Jurnal Rekayasa Elektrika Vol. 17, No. 2, Juni 2021, hal. 107-114
ISSN. 1412-4785; e-ISSN. 2252-620X, Terakreditasi RISTEKDIKTI No. 30/E/KPT/2018
DOI: 10.17529/jre.v17i2.20428

Secure MQTT PUF-Based Key Exchange Protocol for Smart Healthcare

Rizka Reza Pahlevi, Parman Sukarno, and Bayu Erfianto
School of Computing, Telkom University
Jl. Telekomunikasi No. 1, Terusan Buahbatu-Bojongsoang, Sukapura, Bandung 40257
e-mail: rizkarezap@telkomuniversity.ac.id

***Abstrak*—Serangan *replay* dan *eavesdropping* mengancam keamanan informasi yang dimiliki oleh perangkat kesehatan cerdas. Metode pertukaran kunci yang diautentikasi untuk menyediakan sesi kriptografi adalah upayapaling baik dalam memberikan keamanan informasi dan menyediakan autentikasi secara aman. Namun, perangkat kesehatan cerdas tidak memiliki komputasi yang cukup untuk melakukan proses kriptografi berat karena batasan perangkat tertanam yang digunakan. Kami mengusulkan protokol pertukaran kunci yang diautentikasi berbasis *physical unclonable function* (PUF) yang aman dari serangan *replay* dan *eavesdropping*. Protokol didesain dengan satu proses jabat tangan dan tiga proses autentikasi. Selanjutnya protokol usulan dievaluasi dengan menggunakan *Tamarin Prover*. Dari hasil evaluasi, protokol yang diusulkan dapat bertukar properti dengan benar antar aktor komunikasi dan valid dalam membuktikan setiap *lemma* pada serangan *replay* dan *eavesdropping*.**

Kata kunci: ***keamanan informasi, perangkat kesehatan cerdas, puf, tamarin prover, autentikasi***

***Abstract*—Replay and eavesdropping attacks threaten the information security that is held by smart healthcare devices. An authenticated key exchange method to provide cryptography sessions is the best way to provide information security and secure authentication. However, smart healthcare devices do not have sufficient computation to perform heavy cryptography processes due to the limitations of the embedded devices used. We propose an authenticated key exchange protocol based on a physical unclonable function (PUF). The proposed protocol aimed to countermeasure from replay and eavesdropping attacks. The protocol is designed with one handshake process and three authentication processes. Futhermore, the proposed protocol is evaluated using Tamarin Prover. From the results of the evaluation, our proposed protocol can exchange properties correctly between communication actors and is valid in proving each lemma in replay and eavesdropping attacks.**

Keywords: ***information security, smart healthcare device, puf, tamarin prover, authentication***

I. Introduction

In current modern era, the technology has changed human perspectives and behavior, including in the health management. In monitoring and improving the quality of individual health, smart health devices become a technological solution. A smart health device can collect health data in blood pressure, pulse rate, temperature, even heart signals [1]. Smart health devices can assist users in monitoring the quality of health to provide information on prevention actions. Smart health devices can provide health services thanks to a computer network that connects them to the cloud of health care centers [1]–[5].

In communicating towards the healthcare center cloud, smart health devices use communication protocols. One of the communication protocols used is message queue telemetry transport (MQTT). The MQTT protocol is a fast and lightweight publish-subscribe-based protocol [6]. Additionally, the MQTT protocol can be computed by constraint devices, such as smart health wearable. Smart health devices collect information from their users and send it via the MQTT protocol to the healthcare center cloud.

The information that contained in medical records and personal information on the smart health device is confidential. Therefore, information security threats on smart health devices to various cyber security threats can cause varying levels of damage to smart health devices. The result of this cyber-attack can cause loss or even damage to the information. One such attack is eavesdropping and replays. Eavesdropping on communications can result in the leakage of information transmitted by smart health devices over computer networks. The leakage of information happens because the information sent does not have an encryption security mechanism. Replay attacks can cause health device malfunctions and information integrity issues. The malfunctions and information integrity issues happen because the access to the device does not have an authentication mechanism. Therefore, improving information security and providing a level of authentication on smart health devices is receiving attention in this research area.

Received 20 March 2021; Revised -; Accepted 19 April 2021

An authenticated key exchange method to provide cryptography sessions is the best way to provide information security and secure authentication. The MQTT protocol provides a security mechanism using the asymmetric method. However, not all smart health devices have sufficient computing power to perform asymmetric cryptography processes due to the limitations of the embedded devices used [6]. Because asymmetric cryptography imposes high computational costs on smart healthcare devices, symmetric cryptography is a good candidate because it uses lower computation costs. However, symmetric cryptography uses the same key for every communication. It is a range for replay attacks. The attacker could send the same message in order to flood the service with authenticated messages. Thus, the symmetric security property cannot be the same at all times.

One method of getting a security property repeating its security property is to use a physical unclonable function (PUF). PUF can provide security properties for the authentication process on-demand and unique to each session [7]–[10]. Thus, the symmetric authentication process will be safer from eavesdropping and replay attacks. This study proposes a PUF-based and fuzzy extractor-based key exchange protocol model to accommodate smart health device authentication. We also analyzed the security of the proposed protocol using the authentication tool Tamarin Prover. In order to verify its correctness, the protocol model is written formally in notation. For verification purposes, security properties are created, namely conditions or states that are suspected of occurring when an intruder or hacker will attack.

Our contributions to this paper are:

- We propose an authenticated key exchange protocol model using PUF and a fuzzy extractor on the MQTT protocol to counter eavesdropping and replay attacks.
- We evaluated the proposed protocol using Tamarin Prover and proved the validity of the proposed protocol against each lemma that represents replay and eavesdropping attacks.

II. Related Work

Smart health devices must be able to prove their identity to carry out the authentication process. In order to provide authentication, some researchers use several techniques. Mario Barbareschia *et al*. proposed a PUF-based mutual authentication mechanism called Physical Hardware-Enabled Mutual Authentication Protocol (PHEMAP) that was capable of handling man-in-the-middle attacks [11]. Shamsoshoara also uses PUF as a security solution for the internet of things (IoT) [9]. PUF is a potential solution to counter physical attacks.

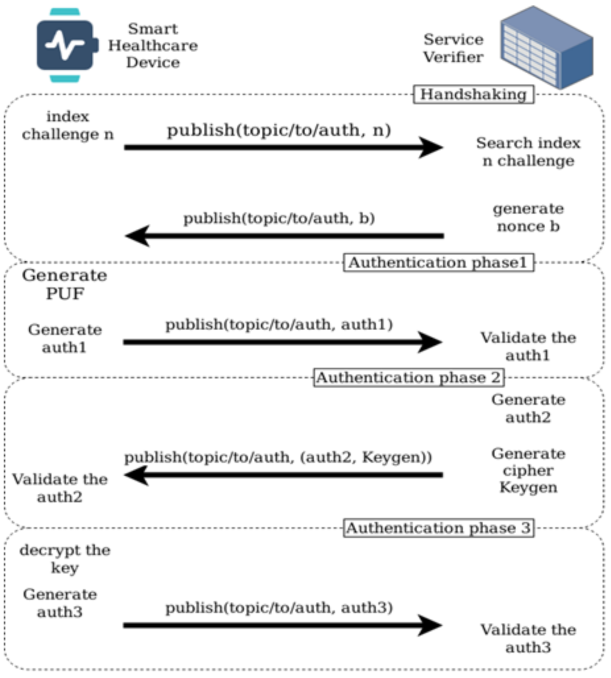
Additionally, PUF can provide different responses to different challenges. Constrained devices can also use PUF. Research conducted by Khan *et al*. proposed a lightweight, ultra-low power, re-configurable ring oscillator (RO) based PUF that can be used for authentication [12]. In this study, the proposed PUF can have better uniqueness and reliability than conventional RO. Tahavori also proposes lightweight and secure PUF-based authentication. In this study, the PUF authentication process was supported by a fuzzy extractor. The event-based authentication protocol model proposed by Pahlevi *et al*. that limited devices are capable of [6]. The event-based protocol design provides mutual authentication with a symmetric encryption mechanism.

III. Method

The proposed protocol design is formally modeled in a logical form that Tamarin Prover can accept. The proposed protocol is divided into four phases, one handshake phase and three authentication phases, as shown in Figure 1. The smart health device sends an initiation message to initiate the authentication process with the server via handshaking. After that, the smart health war generates the PUF to produce the first authentication property. The server evaluates the first authentication property of a smart health device. If it evaluated, the server then forms a second authentication property and encodes the key. The smart health device validates the second authentication property of the server. If it validated, the smart device then decrypts the key and forms a third authentication property. The server validates the third authentication property of the smart health device. If it validated, the communication will establish. Because in modeling the protocol, there is a list of terms used as presented in Table 1.

A. Prerequisite

Before this PUF-based key exchange, several stages must have occurred. We use a standard scenario on a PUF



Gambar 1. Authentication protocol design

based protocol:

1. The *Sver* server has collected the CRP and *IDumd* identifier from the *UmD* device at a stage known as the initiation phase. At this stage, the *Sver* can physically access the *UmD* device in obtaining the CRP. CPR is collected and then stored in the *Sver* storage location to be accessed later in the authentication process. Moreover, *Sver* has also collected the challenge index number from *UmD*. This challenge index number does not represent the challenge's contents but only a random sequence of challenge numbers. It should be noted that this phase must occur in a protected and controlled environment because CRP is strictly confidential.
2. The *UmD* device has been installed and operated. The *UmD* device can no longer be physically accessed by the *Sver* server either to obtain the CRP or access the PUF. In this phase, the *UmD* device can only communicate with the server *Sver* via a computer network.
3. *UmD* devices can access owned PUF to implement the proposed protocol. The response results from the PUF are not stored in the local *UmD* device. Challenges are used to generate different responses for each session.

B. Protocol Design

The proposed protocol design starts when the *UmD* device sends the *n*-th index of the challenge and *IDumd* to the *Sver*. The purpose of this transmission is to trigger *Sver* that *UmD* is asking to authenticate using the *n*-th challenge. *Sver* replies with nonce *b* and stores *IDumd* and *n*. We call this whole stage as the handshake phase. This handshake process initiates that *Sver* recognizes *IDumd* and chooses challenge *n*. The formal model at this stage is written in Table 2.

Table 2, given *UmD1* and *UmD2*, with $UmD1 \neq UmD2$, will always give $Unmanned\ (UmD1) \neq Unmanned\ (UmD2)$ facts. These facts are an effective way where each device has a different *IDumd* to give unique results. *Fr (~ n1)* is a fact that gives a nonce. The nonce is generated by $Fr\ (\sim n1) \neq Fr\ (\sim n2)$ so that the nonce is unique every time. This fact states that the challenges used to generate responses will vary. *UmD* used the nonce used in the handshake process as a challenge. Challenge on the *n*-th index, called the challenge *C '(n)*, is given an XOR operation with nonce *b* obtaining a complete challenge, formally written as $C'(n) \oplus b \Rightarrow chl$ where *chl* is the result of the XOR operation.

Given the fact that $n1 \neq n2$ and $b1 \neq b2$, then $chl1 \neq chl2$. These facts state that the challenges that are used every time are different. *UmD* uses the PUF function to get response res from *chl*, *Sver* gets response *res* from *chl* from the associated CRP database from *IDumd*. With the fact that $chl1 \neq chl2$, the fact $res1 \neq res2$ states that the resulting responses are different. *UmD* performs a fuzzy extractor generator process to get *khl* from *res*. With the fact that $res1 \neq res2$, then $khl1 \neq khl2$ means that the fuzzy extractor results are different. *UmD* forms a cipher by performing a symmetric encoding process from the concatenate *res* and *khl* with the *chl* key. At this stage, we call it the first authentication phase. The first authentication phase was formally written in Table 3.

In Table 3, PUF formation is done by calling *PUFproduce(UmD, chl)* facts, resulting in *PUFresult(UmD, chl, res)* facts. Furthermore, the formation of the fuzzy extractor's formation is done by calling the *FEgenG(UmD, res)* fact, which results in the *FEresult (UmD, res, khl)* fact. From these facts, *UmD* forms a symmetric encoding with the fact *Out(Sver, UmD, senc((res || khl), chl))*.

The results of the fact products *Out(Sver, UmD, senc((res || khl), chl))* are used by *Sver* to perform validation. *Sver* first decrypts with the symmetric *chl* key. Since the *chl* symmetric key results from the same challenge and nonce *b*, the *chl* symmetric key from the *Sver* side and the *UmD* side is the same. From the results of this decryption, *Sver* found *res* and *khl*. *Sver* performs Hamming distanace calculations on the res sent by *UmD* with the *res* that *Sver* has from the CRP database. If the Hamming distance does not cross the threshold α, then the process continues. *Sver* generates the *khls* key from the fuzzy extractor process from the response *res* with the *khl*

Tabel 1. Terms used

UmD	smart health device
IDumd	smart health device identity name
Sver	service provider to perform validation
CRP	Challenge Response Pair
DBcrp	Server-owned CRP database
FEgen	fuzzy extractor generation process
FErep	fuzzy extractor reconstruction process
h()	hash process
PUF(Ci)	process to gain PUF responses from challenge Ci
CRP()	a process to find a partner response from the challenge
TRNG()	true random number generator

Tabel 2. Handshake phase

UmD		*Sver*

Unmanned(UmD), Server(Sver), Fr(~n)
 $\rightarrow$
 Out(UmD, Sver, ~n), DBcrp(UmD, Sver, ~n) CRPpass(Sver, UmD, chl, res)		

	$\Rightarrow$	*In(UmD, Sver, n), DBcrp(UmD, Sver, n), Fr(~b)*
		 $\rightarrow$
		 Out(Sver, UmD, ~b), DBcrpB(Sver, UmD, ~b)

In(<Sver, UmD, b>), DBcrpB(Sver, UmD, b), CRPpass(Sver, UmD, chl, res)	$\Leftarrow$	

helper. With the fact that *khl1* ≠ *khl2*, then *khls1* ≠ *khls2* states that the key results from the fuzzy extractor on *Sver* are different. Following, *Sver* generates the hash for the concatenated results of *chl*, *res*, and *khls*. Then, *Sver* calls the true random number generator (TRNG) function to generate a random key as the *Key*. After that, *Sver* forms a cipher using a symmetric encryption process on the *Key* with the *chl* key. At this stage, we call it the second authentication phase. The second authentication phase was formally written in Table 4.

In Table 4, *Sver* performs the formation of a fuzzy extractor reconstruction by calling the *FEgenR(Sver, UmD, khl)* fact and producing *FEresultR(Sver, UmD, khl, khls)* fact. *Sver* generates a hash by calling the *Auth_2_2(Sver, UmD, Key, res, chl)* fact and returns Auth_2_3(*Sver, UmD, senc(Key, chl*), *h (<res, chl, khls>), Key, res, chl, khls)* fact. After that, *Sver* forms a cipher by calling the facts *Auth_2_3(Sver, UmD, senc(Key, chl), auth2, Key, res, chl, khls)* fact and generates *Out(<Sver, UmD, senc(Key, chl), auth2>)* fact.

The results of the fact products *Out(<Sver, UmD, senc(Key, chl),auth2>)* from *Sver* are used by *UmD* to perform validation and obtain the random key *Key. UmD* generates *auth2* by hashing the concatenate result *chl*, *res*, *khl* to validate *auth2*. If the two hashes are the same, *UmD* decrypts the cipher *senc(Key, chl)* using the *chl* symmetric key to get the *Key*. Then, *UmD* creates authentication *auth3* with hash concatenate *Key* and *chl*. At this stage, we call it the third authentication phase. The third authentication phase was formally written in Table 5.

In Table 5, *UmD* generates *auth3* for the third phase of authentication by calling *In(<Sver, UmD, senc(Key, chl)>), Auth2to3(Sver, UmD, chl, khls)* facts, and generating *Out(<Sver, UmD, h (<Key, chl>)>)* fact. *Sver* uses the results of the *auth3* fact product to validate *UmD*. To validate the message from *UmD*, *Sver* generates the hash of the concatenated *Key* and *chl*. If the *auth3* result from *UmD* is the same as the hash result for *Sver*, then the connection is established. All authentication phases, starting from the handshake phase, the first authentication phase, the second authentication phase, and the third authentication phase, are shown in Figure 2.

Tabel 3. First authentication phase on the *UmD* side

UmD		*Sver*
Auth_1_1(Sver, UmD, chl), PUFresult(UmD, chl, res) → *FEgenG(UmD, res), Auth_1_2(Sver, UmD, chl)*		
Auth1(Sver, UmD, chl), FEresultG(UmD, res, khl) → *Out(Sver, UmD, senc((res || khl), chl)), Auth1to2(UmD, Sver, khl)*		
	⇒	*In(<Sver, UmD, senc(<res, khl>, chl)>), Fr(~Key), Auth1to2(UmD, Sver, khl), CRPpass(Sver, UmD, chl, res)*

IV. Protocol Validation and Analysis

The protocol validation model is a fact that is used to prove that the fact-built protocol can exchange messages. We tested two protocol validations in the proposed protocol design, namely sanity validation and authentication validation. Validation of sanity is validation that aims that the proposed protocol can communicate properly. Authentication validation is validation that aims to see whether each actor can validate the intended message.

The validation of sanity is tested in the handshake phase. The handshake phase involves communication between *UmD* and *Sver*. *UmD* sends a message to *Sver* in the form of index *n*. After that, *Sver* replied by sending nonce *b* to *UmD*. To track these communications, we provide fact-

Tabel 4. Second authentication phase

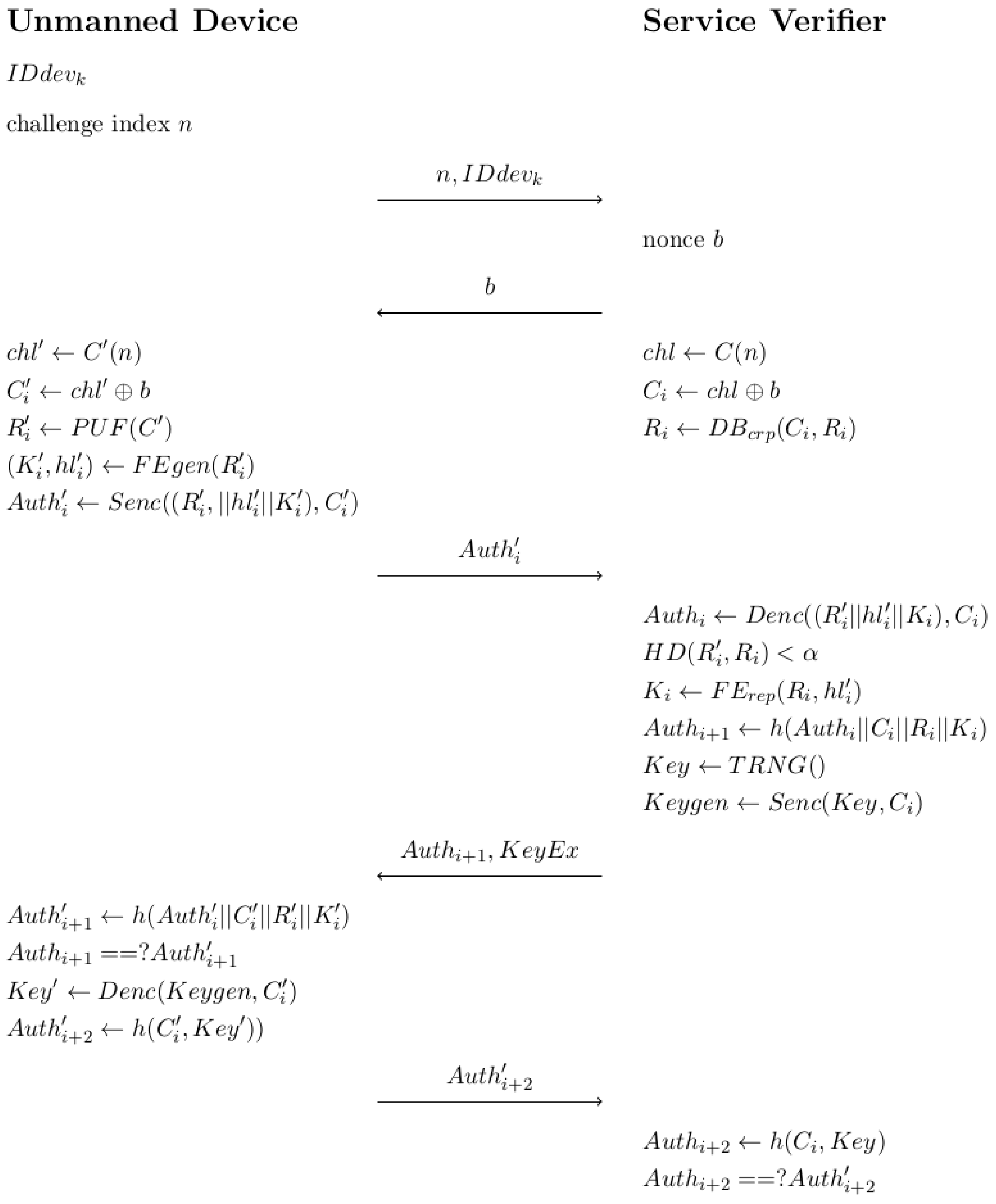
UmD		*Sver*
		Auth_2_1(Sver, UmD, Key, chl, res, khl) → *Auth_2_2(Sver, UmD, Key, res, chl), FEgenR(Sver, UmD, khl)*
		Auth_2_2(Sver, UmD, Key, res, chl), FEresultR(Sver, UmD, khl, khls) → *Auth_2_3(Sver, UmD, senc(Key, chl), h(<res, chl, khls>), Key, res, chl, khls)*
		Auth_2_3(Sver, UmD, senc(Key, chl), auth2, Key, res, chl, khls) → *Out(<Sver, UmD, senc(Key, chl), auth2>), Auth2to3(Sver, UmD, khls)*
In(<Sver, UmD, senc(Key, chl), auth2>), Auth2to3(Sver, UmD, khls), CRPpass(Sver, UmD, chl, res)	⇐	

Tabel 5. Third authentication phase

UmD		*Sver*
In(<Sver, UmD, senc(Key, chl)>), Auth2to3(Sver, UmD, chl, khls) → *Out(<Sver, UmD, h(<Key, chl>)>), Auth3to4(Sver, UmD, Key)*		
	⇒	*In(<Sver, UmD, h(<Key, chl>)>), Auth3to4(Sver, UmD, Key), CRPpass(Sver, UmD, chl, res)*

tracing. The fact-tracing provided in the handshaking phase are, *Handshaking(UmD,Sver, <'UtoS' / 'StoU', c>)*. Fact-tracking *Handshaking(UmD, Sver, <'UtoS' / 'StoU', c>)* is intended to track the communication between *UmD* and *Sver* when sending messages *c*. There are directional flags *'UtoS' / 'StoU'*, where *'UtoS'* indicates communication from *UmD* to *Sver*, and *'StoU'* indicates communication from *Sver* to *UmD*. The fact that tracing the handshake is addressed to the handshake poses in Table 2. The fact-tracking can be proven if *Sver* receives an index *n* message from *UmD*, and *UmD* receives a nonce *b* message from *Sver*. Thus, formally the two tracing facts are written in Lemma 1.

Lemma 1. In one event at a time *i*, there was an *UmD* who shook hands with *Sver* with the message *c*, and there was a *Sver* who shook hands with *UmD* with the message *c*. Formally written with


Unmanned Device
Service Verifier
$IDdev_k$
challenge index n
$n, IDdev_k$
nonce b
b
$chl' \leftarrow C'(n)$
$C'_i \leftarrow chl' \oplus b$
$R'_i \leftarrow PUF(C')$
$(K'_i, hl'_i) \leftarrow FEgen(R'_i)$
$Auth'_i \leftarrow Senc((R'_i, ||hl'_i||K'_i), C'_i)$
$chl \leftarrow C(n)$
$C_i \leftarrow chl \oplus b$
$R_i \leftarrow DB_{crp}(C_i, R_i)$
$Auth'_i$
$Auth_i \leftarrow Denc((R'_i||hl'_i||K_i), C_i)$
$HD(R'_i, R_i) < \alpha$
$K_i \leftarrow FE_{rep}(R_i, hl'_i)$
$Auth_{i+1} \leftarrow h(Auth_i||C_i||R_i||K_i)$
$Key \leftarrow TRNG()$
$Keygen \leftarrow Senc(Key, C_i)$
$Auth_{i+1}, KeyEx$
$Auth'_{i+1} \leftarrow h(Auth'_i||C'_i||R'_i||K'_i)$
$Auth_{i+1} ==? Auth'_{i+1}$
$Key' \leftarrow Denc(Keygen, C'_i)$
$Auth'_{i+2} \leftarrow h(C'_i, Key'))$
$Auth'_{i+2}$
$Auth_{i+2} \leftarrow h(C_i, Key)$
$Auth_{i+2} ==? Auth'_{i+2}$

Gambar 2. Formal model proposed protocol

$$\exists Sver, UmD, c, i.\ \ Handshaking(UmD, Sver, <'UtoS', c>)@$$
$$\wedge$$
$$\exists Sver, UmD, c, i.\ \ Handshaking(Sver, UmD, <'UtoS', c>)@i$$
$$\wedge$$
$$\exists Sver, UmD, c, i.\ \ Handshaking(UmD, Sver, <'StoU', c>)@i$$

Analysis Lemma 1. In Table 2, the *UmD* actor has *Unmanned(UmD)* and *Fr(~n)* facts which result in *Out(UmD, Sver, ~ n)* and *DBcrp(UmD, Sver, ~ n)* facts. Actor *Sver* has *In(UmD, Sver, n)*, *DBcrp(UmD, Sver, n)*, and *Fr(~b)* facts. In Lemma 1, fact-tracking *Handshaking(UmD, Sver, <'UtoS', c>)@i* tracks the communication between the *UmD* and *Sver* actors. The fact-tracking tracks whether actor *UmD* has sent *n*. After that, the actor *Sver* accepts and understands it. Tamarin evaluates this facts-tracing with proven results. This is because the two actors have each other's *DBcrp(UmD, Sver, n)* facts, which means they know each other *n*. Thereafter, actor *Sver* who already had the facts *Fr (~b),* sent it to *UmD*. In Lemma 1, fact tracking *Handshaking(UmD, Sver, <'StoU', c>) @i* communication between actor *Sver* and *UmD*. The tracking fact-tracks whether actor *Sver* has sent *b*, and actor *UmD* accepts and understands it. Tamarin evaluates these tracing facts with proven results. This is because the two actors have mutual *DBcrpB(Sver, UmD, b)* facts, which means they know each other *b*. Facts *DBcrp(UmD, Sver, n)* and *DBcrpB(Sver, UmD, b)* facts owned by actors *UmD* and *Sver* indicate that actor *Sver* has understood actor *UmD*, and vice versa, from the prerequisite process or previous initiation process.

Authentication property validation is the validation used to test the first authentication phase, second authentication phase, and third authentication phase. Authentication validation proves that each actor can accept the properties needed in the authentication process. The first lemma is to prove that *UmD* can send *chl*, *res*, and *khl* to *Sver*. The second lemma is to prove that *Sver* can send the *Key* to *UmD* and is understood. The third tracing fact is to prove that *UmD* can send the hash result of the concatenate *chl* and *Key* to *Sver*. Thus, formally the three tracking facts are written in Lemma 2, Lemma 3, and Lemma 4.

Lemma 2. In one event at a time *i*, there is an *UmD* that sends an authentication property to *Sver* in the form of a *res* message, and there is a *Sver* that receives a *res* authentication property from *UmD*. Formally written down

$$\exists Sver, UmD, res, i.\ \ AuthProtocol1(UmD, Sver, res)@i$$
$$\wedge$$
$$\exists Sver, UmD, res, i.\ \ AuthProtocol1(Sver, UmD, res)@i$$

Analysis Lemma 2. Fact-tracking in Lemma 2 is used to track communications between *UmD* actors and *Sver* at the first authentication phase stage. In Table 3, *UmD* has the *Auth1(Sver, UmD, chl)*, *FEresultG(UmD, res, hl)* facts which produces an authentication property in the form of the *Out(Sver, UmD, senc((res || khl), chl))* fact. This fact was accepted by *Sver* with the *In(<Sver, UmD, senc(<res, khl>, chl)>)* fact. Lemma 2 fact-tracking tracks whether *senc(<res, khl>, chl)* (as the property of *res*) has been submitted by the *UmD* actor and received and understood by the *Sver* actor. Tamarin evaluated the fact-tracking of Lemma 2 with proven results. This is because the *Sver* actor has *CRPpass(Sver, UmD, chl, res)* facts, which are the result of the facts in Table 2, which are also owned by the *UmD* actor. This fact shows that actors *UmD* and *Sver* have a pair of *chl* and *res*. In Table 3, *Sver* uses *chl* to do the decryption. Since the *chl* used by *UmD* and *Sver* is the same, *Sver* can get the properties sent by *UmD*. Thus, the actor *Sver* can know *<res, khl>*.

Lemma 3. In one event at a time *i*, there is an *UmD* that sends an authentication property to *Sver* in the form of a *keyA* message, and there is a *Sver* who receives an authentication property from *UmD* with a *keyA* message. Formally written down

$$\exists Sver, UmD, keyA, i.\ \ AuthProtocol2(UmD, Sver, keyA)@i$$
$$\wedge$$
$$\exists Sver, UmD, keyA, i.\ \ AuthProtocol2(Sver, UmD, keyA)@i$$

Analysis Lemma 3. The fact-tracking in Lemma 3 is used to track the *Sver* actor's communication to *UmD* in the second authentication phase. In Table 4, *Sver* has the *Auth_2_3(Sver, UmD, senc(Key, chl), auth2, Key, res, chl, khls)* facts which produces an authentication property in the form of the *Out(<Sver, UmD, senc(Key, chl)>), Auth2to3 (Sver, UmD, khls)* facts. *UmD* accepted these facts with the *In(<Sver, UmD, senc(Key, chl)>)* facts. Lemma 3 fact-tracking trace whether *senc(Key, chl)* (as property *keyA*) has been sent by actor *Sver* and received also understood by actor *UmD*. Tamarin evaluates the facts-tracking of Lemma 3 with proven results. This is because the *UmD* actor has *CRPpass(Sver, UmD, chl, res)* facts in Table 2. This fact shows that the *UmD* and *Sver* actors have a pair of *chl* and *res*. In Table 4, *UmD* uses *chl* to do the decryption. Since the *chl* used by *Sver* and *UmD* is the same, *UmD* can get the *Key* property sent by *Sver*. Thus, the *UmD* actor was able to find out the *Key* sent by the actor *Sver*.

Lemma 4. In one event at a time *i*, there is an *UmD* that sends authentication properties to *Sver* in the form of *chl* and *keyA* messages, and there is a *Sver* who receives protocol authentication from *UmD* with messages *chl* and *keyA*.

$$\exists Sver, UmD, chl, keyA, i.$$

AuthProtocol3(UmD,Sver,chl, keyA)@i
^
∃ *Sver,UmD,chl,keyA,i.*
AuthProtocol3(Sver, UmD, chl, keyA)@i

Analysis Lemma 4. Fact-tracking in Lemma 4 is used to track communications between *UmD* actors and *Sver* in the third authentication phase process. In Table 5, *UmD* has *In(<Sver, UmD, senc(Key, chl)>), Auth2to3(Sver, UmD, chl, khls)* facts which produce authentication properties in the form of *Out(<Sver, UmD, h(<Key , chl>)>), Auth3to4(Sver, UmD, Key, chl)* facts. *Sver* accepted this fact with the *In(<Sver, UmD, h(<Key, chl>)>)* fact. Lemma 4 fact tracking tracks whether *h(<Key, chl>)* (as the property *chl* and *keyA*) has been submitted by the *UmD* actor and is received and understood by actor *Sver*. Tamarin evaluated the fact-tracking of Lemma 4 with proven results. This is because the *UmD* actor has *CRPpass(Sver, UmD, chl, res)* facts from Table 2 and the *Auth3to4(Sver, UmD, Key)* facts. In Table 5, *UmD* uses *chl* and *Key* to perform the hashing process. Because the *chl* and *Key* used by *Sver* and *UmD* are the same, *Sver* can get the same hash result that *UmD* sent.

The replay attack model in the proposed protocol is used to prove that no usable message exists after the original message has been received. This validation is carried out in the first authentication phase, second authentication phase, and third authentication phase. To track this evidence, we provide fact-tracking for each phase. In general, fact tracking in each phase has the same model. The tracing facts provided are *Send(UmD, Sver, authmessage)* fact and facts *Authentic(UmD, Sver, authmessage)*. The *Send(UmD, Sver, authmessage)* fact validates that the *UmD* actor has sent the *Sver* actor an *authmessage* message. The *Authentic(UmD, Sver, authmessage)* fact validates that the *UmD* actor has received from the *Sver* actor an *authmessage* message. The actors in the role can switch each other to fulfill the tracking. Formally, the proof of the replay attack is written on Lemma 5.

Lemma 5. In one event at a time *i*, each authentication message that has been received from the *UmD* actor to the *Sver* actor with an *authmessage* message, there is an *UmD* actor who sends the *Sver* actor an *authmessage* message, and no other actor uses an acceptable *authmessage* message. Formally written with

∀ *UmD, Sver, authmessage, i.*
Authentic(UmD, Sver, authmessage) @i
⇒ *(∃ j. Send(UmD, Sver, authmessage) @j*
^ j < i
^
¬ *(∃ UmD2,Sver2,i2*
Authentic(UmD2, Sver2, authmessage) @i2
^ (i < i2)))

In Lemma 5, the timing of the incident between *UmD* and *UmD2* actors was different. The *UmD* actor performs the first execution, and then the *UmD2* actor performs the second execution by replicating the *authmessage* message.

Analysis Lemma 5. Fact-tracking on Lemma 5 is used to validate replay attacks on communication between *UmD* and *Sver* actors in Table 3, Table 4, and Table 5. In Table 3, the intended *authmessage* is *chl* on *senc(<res, khl>, chl)>)*. The Lemma 5 tracking facts in Table 3 aim to prove that no same *senc(<res, khl>, chl)>)* sent a second time can be received. Tamarin evaluated the tracking facts of Lemma 5 in Table 3 with proven results. This happens because each session using a different *chl*. With different *chl*, the resulting symmetric encryption results will be different (even though the encrypted properties are the same). Furthermore, Lemma 5 fact-tracking is in Table 4. In Table 4, the intended *authmessage* is *chl* on *senc(Key, chl)*. The tracking facts of Lemma 5 in Table 4 are intended to prove that none of the same *senc(Key, chl)* sent for the second time can be received. Tamarin evaluated the tracking facts of Lemma 5 in Table 4 with proven results. Similar to the reason for Lemma 5 in Table 4, the use of different *chl* will produce different symmetric encryption results. Furthermore, Lemma 5 tracking facts are in Table 5. In Table 5, the intended *authmessage* is *Key* and *chl* on *h(<Key, chl>)*. The Lemma 5 tracking facts in Table 5 aim that no *h(<Key, chl>)* the same sent a second time can be received. Tamarin evaluated the tracking facts of Lemma 5 in Table 5 with proven results. This happens because the hash results with different *Key* and *chl* can produce significant differences. Thus, no message can be received a second time.

The eavesdropping attack in the proposed protocol is used to prove that there is no critical information that other actors can tamper with. This validation is carried out in the first authentication phase, second authentication phase, and third authentication phase. To do this validation, we provide fact-tracking facts at each phase. In general, fact-tracking eavesdropping has the same model. The difference used is the information to be compromised. The fact *Secret(A)* validates *A* information that has been used on both actors cannot be known by actor *K(A)*. Formally, evidence of eavesdropping attacks is written on Lemma 6.

Lemma 6. In one event at a time *i,* for every secret message *secretmessage*, there is no secret message *secretmessage* known by actor *K*

∀ *Sver, UmD, secretmessage (Secret(Sver, UmD, secretmessage)*
⇒ ¬ *(∃ K(secretmessage)))*

In Lemma 6, actor *K* is an actor who plays an attacking actor. In Lemma 6, there is a secret message in the form of a *secretmessage* known only to legitimate actors.

Analysis Lemma 6. Fact-tracking on Lemma 6 are used to validate eavesdropping attacks on communication between *UmD* and *Sver* actors in Table 3, Table 4, and Table 5. In Table 3, the intended *secretmessage* are *chl*, *res*, and *khl*. The tracking facts of Lemma 6 in Table 3 aim to prove that no *chl*, *res*, and *khl* can be compromised by actor *K* at the time of communication. Tamarin evaluated the fact-tracking of Lemma 6 in Table 3 with proven results. This happens because the communication is encrypted using symmetric encryption. Validated actors can only know the *chl*, *res*, and *khl* properties. Even though the message was received by actor *K*, the actor was unable to know *res* and *khl* because he did not have the *chl* key. Furthermore, Lemma 6 tracking facts are in Table 4. In Table 4, the intended *secretmessage* is *Key*. The fact-tracking of Lemma 6 in Table 4 is intended to prove that there is no *Key* that can be compromised by actor *K* at the time of communication. Tamarin evaluated the tracking facts of Lemma 6 in Table 4 with proven results. This is because the *Key* is given symmetric encryption. Thus, even though actor *K* received a message containing an encoded *Key*, the actor was unable to obtain the *Key*. Furthermore, Lemma 6 fact-tracking is in Table 5. In Table 5, the intended *secretmessage* is *Key* and *chl*. The tracking facts of Lemma 6 in Table 5 are intended to prove that there is no *Key* and *chl* that can be compromised by actor *K* at the time of communication. Tamarin evaluated the tracking facts of Lemma 6 in Table 5 with proven results. This happens because the hash is a one-way function and cannot be reversed. Thus, even if actor *K* gets the hash message from the communication, the actor cannot get the hash compiler information.

V. Conclusion

The information on the smart health device is confidential that requires a level of security. Eavesdropping and replay attacks threaten the security of information held by smart health devices. An authenticated key exchange method to provide cryptography sessions is the best way to provide information security and provide secure authentication. However, smart health devices do not have sufficient computation to perform heavy cryptography processes due to the constrained of the embedded devices used. This study proposed a validated PUF-based authenticated key exchange protocol model in proving replay and eavesdropping attacks. We are evaluating our proposed protocol using Tamarin Prover. From the results of the evaluation, the proposed protocol can properly exchange properties between communication actors. Our proposed protocol can prove that no message can be sent twice to prove resilience to replay attack. Our proposed protocol can prove that no property can be compromised during the communication process to prove its resistance to eavesdropping attack.

References

[1] F. Wu, X. Li, L. Xu, S. Kumari, and A. K. Sangaiah,“A novel mutual authentication scheme with formal proof for smart healthcare systems under global mobility networks notion,” *Comput. Electr. Eng.*, vol. 68, pp. 107–118, May 2018.

[2] Y. Zhang, M. Qiu, C. Tsai, M. M. Hassan, and A. Alamri, “Health-cps: healthcare cyber-physical system assisted by cloud and big data,” *IEEE Syst. J.*, vol. 11, no. 1, pp. 88–95, Mar. 2017.

[3] N. Tariq, A. Qamar, M. Asim, and F. A. Khan, “Blockchain and smart healthcare security: a survey,” *Procedia Comput. Sci.*, vol. 175, pp. 615–620, Jan. 2020.

[4] S. B. Baker, W. Xiang, and I. Atkinson, “Internet of things for smart healthcare: technologies, challenges, and opportunities,” *IEEE Access*, vol. 5, pp. 26521–26544, 2017.

[5] Minahil, M. F. Ayub, K. Mahmood, S. Kumari, and A. K. Sangaiah, “Lightweight authentication protocol for e-health clouds in IoT based applications through 5G technology,” *Digit. Commun. Netw.*, vol. 7, no. 2, Jul. 2020.

[6] R. R. Pahlevi, P. Sukarno, and B. Erfianto, “Implementation of event-based dynamic authentication on MQTT protocol,” *J. Rekayasa Elektr.*, vol. 15, no. 2, Sep. 2019.

[7] P. Gope, O. Millwood, and N. Saxena, “A provably secure authentication scheme for RFID-enabled UAV applications,” *Comput. Commun.*, vol. 166, pp. 19–25, Jan. 2021.

[8] S. Kardaş, S. Çelik, M. Yıldız, and A. Levi, “PUF-enhanced offline RFID security and privacy,” *J. Netw. Comput. Appl.*, vol. 35, no. 6, pp. 2059–2067, Nov. 2012.

[9] A. Shamsoshoara, A. Korenda, F. Afghah, and S. Zeadally, “A survey on physical unclonable function (PUF)-based security solutions for Internet of Things,” *Comput. Netw.*, vol. 183, p. 107593, Dec. 2020.

[10] M. Tahavori and F. Moazami, “Lightweight and secure PUF-based authenticated key agreement scheme for smart grid,” *Peer--Peer Netw. Appl.*, vol. 13, no. 5, pp. 1616–1628, Sep. 2020.

[11] M. Barbareschi, A. De Benedictis, E. La Montagna, A. Mazzeo, and N. Mazzocca, “A PUF-based mutual authentication scheme for Cloud-Edges IoT systems,” *Future Gener. Comput. Syst.*, vol. 101, pp. 246–261, Dec. 2019.

[12] S. Khan, A. P. Shah, N. Gupta, S. S. Chouhan, J. G. Pandey, and S. K. Vishvakarma, “An ultra-low power, reconfigurable, aging resilient RO PUF for IoT applications,” *Microelectron. J.*, vol. 92, p. 104605, Oct. 2019.