International Journal of Innovation in Computer Science and Information Technology
Vol.1, NO.2 , P:92 - 113
Received: 08 Des 2018
Accepted: 10 Mar 2019

A survey of Cloud-based Radio-frequency Identification Authentication Protocols and Improvements to One of the Latest Proposed Protocols

Tahereh Atbaei Tabari
MSc student of Kish International Campus, University of Tehran
t.atbaei@ut.ac.ir
Hossein Samimi
Associate Professor, Institute of Communication and Information Technology
hossein-samimi@itrc.ac.ir
Ali Moeini
Professor of algorithms and computation, University of Tehran
moeini@ut.ac.ir

Abstract

The development of IoT and cloud data storage has enabled several objects to access the internet through radio-frequency identification (RFID) systems. Currently, the use of the cloud as a backend server is considered as a viable and cost-effective option. The connection between reader and server, on one hand, and the connection between tag and reader, on the other hand, need to be secure and private. The wired connection between tag and reader is now a wireless cloud-based system and calls for stricter security requirements. Not only the connection is insecure, but the cloud-provider which has full access to the tag and reader's confidential information is not necessarily trusted. In this paper, after analyzing previously proposed authentication protocols in RFID systems, a protocol was chosen to be scrutinized in this research. Owing to the features provided by the Scyther tool, the proposed protocol did not have the required privacy and security properties. By reducing the computational overhead between tag and tag reader, and increasing the protocol's resistance to desynchronized and reply attacks, in the present study, we attempted to further improve the protocol's shortcomings in privacy and security.

Keywords: cloud computing; RFID; security protocol; authentication; Scyther tool

Introduction

Radio-frequency identification (RFID) system is one of the key technologies used in the Internet of things (IoT). This technology has capabilities in automatic identification, localization, and access control of the objects

[1]. Today, RFID is used in applications such as payment, access control, ticketing, and e-passport that require a strong mechanism to maintain security and privacy [2]. An RFID system is comprised of tag, tag reader, and backend server [3]. A tag reader emits radio-frequency waves to perform tag activation, and deliver and receive data without any physical contact [4]. RFID authentication is the first step in ensuring security and privacy. Identification of a tag without authentication can lead to serious problems [5]. In most cases, a communicating channel between tag reader and server is assumed [1], which limits the mobility of the reader and makes the database of back-end server a bottleneck, because RFID systems identify objects whose number is usually increasing [5].

Today, with the development of cloud computing in RFID systems, presenting authentication protocols capable of employing all the advantages of cloud services has received the attention of researchers [6]. Wireless communication between the tag reader and the cloud server presupposes an insecure channel. Moreover, cloud providers are also considered as unreliable individuals. Thus, an authentication protocol in cloud-based RFID systems should be able to store tag information anonymously in the server database, while transferring data securely between reader and cloud server. This entails a great deal of research into RFID systems, cloud servers, and security protocols.

Literature and background

To date, many solutions have been proposed to address problems and prevent attacks happened due to insecure channel and unreliable cloud service provider in cloud-based RFID systems. However, while paying attention to how the system works, few studies have considered ensuring security and privacy. A summary of the most recent works in this regard is presented in Table 1.

Table 1. a review of the protocols presented in the past

developers	Year	Method and idea	Strengths	Weaknesses
Xie W, Xie L, Zhang C, Zhang Q, Tang C [5]	2013	Application of virtual proxy network (VPN)	Attention to insecure channel	Very costly and vulnerable to tag readers and tracking impersonation attacks [4]

		and encryption of hash tables		
Chen SM, Wu ME, Sun HM, Wang KH [7]	2014	Application of tree data mechanism	Reducing search time complexity to O(log N)	Obtaining the data of all tags if one knows the whole data of a limited number of them [6]
Lin IC, Hsu HH, Cheng CY [8]	2015	Encryption by hash functions and exclusive OR (XOR)	Developed the scheme to be applied to supply chain	Vulnerable to attacks for reusing old keys and denial of service [6]
Abughazalah S, Markantonakis K, Mayes K [4]	2015	Use of symmetrical encryption and hash functions	Addressed Xie et al., 2013 protocol problems	Lack of considering insecure channel
Dong Q, Tong J, Chen Y [10]	2015	Use of location and private cloud service	Prevented the location of tag reader from being leaked	A need for making vast changes in infrastructures [1]
Xiao H, Alshehri AA, Christianson B [6]	2017	Use of symmetric encryption and hash functions and encryption of hash tables	Addressed Abughazalah et al., 2014 protocol problems	Leakage of the reader's location [1]
Dong Q, Chen M, Li L, Fan K [1]	2018	Use of MIPv61 protocol	Prevented tag reader's location from being disclosed	Vulnerable to old key and desynchronization attacks

There follows a brief account of the studies presented in Table 1.

[1] Mobile Internet Protocol version

- Chi et al. (2013) came up with a cloud-based authentication protocol for RFID systems for the first time. One of the main features of this protocol is the use of a VPN to establish a secure connection between the reader and the cloud server. But this scheme, apart from high costs of VPN implementation, is susceptible to tracking and reader impersonation attacks [4].
- Chen and colleagues (2014) proposed a protocol to reduce search time complexity in a database from O (N) to O (logN) and be resistant to desynchronization and tracking attacks. Because of using tree data mechanism in this scheme, the attacker can retrieve the information of all tags by acquiring the information of a number of tags [6].
- Lin et al. (2015) offered a third-part authentication protocol that is useful for supply chains using cloud-based RFID systems. However, the protocols are vulnerable to old key reuse and DoS attacks due to the simple structure of a key generation algorithm [6].
- Abughazalah et al. (2015) identified weaknesses in Chi et al.'s protocol and came up with an updated protocol. The assumption of the scheme is a secure communication channel between the tag reader and could server that is unusable for mobile readers. Moreover, the tag's data are explicitly stored in the cloud [6].
- Dong et al. (2015) proposed a protocol based on location privacy cloud service. However, its implementation called for some changes in the overall infrastructure that are not easy in practice [1].
- Xiao et al. (2017) suggested a protocol for insecure channels that can protect data during transmission and ensure data security without a third party. But the protocol does not prevent the reader's location from leaking [1].
- Dong et al. (2018) offered a scheme that can prevent the leakage of the reader's location by employing the MIPv6 protocol. They posited that the protocol is resistant to tag tracking, replay, desynchronization, and DoS attacks and mutual authentication is properly established between the tag reader and the tag. Moreover, they verified their claim using the AVISPA tool.

The purpose of this research is to validate the security of an authentication protocol proposed for the above cloud-based RFID systems; i.e., Authentication Protocol with Location Privacy (APWLP). In the following, this protocol is analyzed, the encountered problems and damages are

presented, and an attempt is made to improve it using the security tool Scyther.

The notations used in this article are interpreted in Table 2.

Authentication in APWLP protocol

Table 2. notations used in APWLP authentication protocol

notations	Description
T	Tag
R	Mobile reader
Cloud	Cloud server
id_T	Identity of the tag
id_R	Identity of the reader
key	A secret key of the tag
kRC	A key shared by the reader and the cloud server
k	Symmetric encryption and decryption key for encrypting the data stored in the cloud
r_i	A random number
MAC	Message authentication code
h()	Hash function
E()	Symmetric encryption algorithm
||	The concatenation of two strings

Initialization phase

1. The tags store their own identity (id_T) and secret value key

2. The cloud server stores

- $(h(id_R), kRC)$
- $\{h(key \| id_T \| h(id_R)), E_k(key \| id_T)\}$
- $\{h(key_{old} \| id_T \| h(id_R)), E_k(key_{old} \| id_T)\}$, and sets $h(key \| id_T \| h(id_R)) =$

$h(key_{old} \| id_T \| h(id_R))$ and $E_k(key \| id_T) = E_k(key_{old} \| id_T)$.

3. The reader stores its identity id_R, key, and pre-computes $h(id_R)$ and shares authentication key **kRC** with the cloud server.

Authentication phase

The protocol is comprised of three factors. In the authentication phase, the cloud server is only responsible for storing and searching the RFID data, and the authentication is between tag and reader.

Step 3: $R \rightarrow C$: $r_1 \| h(id_R) \| X \| MAC_{kRC}\{r_1 \| h(id_R) \| X\}$
The reader stores the messages $M_1 = h(r_1 \| key \| id_T)$ and r_2 sent by the tag. Next, it sends $r_1 \| h(id_R) \| X \| MAC_{kRC}\{r_1 \| h(id_R) \| X\}$ to the cloud server. $MAC_{kRC}\{r_1 \| h(id_R) \| X\}$ denotes the MAC.

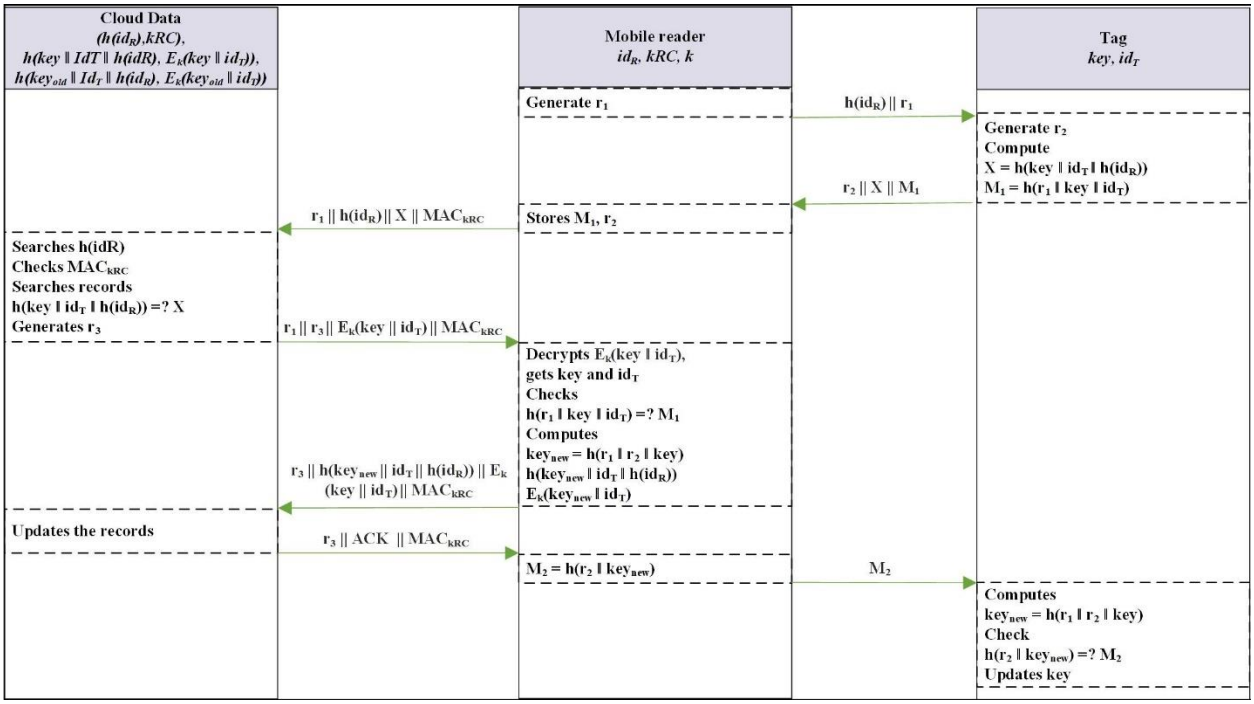
Fig. 1. APWLP authentication protocol

As shown in Fig. 4, the cloud-based RFID mutual authentication protocol is as follows;

Step 1: $R \rightarrow T$: $h(id_R) \| r_1$
The mobile reader generates a random number r_1 and sends $h(id_R) \| r_1$ to the tag.

Step 2: $T \rightarrow R$: $r_2 \| h(key \| id_T \| h(id_R)) \| h(r_1 \| key \| id_T)$
After receiving $h(id_R) \| r_1$, the tag generates a random number r_2, uses the hash to compute $X = h(key \| id_T \| h(id_R))$, $M_1 = h(r_1 \| key \| id_T)$, and sends $r_2 \| X \| M_1$ to the reader as a response.

Step 4: $C \rightarrow R$: $r_1 \| r_3 \| E_k(key \| id_T) \| MAC_{KRC}\{r_1 \| r_3 \| E_k(key \| id_T)\}$



The cloud server searches $h(id_R)$; if finds it, the cloud server can find the corresponding kRC, then uses kRC to verify $MAC_{kRC}\{r_1 \| h(id_R) \| X\}$ to determine whether the information from the reader is tampered or not. If the verification passes, the cloud server searches X in its own records. The searching results can be one of the following cases:

- There is no matching hash value, the authentication is failed, and then the protocol stops.
- If the cloud server finds $h(key \| id_T \| h(id_R))$ that is equal to X, it indicates that both the cloud server and the tag made the update normally in the last authentication. The cloud server generates a random number r_3 and sends to $r_1 \| r_3 \| E_k(key \| id_T) \| MAC_{kRC}\{r_1 \| r_3 \| E_k(key \| id_T)\}$ to the reader.
- If the cloud server finds $h(key_{old} \| id_T \| h(id_R))$, which is equal to X, it indicates that the cloud server performed the update normally, while the tag did not renew its key in the last authentication. The cloud server generates a random number r_3 and sends $r_1 \| r_3 \| E_k(key_{old} \| id_T) \| MAC_{kRC}\{r_1 \| r_3 \| E_k(key_{old} \| id_T)\}$ to the reader.

Step 5: $R \rightarrow C$: $r_3 \| h(key_{new} \| id_T \| h(id_R)) \| E_k(key_{new} \| id_T) \| MAC_{kRC}\{r_3 \| h(key_{new} \| id_T \| h(id_R)) \| E_k(key_{new} \| id_T)\}$

The reader uses k to decrypt $E_k(key \| id_T)$ to get key and id_T. Then, it checks whether $h(r_1 \| key \| id_T)$ is equal to M1. If it succeeds, the tag is authenticated by the reader. The reader computes $key_{new} = h(r_1 \| r_2 \| key, h(key_{new} \| id_T \| h(id_R), E_k(key_{new} \| id_T)$. Then, it sends $r_3 \| h(key_{new} \| id_T \| h(id_R)) \| E_k(key_{new} \| id_T) \| MAC_{kRC}\{r_3 \| h(key_{new} \| id_T \| h(id_R)) \| E_k(key_{new} \| id_T)\}$ to the cloud server to remind the cloud server to carry out the update.

Step 6: $C \rightarrow R$: $r_3 \| ACK \| MAC_{kRC}\{r_3 \| ACK\}$

The cloud server checks $MAC_{kRC}\{r_3 \| h(key_{new} \| id_T \| h(id_R) \| E_k(key_{new} \| id_T)\}$ and updates the corresponding records, sets:

- $h(key_{old} \| id_T \| h(id_R)) = h(key \| id_T \| h(id_R))$
- $E_k(key_{old} \| id_T) = E_k(key \| id_T)$
- $h(key \| id_T \| h(id_R)) = h(key_{new} \| id_T \| h(id_R))$
- $E_k(key \| id_T) = E_k(key_{new} \| id_T)$

Next, it sends $r_3 \| ACK \| MAC_{kRC}\{r_3 \| ACK\}$ to the reader.

Step 7: $R \rightarrow T: M_2$

When the reader receives $r_3 \| ACK \| MAC_{kRC}\{r_3 \| ACK\}$, it computes $M_2 = h(r_2 \| key_{new})$ and sends M_2 to the tag.

Step 8: The tag computes $key_{new} = h(r_1 \| r_2 \| key)$ and verifies whether $h(r_2 \| key_{new})$ is equal to M_2 or not; if it succeeds, the reader is authenticated by the tag. Then, the tag updates key = key_{new}.

Security analysis of APWLP protocol

In this section, we analyze the security features in the APWLP protocol.

Mutual authentication

With the analysis of APWLP protocol in the third phase ($E_k(key \| id_T)$, we realize that the major part of a security protocol is dependent on the reader's key because authentication messages M1 and M2 will lose their credits in case the key is leaked. As the reader's key is constant, its security decreases in all sessions.

Reader tracking

As mentioned in the analysis of APWLP protocol, the identity of the reader is sent in plaintext $h(id_R)$ and it will be a constant in all subsequent sessions. Therefore, it can be concluded that the protocol does not provide forward security and the reader may be subjected to a tracking attack.

Desynchronization attack

Similarly, the values not corresponding to the tag are stored in the cloud database in this protocol. To prevent a desynchronization attack, both the new and old values of the tag key are kept, while the mechanism just makes one future session resistant to desynchronization attack and there remains the risk of vulnerability in other sessions. The value of the tag is updated before sending the new key to the tag, which raises the possibility of the desynchronization between the tag and cloud.

Replay attack

If the attacker attains the reader's symmetric key, the possibility of the attack by the old key remains because it sends the message $X = h(key \| id_T \| h(id_R))$ of the tag's key without any random number or timestamp.

Research method

In this section, we introduce Scyther by which we analyzed the APWLP protocol and the optimized protocol in the following sections.

One of the means of analyzing security protocols is to find the violations of security claims made at the time of a protocol description. This method has been applied by some tools including Scyther. It deals with violations of security claims by searching for attack patterns in different cases of the protocol (i.e., Trace patterns). A protocol can be attacked in different ways by an attacker, which is referred to as trace in the logic of Scyther. There is another concept known as Trace pattern in the logic of Scyther; it is a symbolic and disciplined demonstration of a set of traces that consist of existing definitions and events in a protocol.

According to the described rules, Scyther attempts to find threat patterns against security claims in a protocol definition by making search and deduction about a trace pattern. A trace pattern occurs only if Scyther can reach it from an initial state of the protocol's symbolic definition. Search in trace patterns can give rise to new trace patterns distinct from preceding states, or may end up with duplicate trace patterns. A security claim is made when there is no violation in any of the trace patterns. During this process, the execution of this algorithm may sometimes cease to stop. Hence, in such cases, Scyther envisages some limitations on the number of executions. In this way, if a security claim is violated, the algorithm can go ahead to find a more optimized violation or terminate at that point and denote the violation [11].

Security requirements define the goal of a security protocol. The main idea of claims is their locality. According to this concept, based on the messages they receive, agents have a local view on the state of a system. A protocol should guarantee whether an agent can be assured of some characteristics of the general situation of the system according to this local view. For instance, certain conditions cannot be detected by an attacker or a special agent is active. In the following, we describe some claims of authentication and security.

Secrecy: This claim states that certain information will not be leaked to an attacker even if it is exchanged on an insecure network. Secrecy is a protocol that holds only when for network components we have; in each run. Roles are mapped only to reliable agents. Also, there are certain conditions for secrecy that should not be understood by the attacker. The claim of secrecy can assign roles for any condition including variables.

Authentication: There are many cases of authentication. In the basic case, authentication is a simple statement about the

existence of a communication entity. In describing a protocol, it is stated that at least two agents are communicating; however, we cannot guarantee that there is certainly one communication entity in each run when the network is controlled by the attacker. For example, the initiator cannot be sure who sent the message "hello" it received. This message might have been sent by an attacker or any other agent. The minimum requirement for authentication is that when agent A performs a role it can make sure that there is at least one communication entity in the network. In most cases, stronger authentication is required; for example, when a communication entity is aware of the matter that it is communicating with A and knows that the communication entity performs a given role. In addition, maybe we need to make sure that the message was sent according to the description of the protocol. In the following, we describe some cases of authentication;

- Aliveness: This claim lays down that the communication entity performed some events, which means that the entity is alive, so to speak. In Scyther, four cases of aliveness are described: "weak aliveness", "weak aliveness in the correct role", "recent aliveness", and "recent aliveness in the correct role". The description of weak aliveness determines that if an agent performs a claim event and the communication entity in question is reliable, so this entity performs an event. In most cases, knowing that a communication entity is in the state of weak aliveness per se is not enough, as at least we need to know it is performing a role as expected with respect to the description of the protocol. "Weak aliveness in the correct role" describes that the communication entity is alive but it does not reveal whether a claim event happened before, after or during execution. "Recent aliveness" shows that a communication entity performs an event when performing a role in which the claim event occurs.

- Synchronization: aliveness requires exciting certain events by a communication entity without putting restrictions on the content of exchanged messages. One of the requirements of stronger authentication is fulfilled through synchronization. In this case, all messages received should have been sent by a communication entity. Indeed, the messages need to be

received by a communication entity. In this case, the exchange of a real message should have occurred in accordance with the description of a protocol.

- Non-injective synchronization: This characteristic states that any event that we want to occur in the description of a protocol occurs in each component of the protocol. It is important to know that synchronization does not simply consider the content and order of messages. If an attacker sends messages without making any change in a receiver, this is not held to be a synchronization attack.

- Injective synchronization: Non-injective synchronization ensures that the protocol will be precisely executed and this happens when there is no attacker. Although agents may have a couple of executions probably concerning similar agents, the attacker can imply an unexpected behavior by changing the messages of a session versus another session. Particularly, protocols may be still susceptible to replay attacks in the event of non-injective synchronization. In a replay attack, the attacker responds to the message received so that reliable agents are fooled. In this way, they think that the protocol run was successfully completed. If there is no attacker whatsoever, the behavior ceases to exist. For a protocol with two agents (i.e., initiators with claim event and respondent), there must be an injective scheme of initiator's runs together with a claim event for corresponding respondent's run. In other words, the initiator's two different versions of claim should correspond to the respondent's different runs.

- Agreement: Synchronization ensures that the protocol is run as described for it even in the presence of an attacker. Another group of authentication characteristics focuses on the agreement on the data exchanged between agents. The idea behind these agreements maintains that after a protocol runs, parties agree on the values of variables. According to the description of the protocol, an agreement is defined by requesting that the contents of the received messages are related to the

message sent. As a result, after running a protocol, the values of variables exactly match the protocol description.

The description of an agreement is much like that of synchronization, except that the latter entails the order of the relation; i.e., sending an event should precede the receiving event.

The agreement states that there are runs for other roles of the protocol for all claim events presented in every single part of a security protocol so that all communication events preceding the claim event should have occurred before a claim. The injective agreement is defined in the same way as injective synchronization, which is obtained from a non-injective synchronization [12].

Comparison with other tools

Scyther tool was compared with the three tools: Avispa (with four backup tools CL-Atse, OFMC, Sat-MC, and TA4SP), ProVerif, and Casper/FDR. The results are based on the key-sharing protocols Needham-Schroeder, EKE, and TLS. In the following, a brief account of the comparison is presented.

Comparing the protocol analysis tools' performance showed that Proverif is the fastest tool for analyzing the protocols. With a slight difference, Scyther stands in second place. However, it has the advantage of not using approximations. CL-Atse and OMFC were so close with concrete sessions, and are the most efficient of the Avispa tools. The

Sat-MC is placed in the next rank. Casper/FDR has an exponential behavior and is slower than OMFC and CL-Atse but faster than Sat-MC for a small number of runs.

For a higher number of runs of simple protocols, it seems that Sat-MC can outperform the other two tools. In some cases, TA4SP can complement other Avispa tools, but it generally is significantly slower than tools that can handle unbounded verification (Scyther and proverif) and has the drawback of not being able to show attack traces (Cremers et al., 2009).

In Fig. 2., we show the results of executing this protocol by using the Scyther tool.

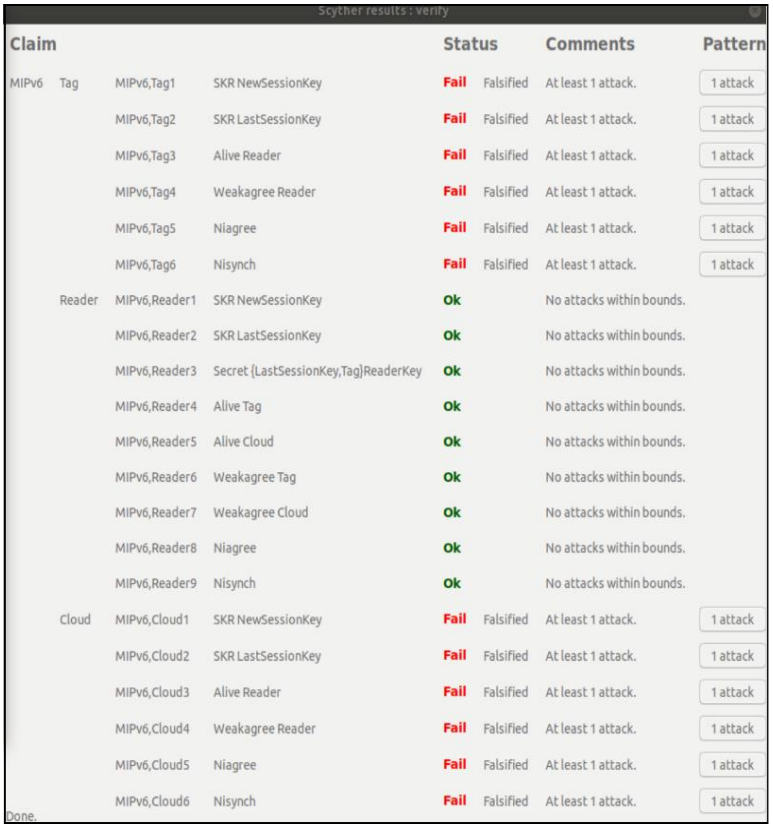


Fig. 2. Attacks to APWLP protocol by using Scyther tool

As shown in Fig. 2, the characteristics of authentication are not provided for the tag, so the authenticity of the reader is not verified for it. In addition, given that Scyther hampers the checking of the states of illogical leakage of a long-term key, the attacker, after all, gets access to the reader's and cloud's shared long-term key after three consecutive sessions. The graph of its attack is shown in Figs. 3, 4, and 5. It is noteworthy that the leakage violates Niagree, Aliveness, and Nisynch's claims in the cloud.

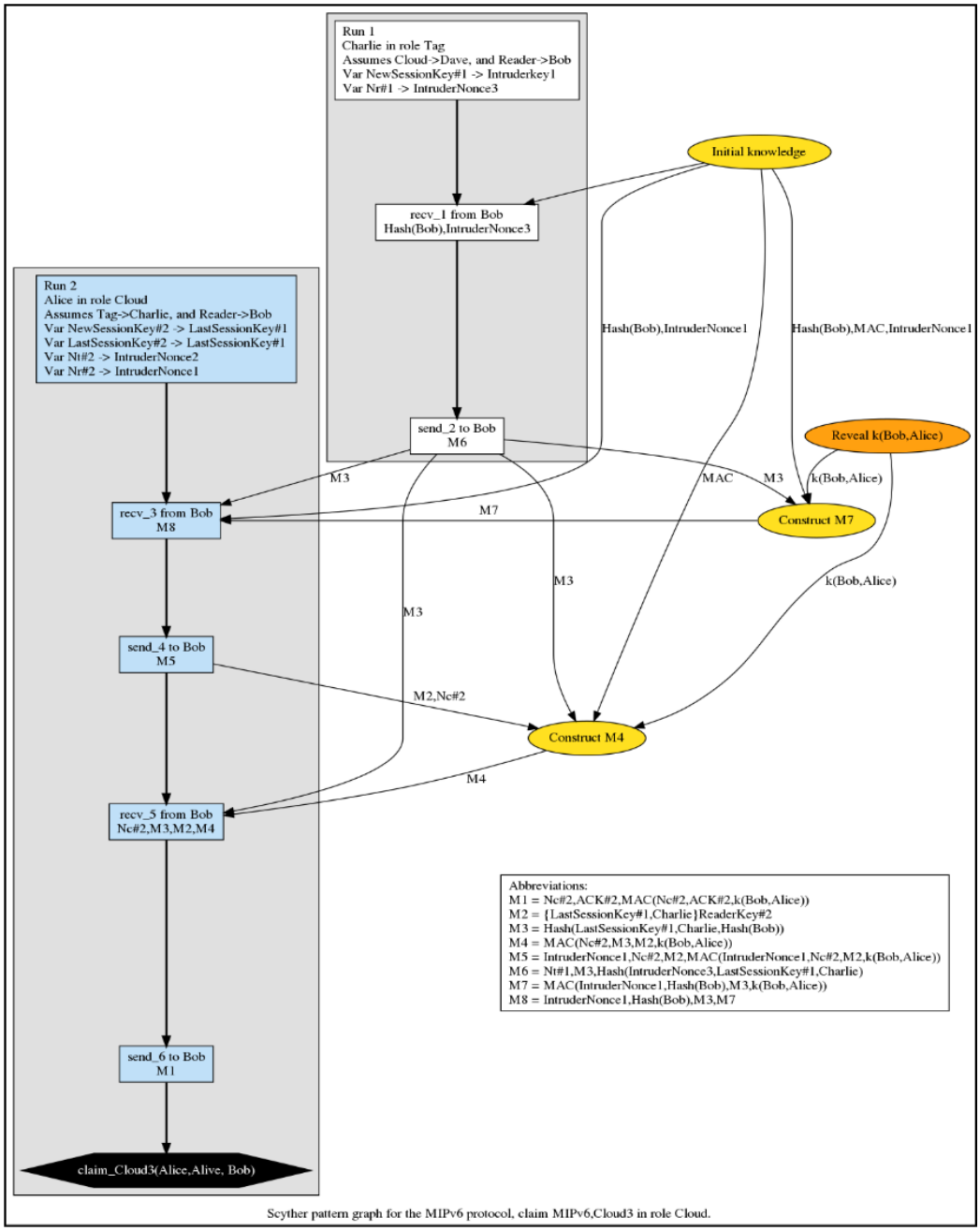


Fig. 3. Violation of aliveness claim in APWLP protocol

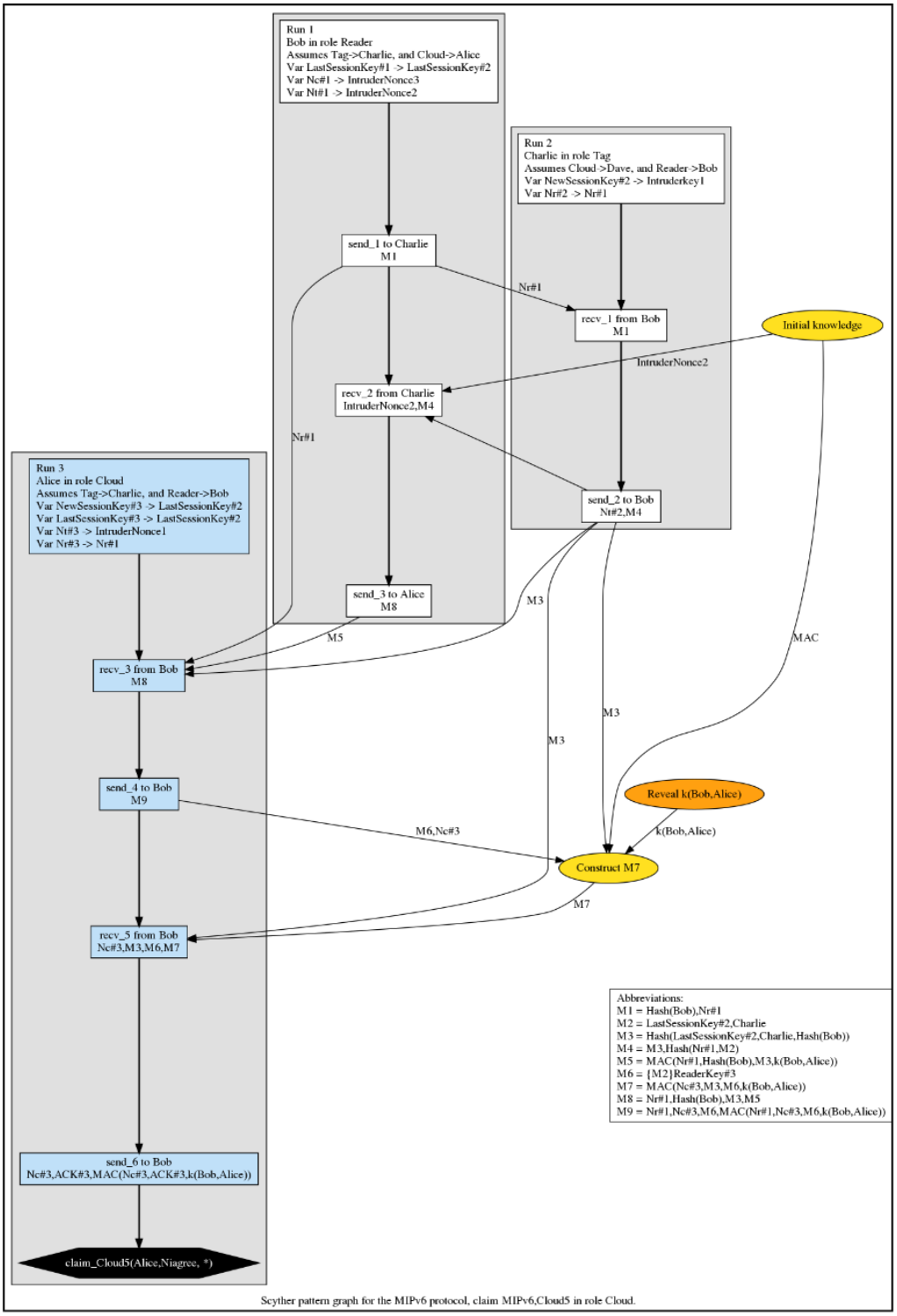


Fig. 4. Violation of agreement claim in the APWLP protocol

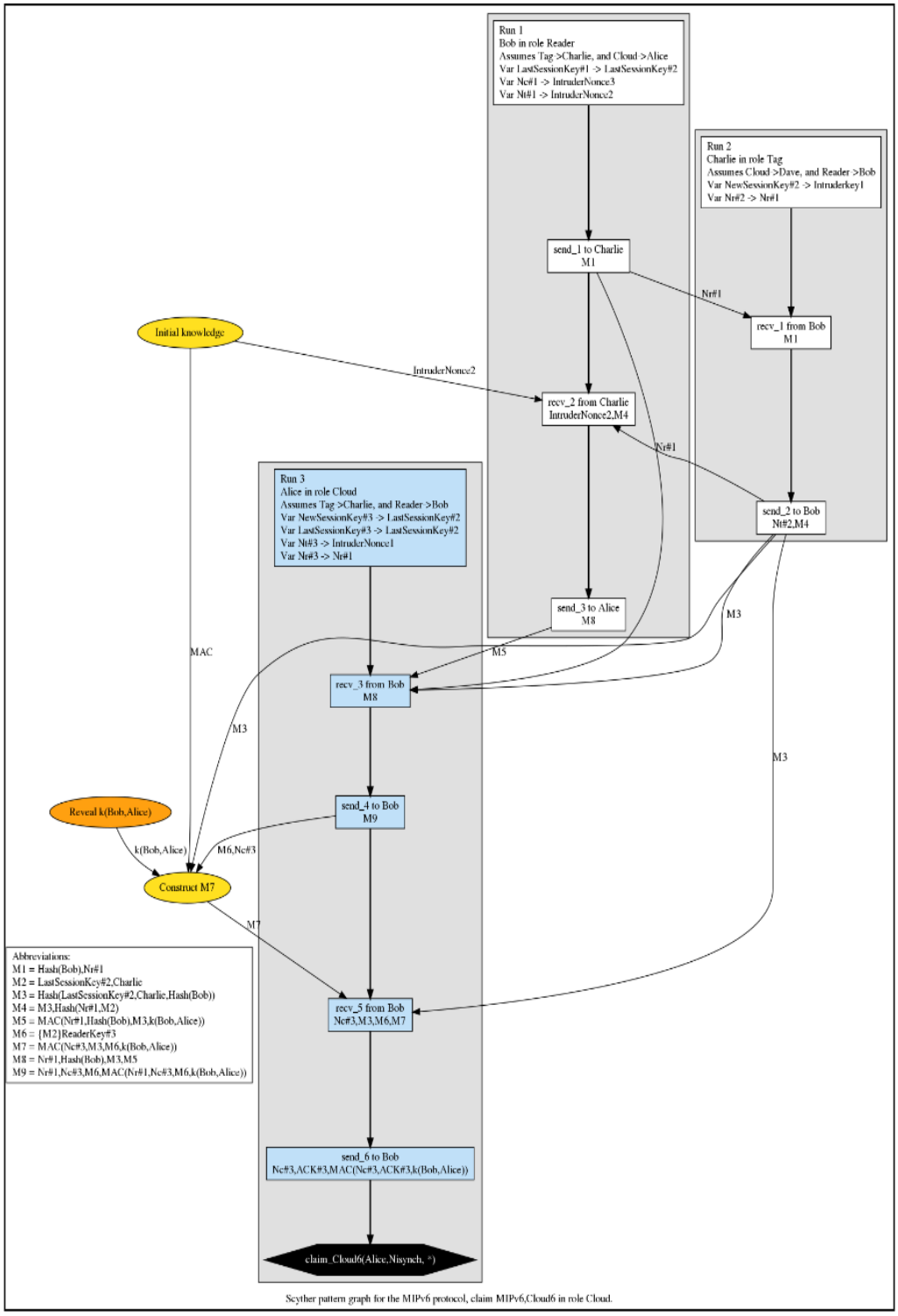


Fig. 5. Violation of synchronization claim in the APWLP protocol

Optimized protocol

In this section, an attempt was made to reduce the computational overhead for the tag using the rotation function instead of the hash

function. This can be observed in reducing the time of protocol run by means of a Scyther tool. Moreover, when the reader's key is leaked, the attacker cannot get access to the identity of the tag. In this protocol, to establish the feature of message freshness and data indexes stored in the cloud, we used timestamp instead of a random number. Table 3 presents the notations used in the optimized protocol.

Table 3. Notations used in the optimized protocol

Notations	**description**
T	Tag
R	Mobile reader
C	Cloud server
id_T	Identity of the tag
id_R	Identity of the reader
key	The secret key of a tag
kRC	The shared key between reader and cloud server
k	The symmetric key for encrypting and decrypting data stored in the cloud
Tr	Timestamp generated by the reader
Tt	Timestamps stored in the tag
Rot(A,B)	Rotation to the left A by the length of B bites
$\oplus$	Exclusive Or
$\|\|$	The concatenation of two strings

Initialization phase

The tag stores $E_k(id_T)$ and Tt.

The reader calculates and stores $Rot(E_k(id_R),(E_k(id_R) \oplus id_R)$ and KRC.

The cloud database stores data as shown in Tables 4 and 5.

Table 4. The way shared key is stored between reader and cloud in the cloud

Index	Data
$Rot(E_k(id_R),(E_k(id_R) \oplus id_R))$	kRC
⋮	⋮

Table 5. How the identity of the tag is stored in the cloud

Index	Data
T	$E_k(id_T)$
⋮	⋮

Authentication phase

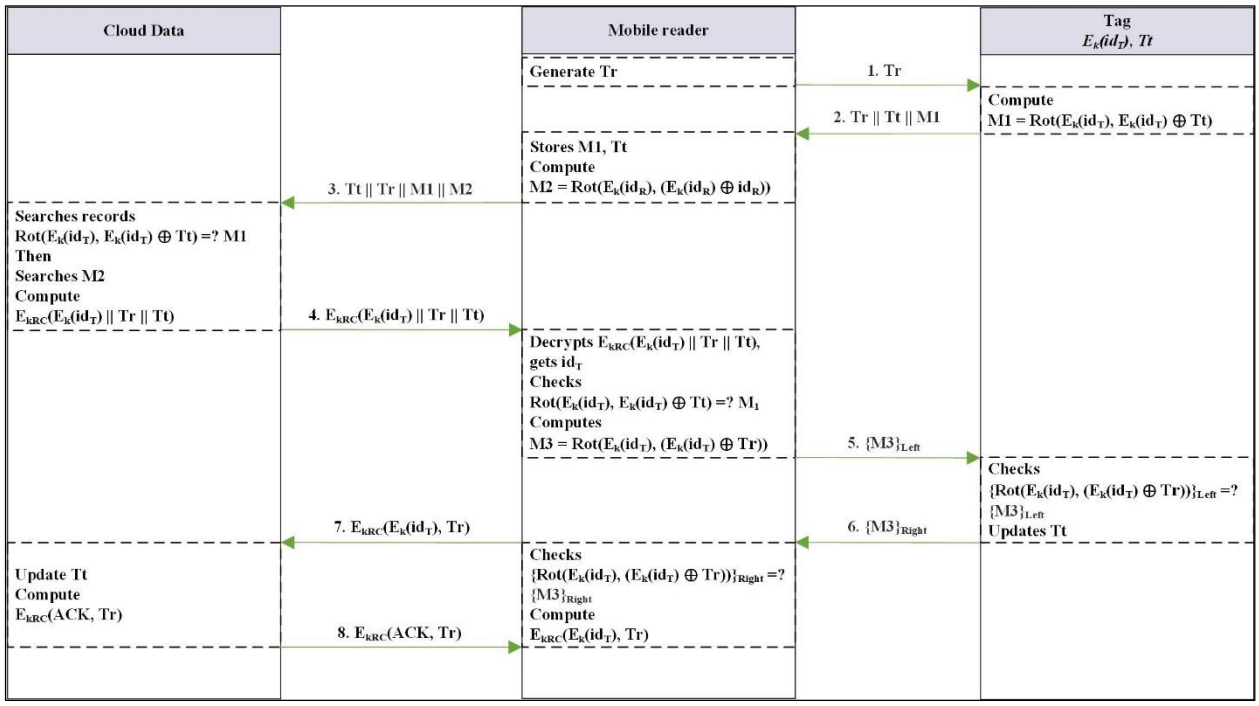


Fig. 6. Optimized authentication protocol

The optimized protocol shown in Fig. 6 is as follows;

Step 1: R → T: Tr

The reader generates a timestamp, i.e., Tr, and sends it to the tag.

Step 2: $T \rightarrow R$: $Tr \parallel Tt \parallel Rot(E_k(id_T), E_k(id_T) \oplus Tr)$

Having received Tr, the tag computes the message $Rot(E_k(id_T) \oplus Tr$ and sends it as a response to the reader along with the timestamp of the preceding session Tt.

Step 3: $R \rightarrow C$: $Tr \parallel Tt \parallel Rot(E_k(id_T), (E_k(id_T) \oplus Tr)) \parallel Rot(E_k(id_R), (E_k(id_R) \oplus id_R))$

The reader sends the message M1 = $Rot(E_k(id_T), (E_k(id_T) \oplus Tr))$ sent from the tag to find the identity of the tag and the message M2 = $Rot(E_k(id_R), (E_k(id_R) \oplus id_R))$ to present itself to the cloud and use their shared key to the cloud server.

Step 4: $C \rightarrow R$: $E_{kRC}(E_k(id_T) \parallel Tr \parallel Tt)$

Cloud server searches M2. If finds it, the cloud server can find corresponding KRC and is certain that no other agent is able to compute it. If the verification operation succeeds, the cloud server searches Tt in its records. The result of the search can be either of the two cases:

- There is no compatible timestamp, the authentication fails, and the protocol run is terminated.
- If the cloud server finds Tt, the cloud server and tag both successfully make the update in the last authentication and send $E_{kRC}(E_k(id_T)\|Tr\|Tt)$ to the reader.

Step 5: $R \rightarrow T$: $\{Rot(E_k(id_T), (E_k(id_T) \oplus Tt))\}_{Left}$

The reader uses KRC to decrypt $E_{kRC}(E_k(id_T)\|Tr\|Tt)$. Once Tr is seen, it makes sure whether it is fresh, and the identity of the cloud is authenticated for the reader because the cloud is aware of the shared key. Next, using K, it gets id_T, and its authenticity is confirmed for the reader after computing $Rot(E_k(id_T), (E_k(id_T) \oplus Tr))$ and comparing it with the message of M1 received from the tag. The reader send the left part of M3 = $Rot(E_k(id_T), (E_k(id_T) \oplus Tt))$ to the tag.

Step 6: $T \rightarrow R$: $\{Rot(E_k(id_T), (E_k(id_T) \oplus Tt))\}_{Right}$

Having received the left part of M3, the tag computes it. If it is equal to the message received from the reader, the authenticity of the reader will be confirmed for the tag, because it is only the reader that is allowed to retrieve the identity of the tag from the cloud server. Next, to confirm Tr, it sends the right part of M3 to the reader and replaces Tr with its own Tt.

Step 7: $R \rightarrow C$: $E_{kRC}(E_k(id_T), Tr)$

Once the right part of M3 is received, the reader will be aware of the tag's update and

computes $E_{kRC}(E_k(id_T)$, Tr and sends it to the server so that it matches the tag by updating Tt.

Step 8: C → T: E_{kRC}(ACK, Tr)

Once the message $E_{kRC}(E_k(id_T)$, Tr) is received, the cloud server replaces the corresponding Tt with Tr. Then, it sends the verification message $E_k(id_T)$ to the reader.

Using the Scyther tool, we analyzed the security of the proposed protocol. As shown in Fig. 7, none of the security claims were violated and no attack happened to it.

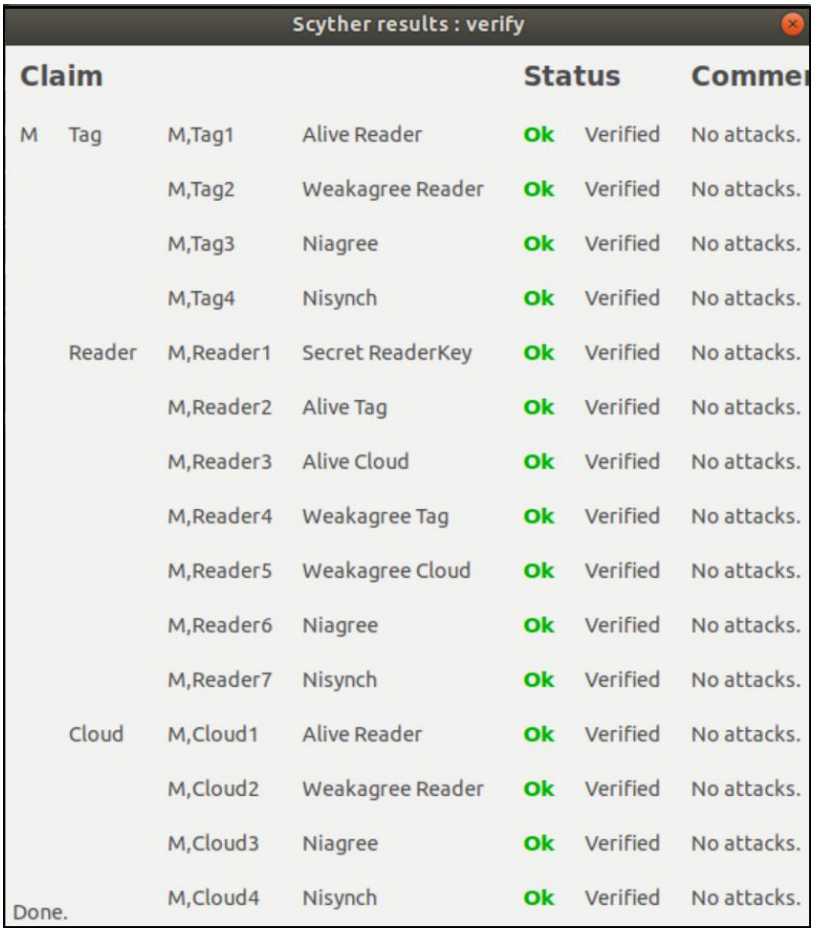


Fig. 7. Result of the analysis of the proposed protocol by means of Scyther tool

Conclusion

In this paper, we analyzed the evolution of authentication protocols in could-based RFID systems. Then, we thoroughly analyzed one of the protocols recently introduced in this context (i.e., APWLP). Using the Scyther tool, we demonstrated that the protocol does not have the essential

security characteristics. We attempted to reduce the tag's and reader's computation overhead in an RFID system by employing lighter functions, making it resistant to desynchronization attacks and replay. We demonstrated it also by means of the Scyther tool. One of the reasons for choosing the APWLP protocol is, in addition to being up to date, the use of the Avispa security validation tool by its developers. In the present study, we demonstrate that this tool can detect all attacks by drawing on the material presented and validating our own protocol.

There are plenty of powerful tools in this context. For future studies, we will try to expose the weaknesses and strengths of other tools by investigating them in detail. In this way, we can obtain a tool with the minimum weaknesses to validate security protocols.

References

[1] Dong, Q., Chen, M., Li, L., & Fan, K. (2018). Cloud-based Radio-frequency identification authentication protocol with location privacy protection. International Journal of Distributed Sensor Networks, 14(1), 1550147718754969.

[2] Kardas, S., Çelik, S., Bingöl, M. A., & Levi, A. (2013, December). A new security and privacy framework for RFID in cloud computing. In 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (Vol. 1, pp. 171-176). IEEE.

[3] Fan, K., Luo, Q., Li, H., & Yang, Y. (2017, June). Cloud-based lightweight RFID mutual authentication protocol. In 2017 IEEE Second International Conference on Data Science in Cyberspace (DSC) (pp. 333-338). IEEE.

[4] Abughazalah, S., Markantonakis, K., & Mayes, K. (2014). Secure improved cloud-based RFID authentication protocol. In Data Privacy Management, Autonomous Spontaneous Security, and Security Assurance (pp. 147-164). Springer, Cham.

[5] Xie, W., Xie, L., Zhang, C., Zhang, Q., & Tang, C. (2013, April). Cloud-based RFID authentication. In 2013 IEEE International Conference on RFID (RFID) (pp. 168-175). IEEE.

[6] Xiao, H., Alshehri, A. A., & Christianson, B. (2016, August). A cloud-based RFID authentication protocol with insecure communication channels. In 2016 IEEE Trustcom/BigDataSE/ISPA (pp. 332-339). IEEE.

[7] Chen, S. M., Wu, M. E., Sun, H. M., & Wang, K. H. (2014). CRFID: An RFID system with a cloud database as a back-

end server. Future Generation Computer Systems, 30, 155-161.

[8] Lin, I. C., Hsu, H. H., & Cheng, C. Y. (2015). A cloud-based authentication protocol for RFID supply chain systems. Journal of Network and Systems Management, 23(4), 978-997.

[9] Rong, C., Zhao, G., Yan, L., Cayirci, E., & Cheng, H. (2013). RFID security. In Computer and Information Security Handbook (pp. 345-361). Morgan Kaufmann.

[10] Dong, Q., Tong, J., & Chen, Y. (2015). Cloud-based RFID mutual authentication protocol without leaking location privacy to the cloud. International Journal of Distributed Sensor Networks, 11(10), 937198.

[11] Cremers, C. J. (2008, July). The Scyther Tool: Verification, falsification, and analysis of security protocols. In International Conference on Computer Aided Verification (pp. 414-418). Springer, Berlin, Heidelberg.

[12] Cremers, C. J. F. (2006). Scyther: Semantics and verification of security protocols. Eindhoven, Netherlands: Eindhoven University of Technology.

[13] Cremers, C. J., Lafourcade, P., & Nadeau, P. (2009). Comparing state spaces in automatic security protocol analysis. In Formal to Practical Security (pp. 70-94). Springer, Berlin, Heidelberg.