',n) in
    {24}if (v_10 = r_rx) then
    {25}event endAuthenticationCheck(username_rx)
)

-- Query inj-event(endAuthenticationCheck(id)) ==> inj-event(beginAuthenticationCheck(id))
Completing...
Starting query inj-event(endAuthenticationCheck(id)) ==> inj-event(beginAuthenticationCheck(id))
RESULT inj-event(endAuthenticationCheck(id)) ==> inj-event(beginAuthenticationCheck(id)) is true.
```

Fig. 16. The Results for the Authentication Formal Verification Stage.

```
(*-------------- Channel -------------------*)
free PC_Broker_Ch: channel.
free Dr_Broker_Ch: channel.

type x_y_point.

(*------------ Shared Key Encryption -------------*)
fun enc(bitstring, bitstring): bitstring.                    (* Symmetric Key Constructor *)
reduc forall x:bitstring, y:bitstring; dec(enc(x,y), y) = x.    (* Symmetric Key Destructor *)

(*--------------- Constructors -------------------*)
fun EC_Mult(bitstring,x_y_point): x_y_point.
fun x_y_point_to_bitstring(x_y_point): bitstring.
fun bitstring_to_x_y_point(bitstring): x_y_point.

(*------------ Constants -------------*)
const G: x_y_point.          (* Elliptic Curve GENERATOR *)

(*------------ Variables -------------*)
free msg: bitstring [private].
free session_key: bitstring [private].

(*---------- Test whether msg is secret ----------*)
query attacker(session_key).
query attacker(msg).
```

Fig. 17. The Declarations of the Formal Verification for the Session Key Secrecy and the Encrypted Message Secrecy.

sage secrecy are shown in Figs. 17–19 respectively. Three processes are needed to formally check the secrecy of the generated session key and the secrecy of the encrypted message in the PMQTT protocol. They are the PC, the Dr and the Broker process as illustrated in Fig. 18.

In the main process of Fig. 19 unbounded number of the PC process is created in parallel with unbounded number of the Dr process in parallel with unbounded number of the Broker process. In the PC process, the Dr process and the Broker process the messages of the ECDH is constructed and exchanged between the PC and the Dr through the Broker using the constructors declared in Fig. 17. Hence the session key is generated between the PC and the Dr. After that, the message is symmetrically encrypted using the created session key.

The session key secrecy query is verified to check whether the created session key is kept secret between the PC and the Broker or an adversary can reveal it. Moreover, the message secrecy query is tested too to verify that only the Dr is the one who can read the messages sent by the PC and vice versa. The Proverif results to verify those queries are illustrated in Fig. 20. Conducting attentive investigation for the results, one can deduce that both the message secrecy and the session key secrecy queries are verified by Proverif as true. So, the channel between the Dr and the PC through the Broker is reliable from a security point of view and they can exchange data securely over it.

4.3. The security goals of the PMQTT protocol

Approaching rigorous examination for the security queries results of the automated verifier tool ProVerif, it is clear that the following security goals are attainable by the PMQTT protocol:

1. The Client Successful Authentication: The client identification is successfully authenticated by the Broker. This is cleared by the correctness of the following query: inj-event(endAuthenticationCheck(id)) ==> inj-event(beginAuthenticationCheck(id))
2. The Session Key Secrecy: The value of the session_key is only known to the PC and the Dr. This is illustrated by the attacker failure to resolve the session key as indicated by the following query: query attacker(session_key)
3. The Message Secrecy: The msg contents can only be read by the PC and the Dr. This is depicted by the adversary failure to reveal the contents of the exchanged messages between the PC and the Dr as displayed by the following query: query attacker(msg)

Consequently, the proposed PMQTT protocol is formally verified and the security of the channels between the PCs and the Drs. over the Broker are maintained. Thus, the proposed PMQTT protocol can be implemented over untrusted network offering secure transactions between the PCs and their Drs. Even the Broker may reside over the cloud and of course this will have a great effect on the cost reduction for applying the proposed PMQTT protocol on a real MQTT based IoT network.

5. Conclusions and future work

Approaching careful analysis for the presented protocol, one can conclude that it has the following strength points:

1. It comes up with an authentication approach without introducing extra server for the MQTT based IoT network. So, there is no server dedicated for the authentication process. Instead, the already found Broker of the MQTT system can handle the proposed authentication scheme.
2. It comes up with an authentication scheme in which a signature is attached to the username to construct a new modified MQTT connect packet. The payload of this modified connect packet has no user password and this password is used in the production of the digital signature. Accordingly, the username of the payload can securely transferred from the client to the Broker in plain text without the need for the burden of the SSL in the transport layer.

```
(*--------- The Processes -------------*)

(*---------- The Patient Controller Process -----------*)
let PC =
    new alpha: bitstring;
    let A' = EC_Mult(alpha, G) in
    let A = x_y_point_to_bitstring(A') in
    out (PC_Broker_Ch, A);
    in (PC_Broker_Ch, B_rx: bitstring);
    let B'_rx = bitstring_to_x_y_point(B_rx) in
    let k_pc = EC_Mult(alpha, B'_rx) in
    let (= session_key) = x_y_point_to_bitstring(k_pc) in
    out (PC_Broker_Ch, enc(msg,session_key)).

(*---------- The Doctor Process -----------*)
let Dr =
    new beta: bitstring;
    let B' = EC_Mult(beta, G) in
    let B = x_y_point_to_bitstring(B') in
    out (Dr_Broker_Ch, B);
    in (Dr_Broker_Ch, A_rx: bitstring);
    let A'_rx = bitstring_to_x_y_point(A_rx) in
    let k_dr = EC_Mult(beta, A'_rx) in
    in (Dr_Broker_Ch, m: bitstring);
    let (= session_key) = x_y_point_to_bitstring(k_dr) in
    let msg_rx = dec(m,session_key) in 0.

(*----------- The Broker Process -----------*)
let Broker =
    in(PC_Broker_Ch, A''_pc: bitstring);
    out(Dr_Broker_Ch, A''_pc);
    in(Dr_Broker_Ch, B''_dr: bitstring);
    out(PC_Broker_Ch, B''_dr);
    in(PC_Broker_Ch, m_pc: bitstring);
    out(Dr_Broker_Ch, m_pc);
    in(Dr_Broker_Ch, m_dr: bitstring);
    out(PC_Broker_Ch, m_dr).
```

Fig. 18. The Processes of the Formal Verification for the Session Key Secrecy and the Encrypted Message Secrecy.

```
(*---------- The Main Process ------------*)
process
    !PC | !Broker | !Dr
```

Fig. 19. The Main Process of the Formal Verification for the Session Key Secrecy and the Encrypted Message Secrecy.

3. It satisfies the requirements of the confidentiality service by presenting an end-to-end encryption scheme over PMQTT environment.
4. It gives an opportunity for the Broker to reside over the cloud and extend the PMQTT system over untrusted network where the two communicating parties can exchange encrypted data using a session key that is hold only by those communicating parties.
5. The presented PMQTT security protocol is formally verified using the cryptographic automated verifier ProVerif and the queries regarding the authenticity and the secrecy are proved to be true such that following the proposed protocol, the Broker can successfully authenticate the identity of the client. Furthermore, the secrecy of the session key and the encrypted message are maintained.

Consequently, the presented PMQTT protocol is suitable for providing secure services for the MQTT based IoT platforms. Future research directions may be explained as follows:

1. The performance of the presented PMQTT need to be analyzed in a practical MQTT network.

```
Process:
(
    {1}!
    {2}new alpha: bitstring;
    {3}let A': x_y_point = EC_Mult(alpha,G) in
    {4}let A: bitstring = x_y_point_to_bitstring(A') in
    {5}out(PC_Broker_Ch, A);
    {6}in(PC_Broker_Ch, B_rx: bitstring);
    {7}let B'_rx: x_y_point = bitstring_to_x_y_point(B_rx) in
    {8}let k_pc: x_y_point = EC_Mult(alpha,B'_rx) in
    {9}let =session_key = x_y_point_to_bitstring(k_pc) in
    {10}out(PC_Broker_Ch, enc(msg,session_key))
) | (
    {11}!
    {12}in(PC_Broker_Ch, A''_pc: bitstring);
    {13}out(Dr_Broker_Ch, A''_pc);
    {14}in(Dr_Broker_Ch, B''_dr: bitstring);
    {15}out(PC_Broker_Ch, B''_dr);
    {16}in(PC_Broker_Ch, m_pc: bitstring);
    {17}out(Dr_Broker_Ch, m_pc);
    {18}in(Dr_Broker_Ch, m_dr: bitstring);
    {19}out(PC_Broker_Ch, m_dr)
) | (
    {20}!
    {21}new beta: bitstring;
    {22}let B': x_y_point = EC_Mult(beta,G) in
    {23}let B: bitstring = x_y_point_to_bitstring(B') in
    {24}out(Dr_Broker_Ch, B);
    {25}in(Dr_Broker_Ch, A_rx: bitstring);
    {26}let A'_rx: x_y_point = bitstring_to_x_y_point(A_rx) in
    {27}let k_dr: x_y_point = EC_Mult(beta,A'_rx) in
    {28}in(Dr_Broker_Ch, m: bitstring);
    {29}let =session_key = x_y_point_to_bitstring(k_dr) in
    {30}let msg_rx: bitstring = dec(m,session_key) in
    0
)

-- Query not attacker(session_key[])
Completing...
Starting query not attacker(session_key[])
RESULT not attacker(session_key[]) is true.
-- Query not attacker(msg[])
Completing...
Starting query not attacker(msg[])
RESULT not attacker(msg[]) is true.
```

Fig. 20. The Results for the Formal Verification of the Session Key Secrecy and the Encrypted Message Secrecy.

2. Since the Broker is a single point of failure in the presented system, another hot redundant backup Broker may be added to the PMQTT based IoT system. Consequently, there must be some sort of database synchronization between the two Brokers.

References

[1] Riahi Sfar A, Natalizio E, Challal Y, Chtourou Z. A roadmap for security challenges in the Internet of Things. Digit Commun Netw, Apr 2018;4 (2):118–37.

[2] MQTT IoT Protocol complete Tutorial – How it Works with a demo, 1Sheeld | All Arduino shields on your Smartphone, 04-Jul-2018.

[3] Niruntasukrat A, Issariyapat C, Pongpaibool P, Meesublak K, Aiumsupucgul P, Panya A. Authorization mechanism for MQTT-based Internet of Things. In: 2016 IEEE International Conference on Communications Workshops (ICC), Kuala Lumpur, Malaysia; 2016, pp. 290–295.

[4] MQTT Version 5. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. [Accessed: 26-Aug-2019].

[5] Mahmood K, Chaudhry SA, Naqvi H, Kumari S, Li X, Sangaiah AK. An elliptic curve cryptography based lightweight authentication scheme for smart grid communication. Future Gener Comput Syst 2018;81:557–65.

[6] Ryu J, Lee H, Kim H, Won D. Secure and efficient three-factor protocol for wireless sensor networks. Sensors 2018;18(12):4481.

[7] Blanchet B, Smyth B, Cheval V, Sylvestre M. ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial.

[8] Usha S, Kuppuswami S, Karthik M. A new enhanced authentication mechanism using session key agreement protocol. Cybern Inf Technol 2018;18(4):61–74.
[9] Bhawiyuga A, Data M, Warda A. Architectural design of token based authentication of MQTT protocol in constrained IoT device. In: 2017 11th International Conference on Telecommunication Systems Services and Applications (TSSA), Lombok, 2017, pp. 1–4.
[10] JSON Web Token (JWT). [Online]. Available: https://tools.ietf.org/html/rfc7519. [Accessed: 26-Aug-2019].
[11] The OAuth 1.0 Protocol. [Online]. Available: https://tools.ietf.org/html/rfc5849.
[12] Hardt D., The OAuth 2.0 Authorization Framework, RFC Editor, RFC6749, Oct. 2012.
[13] Rahman A, Roy S, Kaiser MS, Islam MdS. A Lightweight Multi-tier S-MQTT Framework to Secure Communication between low-end IoT Nodes. In: 2018 5th International Conference on Networking, Systems and Security (NSysS), Dhaka, Bangladesh; 2018. p. 1–6.
[14] Bali RS, Jaafar F, Zavarasky P. Lightweight authentication for MQTT to improve the security of IoT communication. In: Proceedings of the 3rd International Conference on Cryptography, Security and Privacy – ICCSP '19, Kuala Lumpur, Malaysia; 2019. p. 6–12.
[15] Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). [Online]. Available: https://tools.ietf.org/html/rfc6979. [Accessed: 27-Aug-2019].
[16] Use of the Elliptic Curve Diffie-Hellman Key Agreement Algorithm with X25519 and X448 in the Cryptographic Message Syntax (CMS). [Online]. Available: https://tools.ietf.org/html/rfc8418. [Accessed: 27-Aug-2019].
[17] Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS). [Online]. Available: https://tools.ietf.org/html/rfc3565. [Accessed: 27-Aug-2019].