# Provably Lightweight RFID Mutual Authentication Protocol

Rima Hussin Embrak Alakrut, Azman Samsudin and Alfin Syafalni
*School of Computer Sciences, Universiti Sains Malaysia*
*11800 USM, Penang, Malaysia*
*rima@student.usm.my, azman@cs.usm.my, as10_com083@student.usm.my*

***Abstract***

*Simplest-Lightweight Authentication Protocol (SLAP) is one of the recent mutual authentication protocols for lightweight RFID environment. However, server impersonation attack can be launched on the aforementioned protocol. The main goal of this paper is to comprehensively verify the security weakness of SLAP. This paper also propose an alternative mutual authentication protocol which proven to be secure and convenient for lightweight RFID environment. The proposed protocol is verified by a security protocol verifier, AVISPA, and benchmarked against the original SLAP. The results show that the proposed protocol has the comparable capability with SLAP in preventing known attacks and simultaneously removes the known security weakness on SLAP.*

***Keywords:*** *Authentication, Lightweight RFID, Security Analysis, Applied Cryptography.*

## 1. Introduction

Radio-frequency identification (RFID) is a wireless communication used to transfer data using radio wave. A typical RFID system comprises of one or more readers that has the capacity to communicate within narrow ranges with many tags [1], [2] as shown in Figure 1. An RFID tag can be attached on various entities such as product, person, or animal for the purpose of identification. The main advantage of using RFID is that some tags can be read from several meters away and even beyond the line of sight of the reader.

RFID has evolved in numerous new forms and applications. For instance, the battery-powered active RFID devices have been used for automatic toll collection on motorways since the early 80s. In addition, RFID tags are now can be packed into a very cheap and small form suitable for application on single-product packages due to the technological advances in miniaturization. Hence, most of the current successful hypermarkets are using the RFID system and tags to enable per item tracking of goods from the manufacturer to the end-user [3].

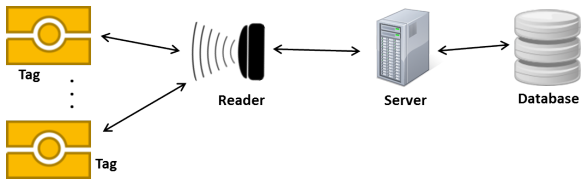



**Figure 1: RFID system**

Nevertheless, like any other communication systems, RFID systems are vulnerable to different kind of attacks such as eavesdropping, replay attack, man-in-the-middle attack, and denial of service (DoS) attack. Besides, the data obtained from a valid tag can be easily used to impersonate the tag. As a consequence, the reader will not able to confirm whether it communicates with a valid or fake tag [4]. This type of attack is known as impersonation attack which is preventable by engaging a secure authentication mechanism.

In 2004, Ohkubo et al. [5] proposed an RFID protocol based on hash-chains to prevent eavesdropping. However, this protocol is vulnerable to replay attacks which can be mounted after unsuccessful authentication. Henrici et al. [6] proposed a simple scheme relying on one way hash-functions based on Ohkubo et al. [5] scheme to prevent the eavesdropping, message interception, spoofing, and replay attacks. Unfortunately, this protocol is vulnerable to the tracing attack.

In 2005, Dimitriou [7] presented a protocol that provides secure and authenticated tag-reader transactions to prevent against standard attacks such as impersonation, replay, and cloning. However, this protocol is vulnerable to the tracing attack since the tag ID stays constant during the identification session of the tag. Alternatively, Luo et al. [8] proposed another RFID scheme based on Ohkubo et al. [5]. The strength of the Lous protocol is that encrypted data traffic is supported between the parties without the use of cryptographic ciphering algorithms yet, Lous protocol is vulnerable against to replay attack.

In 2008, Gódor et al. [9] proposed a lightweight RFID mutual authentication protocol known as Simplest-Lightweight Authentication Protocol (SLAP). The protocol can resist well-known attacks and does not demand complex computational processes. Compared to the previous protocols, SLAP is considered as the most secure RFID authentication protocol which is appropriate for lightweight RFID environment. However, as demonstrated by Akgün and Çaglayan [10], server impersonation attack can be performed against SLAP. The demonstration shows the possibility of impersonation attacks which breaks the synchronization between the server and the tag without having to know the internal state of the tag. In addressing the issue, Akgün and Çaglayan [10] has proposed a revised SLAP which nearly the same as the original work by simply reordering the content of the authentication message (from server to tag).

Notwithstanding a very compact implementation, either SLAP or revised SLAP reveals partial of the secret (based on the number of tag available within the system) to assist the server in finding the corresponding secret. Such an attempt can pave a way for an attacker to deceive the tag to reveal parts of the secret, at most its first quarter [9]. Although it is hard to accomplish, revealing at this portion could give higher possibility on other threats such as pattern analysis and brute-force (guessing) to deduce the complete secret. For that reason, an alternative mutual authentication protocol, further known as PLAP, is proposed which highly maintains the confidentiality of the secret. The security level of the exchanged authentication messages in PLAP is further discussed on Section 4.2.

## 2. Provably Lightweight Authentication Protocol (PLAP)

PLAP is initially designed to overcome security weaknesses of SLAP, primarily performing authentication without revealing any portion of the secret and at the same time capable to prevent the server impersonation attack which causes de-synchronization between the tag and the server as addressed in revised SLAP [10]. PLAP utilizes hash function, exclu-

sive $OR$ ($XOR$) operation, and pseudorandom number generator (PRNG) in performing the authentication. Figure 2 illustrates the overall authentication steps in PLAP. PLAP consists of two phases, which are the initialization phase and the authentication phase as described in the following subsections. Table 1 summarizes the notations used in this paper.

### 2.1. Initialization Phase

For each new tag, the server initializes three values, namely authentication key ($K_0$), access key ($P_0$), and database index ($C_0$). The initial $K_0$ and $P_0$ values are randomly generated while $C_0$ set to 0 by default. Figure 3 illustrates the initialization phase of PLAP. In this example, when a new RFID tag $X$ is being introduced, the server stores the initialized values to the tag as $K_i = K_0$, $P_i = P_0$, and $C_i = C_0$. Finally, the corresponding records are also stored by the server in the back-end database based on the tags $TID$ as $K_{old} = K_0$, $K_{new} = K_0$, $P_{old} = P_0$, $P_{new} = P_0$, $C_{old} = C_0$, and $C_{new} = C_0$.

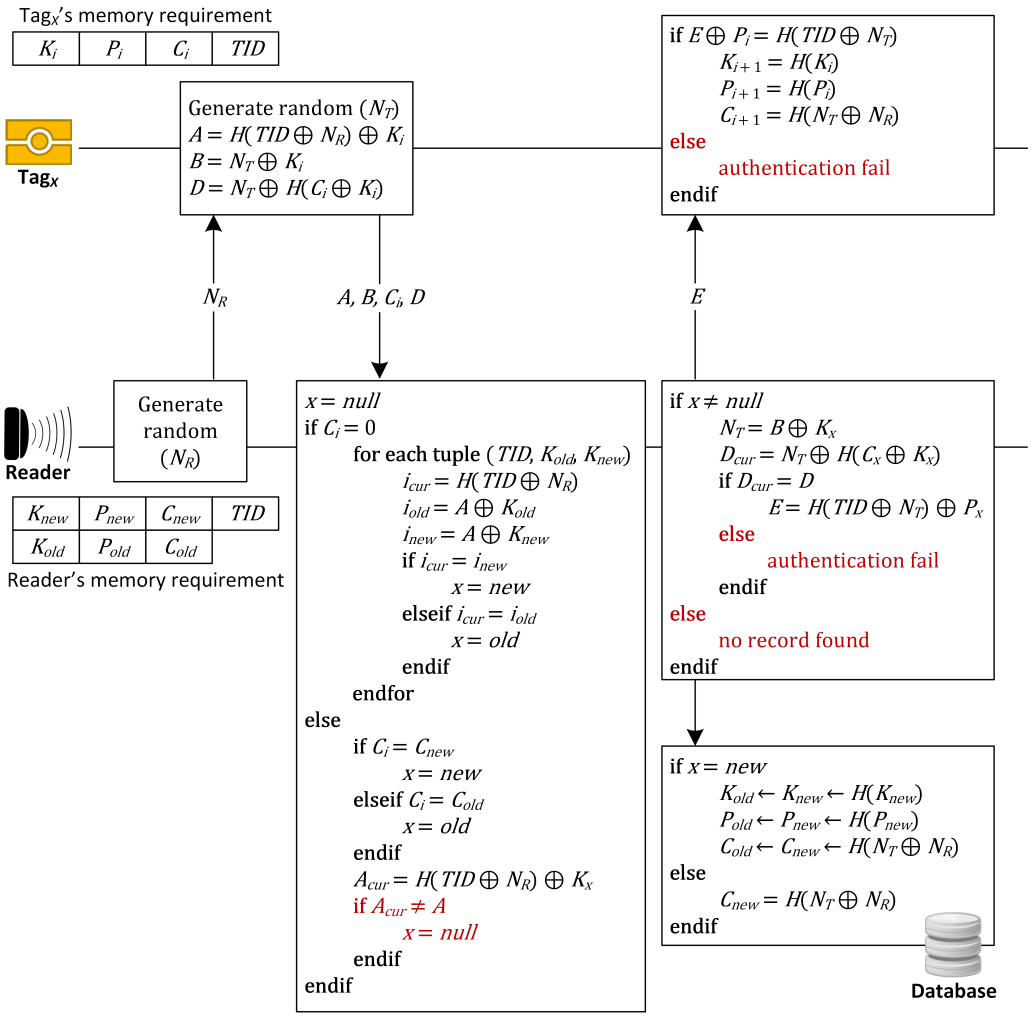



**Figure 2: The overall authentication steps of PLAP**

**Table 1: The notations used**

| Notation | Description |
|---|---|
| $TID$ | Tag Identification. |
| $N_R$ | Random number generated by the reader as the authentication challenge. |
| $N_T$ | Random number generated by the tag. |
| $K_i$ | Authentication key stored in the tag for the database to authenticate the tag at the $(i+1)^{\text{th}}$ authentication phase. |
| $K_{old}$ | Old authentication key stored in the database. |
| $K_{new}$ | New authentication key stored in the database. |
| $P_i$ | Access key stored in the tag for the database to authenticate the tag at the $(i+1)^{\text{th}}$ authentication phase. |
| $P_{old}$ | Old access key stored in the database. |
| $P_{new}$ | New access key stored in the database. |
| $C_i$ | Database index stored in the tag to find the corresponding record of the tag in the database. |
| $C_{old}$ | Old database index stored in the database. |
| $C_{new}$ | New database index stored in the database. |
| $H()$ | Hash function. |

### 2.2. Authentication Phase $(i+1)^{\text{th}}$

In this phase, the tag and the reader authenticate themselves mutually. The authentication phase involves three sequential steps. The steps are described as follows:

**Step 1** *(Reader* $\rightarrow Tag_x : N_R$*):*
The reader generates a nonce random, $N_R$, and sends $N_R$ to the tag as a challenge. Figure 4 illustrates Step 1 of the authentication phase. After $N_R$ is received, the tag generates another random number $N_T$ and uses both random values along with the stored values in the tag ($K_i$, $P_i$, $C_i$, and $TID$) to create the message for authentication purpose which consists of $A$, $B$, $C_i$, and $D$ values. $A$, $B$, and $D$ sub-messages are calculated by using the following formulas:

$$\begin{aligned} A &= H(TID \oplus N_R) \oplus K_i \\ B &= N_T \oplus K_i \\ D &= N_T \oplus H(C_i \oplus K_i) \end{aligned} \tag{1}$$

**Step 2** *(*$Tag_x \rightarrow Reader : A, B, C_i, D$*):*
Figure 5 illustrates Step 2 of the authentication phase. The sub-messages, $A$, $B$, $C_i$, and $D$, are bundled and sent to the reader which continued to the server. Upon receiving the message, the server checks $C_i$ value. It is considered as the first time access if $C_i = 0$. The server picks up every record in the database sequentially and computes three values, $i_{old}$, $i_{new}$, and $i_{cur}$, based on the received $A$ and the stored values, $K_{old}$, $K_{new}$, and $TID$, such that $i_{old} = A \oplus K_{old}$, $i_{new} = A \oplus K_{new}$, and $i_{cur} = H(TID \oplus N_R)$. When either $i_{cur} = i_{new}$ or $i_{cur} = i_{old}$ is found, the server sets $x$ as *new* or *old* accordingly. On the other hand, if $C_i \neq 0$, the server uses $C_i$ as the index to look up for the corresponding record in the database. When either $C_i = C_{new}$ or $C_i = C_{old}$ is found, the server respectively marks $x$ as *new* or *old* and recalculates $A$ as $A_{cur}$ where $A_{cur} = H(TID \oplus N_R) \oplus K_x$. Failing to find $x$ ($x = null$) indicates that the records between the database and the stored values

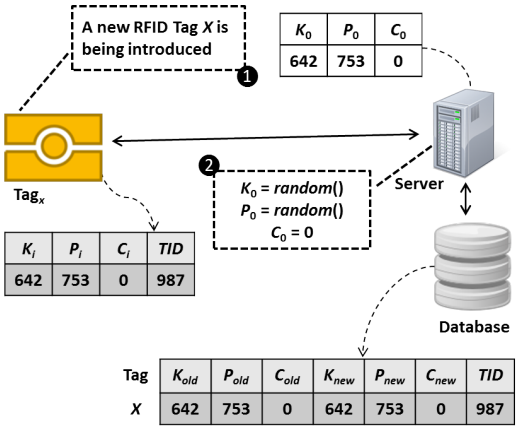



**Figure 3: Initialization of PLAP**

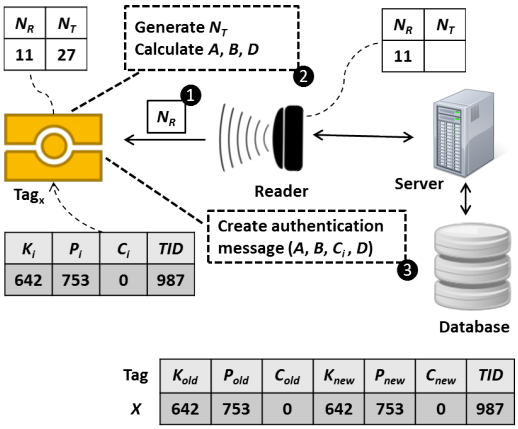



**Figure 4: PLAP Step 1 Authentication**

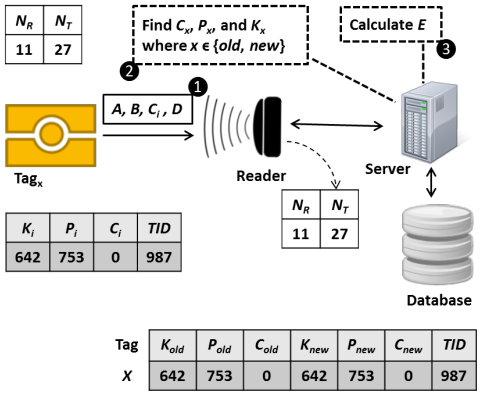



**Figure 5: PLAP Step 2 Authentication**

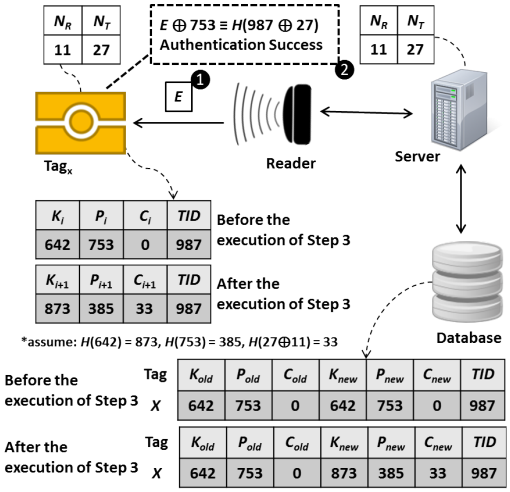



**Figure 6: PLAP Step 3 Authentication**

in Tag$_x$ are neither found nor matched. In the success condition, the server retrieves the value of $N_T$ from the sub-message $B$ using the formula $N_T = B \oplus K_x$. The server then computes $D_{cur} = N_T \oplus H(C_x \oplus K_x)$ and compares this value with the received sub-message $D$. If both values are equal, the server calculates message $E$ as $E = H(TID \oplus N_T) \oplus P_x$. Otherwise, the server aborts the protocol and ends the session.

**Step 3** *(Reader → Tag$_x$ : E):*
Figure 6 illustrates Step 3 of the authentication phase. The server forwards $E$ to the tag through the reader. Afterward, the server updates the stored values, $C_{old}$, $C_{new}$, $K_{old}$, $K_{new}$, $P_{old}$, and $P_{new}$, based on the identified $x$'s value. If $x = old$ then $C_{new} = H(N_T \oplus N_R)$ while the rest of the values remain unchanged. If $x = new$ then the stored values labeled as *new* in the current session ($C_{new}$, $K_{new}$, and $P_{new}$) will become the *old* values for the next session ($C_{old}$, $K_{old}$, and $P_{old}$), which is calculated as $K_{old} = K_{new}$, $P_{old} = P_{new}$, and $C_{old} = C_{new}$. Furthermore, the *new* values are computed for the next session as $K_{new} = H(K_{new})$, $P_{new} = H(P_{new})$, and $C_{new} = H(N_T \oplus N_R)$. Meanwhile in Tag$_x$, after receiving $E$, the tag computes $H(TID \oplus N_T)$ and computes $E \oplus P_i$ using the received $E$. If both values have the same result, the authentication has been successfully carried out

and the current records in the tag are updated excluding $TID$, such that $K_{i+1} = H(K_i)$, $P_{i+1} = H(P_i)$, and $C_{i+1} = H(N_T \oplus N_R)$. On the other hand, the authentication protocol is failed when $E \oplus P_i \neq H(TID \oplus N_T)$, and the old values are preserved in the tag.

## 3. SLAP vs. PLAP

This section discusses the differences between the existing protocol, SLAP, and the proposed protocol, PLAP. The protocols are compared from three points of view which are the initialization, mutual authentication, and updating.

### 3.1. Initialization

In the initialization phase of SLAP, the server stores two values which are $ID_{i-1}$ and $ID_{i-2}$ whereas the tag only stores one value, $ID_{i-1}$. On the other hand, the server in PLAP stores seven values which are $K_{old}$, $P_{old}$, $C_{old}$, $K_{new}$, $P_{new}$, $C_{new}$ and $TID$, while the tag stores four values which are $K_i$, $P_i$, $C_i$, and $TID$. Table 2 shows the values initialized in SLAP and PLAP during the initialization phase. PLAP uses more storage space compared to SLAP in order to prevent the de-synchronization from happening. PLAP stores all the values from the current session and the previous successful session. By storing all these values, the possibility of server impersonation and de-synchronization are eliminated. This matter will further be discussed in Section 4.

### 3.2. Mutual Authentication

In SLAP's mutual authentication phase, the reader (as the server's front-end) sends message $H(N+1, ID_i)$ to the tag whereas the tag sends message $ID_{i-I}^{[logT]}$ and $H(N, ID_i)$ to the reader. In PLAP, the tag sends authentication message ($A$, $B$, $C_i$, and $D$) to the reader during the mutual authentication phase. As the response, the reader replies with message $E$ to the tag. Table 3 shows the messages transmitted during the mutual authentication phase in SLAP and PLAP.

In PLAP, the index $C_i$ is stored on the tag as well as the back-end database. For the database, $C_i$ is used for finding the corresponding record of a given tag in the database.

**Table 2: The initialized values in SLAP and PLAP initialization phase**

| | Server and Back-end database | Tag |
|---|---|---|
| **SLAP** | $ID_{i-1}$ and $ID_{i-2}$ | $ID_{i-1}$ |
| **PLAP** | $K_{old}$, $P_{old}$, $C_{old}$, $K_{new}$, $P_{new}$, $C_{new}$, and $TID$ | $K_i$, $P_i$, $C_i$, and $TID$ |

**Table 3: The exchanged messages in SLAP and PLAP mutual authentication phase**

| | Reader to Tag | Tag to Reader |
|---|---|---|
| **SLAP** | $H(N+1, ID_i)$ | $ID_{i-I}^{[logT]}$ and $H(N, ID_i)$ |
| **PLAP** | $E = H(TID \oplus N_T) \oplus P_x$ | $A = H(TID \oplus N_R) \oplus K_i$,<br>$B = N_T \oplus K_i$,<br>$C_i$ = last successful database index,<br>and $D = N_T \oplus H(C_i \oplus K_i)$ |

**Table 4: The updated values in SLAP and PLAP updating phase**

| | Server and Back-end database | Tag |
|---|---|---|
| **SLAP** | $ID_{i-1}$ and $ID_{i-2}$ | $ID_{i-1}$ |
| **PLAP** | $K_{old}$, $P_{old}$, $C_{old}$, $K_{new}$, $P_{new}$, and $C_{new}$ | $K_{i+1}$, $P_{i+1}$, and $C_{i+1}$ |

Sub-message $A$ is used to camouflage the tag $ID$ ($TID$) whereas the objective of having sub-message $B$ is to hide the random number generated by the tag ($N_T$). In addition, $TID$ is used as tag identification while $N_T$ is generated by the tag to identify the reader. Sub-message $D$ on the other hand, is required for the integrity check on both sub-messages $A$ and $B$. Finally, message $E$ uses $P_x$ to create a fresh reply to avoid any pattern analysis on the exchanged messages.

### 3.3. Updating

Table 4 shows the updated values of SLAP and PLAP during the updating phase which is executed after a successful authentication. The server in SLAP updates two values to the back-end database which are $ID_{i-1}$ and $ID_{i-2}$ whereas the tag updates one value which is $ID_{i-1}$.Meanwhile in PLAP, the tag updates three values, which are $K_{i+1}$, $P_{i+1}$, and $C_{i+1}$, whereas the server updates two types of variables (*old* and *new*) with a total of six values which are $K_{old}$, $P_{old}$, $C_{old}$, $K_{new}$, $P_{new}$, and $C_{new}$. The *new* and *old* variables represent the values of the current session and the previous successful session, respectively. This mechanism aims to prevent the de-synchronization from happening.

## 4. PLAP Implementation and Evaluation

This section analyzes PLAP in term of security. PLAP is coded in Python and then tested for randomness. In Subsection 4.1, AVISPA tool is used to verify the implementation of PLAP against SLAP. Subsection 4.2 shows the results and analysis of random test experiment which describes the level of difficulty in performing pattern analysis on the authentication message on PLAP. Furthermore, PLAP is also being analyzed against the known attacks as described in Subsection 4.3.

### 4.1. Protocol Verification Using AVISPA Tool

Figure 7 shows the overall architecture of AVISPA, which includes Security Protocol Animator (SPAN) that was initially developed by Olivier Heen and Olivier Courtay [11]. SLAP was simulated in parallel with PLAP to determine the reactions of both protocols in taking care of security attacks on AVISPA.

#### 4.1.1. The Back-Ends of AVISPA Tool

AVISPA tool consists of four back-ends: On-the-Fly Model-Checker (OFMC), Constraint-Logic-Based Attack Searcher (CL-AtSe), SAT-Based Model-Checker (SATMC), and Tree Automata-Based Protocol Analyser (TA4SP). The four back-ends of AVISPA are described as follows:

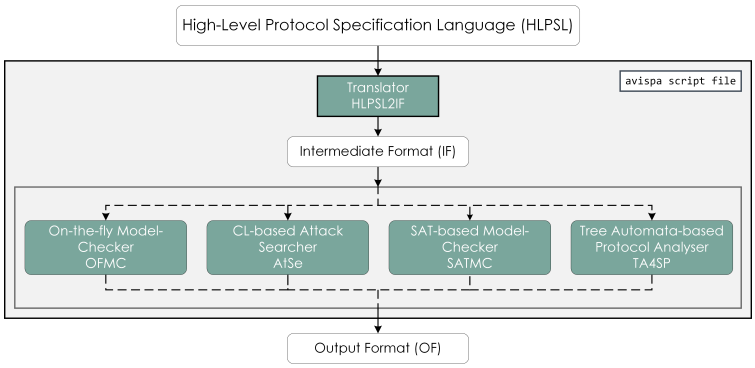



**Figure 7: Architecture of the AVISPA Tool**

1. *On-the-Fly Model-Checker (OFMC)*
   OFMC performs both protocol falsification and bounded session verification, by exploring the transition system described by the given specification in a demand-driven way (on-the-fly). OFMC considers both typed and un-typed protocol models. OFMC integrates number of symbolic constraint-based techniques and implements a number of efficient heuristic search algorithms. It also provides support for the modeling of an intruder who is capable of performing guessing attacks on weak passwords, and for the specification of algebraic properties of cryptographic operators [12].

2. *Constraint-Logic-Based Attack Searcher (CL-AtSe)*
   CL-AtSe applies constraint solving to perform both protocol falsification and verification for bounded number of sessions. The protocol messages can be typed or un-typed, and the pairing can be considered to be associative or not associative. Several properties of the XOR operator can be handled. CL-AtSe is built in a modular way and thus opens to extensions for handling algebraic properties of cryptographic operators. CL-AtSe performs several kinds of optimizations to reduce redundancies or useless branches in the protocol symbolic execution [13].

3. *SAT-Based Model-Checker (SATMC)*
   SATMC considers the typed protocol model and performs both protocol falsification and bounded session verification by reducing the input problem to a sequence of SAT solvers. SATMC first builds the transition relation specified by the given specification. The initial propositional formula encodes a bounded unrolling of state, and the set of states representing a violation of the security properties. The propositional formula is then fed to a SAT solver and any model found is translated back into an attack [14].

4. *Tree Automata-Based Protocol Analyser (TA4SP)*
   TA4SP performs unbounded protocol verification by approximating the intruder knowledge by using regular tree languages [15]. TA4SP starting point is an extension of an approximation method based on tree automata, introduced by Genet and Klay [16] to verify security protocols. The presence of an expert is prerequisite in this tool to transform by hand a security protocol into a term-rewriting system and compute an ad-hoc approximation function. In addition, an approximation function can automatically be calculated using AVISPA. For secrecy properties in the typed model, TA4SP can show whether a protocol is flawed (by under-approximation) or safe.

#### 4.1.2. AVISPA Testing on SLAP and PLAP

The implementation of the SLAP in AVISPA shows that attacks are eminent in such a way that an intruder can masquerade and get authenticated by the server simply by using a man-in-the-middle attack. Figure 8 shows the conducted testing by using AVISPA back-ends. Figure 9(a) shows the code simulating the three messages of SLAP, which uses random nonce and hash function on the tag and the reader. The figure describes the code for simulating SLAP in HLPSL back-end of AVISPA tool. The CAS+ programing language was designed specifically to facilitate specification writing of security protocols.

In Step 1, the reader sends $nonce - 1$ (random number) to the tag. In Step 2, the tag sends the hash value of $const_1(N, ID_i)$ and $const_1(ID_{i-1}^{[logT]})$ which are being eavesdropped by the intruder. In Step 3, the intruder replays $const_1(N, ID_i)$ and $const_1(ID_{i-1}^{[logT]})$ back to the reader. Consequently, the reader recognizes the intruder as a legitimate tag thus the impersonation attack has been successfully mounted. OFMC and CL-AtSe confirm that the SLAP is vulnerable. Meanwhile, SATMC and TA4SP show inconclusive in their tests.

Figure 9(b) describes the implementation of PLAP in CAS+ language. A scenario for PLAP is implemented to achieve the two authentication goals: the tag authenticates the reader and the reader authenticates the tag. In PLAP a randomly generated number is produced on both sides (the reader and the tag) on every session denoted as $N_R$ and $N_T$, respectively. PLAP uses hash function and $XOR$ operation. The results show that PLAP is secured where the intruder is unable to masquerade as the server. PLAP is tested against all the AVISPA back-ends. OFMC and CL-AtSe show that PLAP is safe, while SATMC and TA4SP indicate that the test results are inconclusive as shown in Table 5.

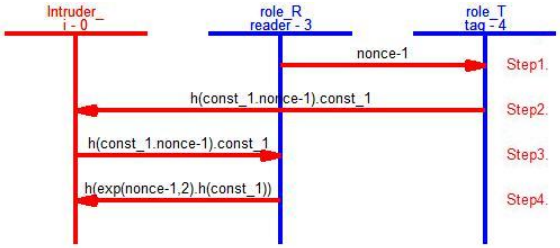



**Figure 8: Intruder attack-snapshot taken from AVISPA running SLAP**

```
protocol SLAP;

identifiers
T, R      : user;                     /* T = tag; R = reader */
Nt        : number;                   /* Nt = tag's random number; */
ID        : number;                   /* ID = identification number */
H         : function;                 /* H = hash function */

messages
1. R -> T : Nt                        /* reader send Nt to tag */
2. T -> R : H(ID, Nt),ID              /* tag reply H(ID,Nt),ID to reader */
3. R -> T : H(Nt^2, H(ID))            /* reader reply H(Nt^2, H(ID)) to tag */

knowledge
T         : T, R, ID, H;              /* tag knows these values */
R         : T, R, H;                  /* reader knows these values */

session_instances
[T:tag, R:reader,H:h];

goal
R authenticates T on Nt;              /* value used for tag authentication */
T authenticates R on H(Nt^2, H(ID));  /* value used for reader authentication */
```

(a) SLAP code

```
protocol PLAP;

identifiers
T, R      : user;                     /* T = tag; R = reader */
Nr, Nt, P : number;                   /* Nr = reader's random numbers */
                                      /* Nt = tag's random number */
                                      /* P = access key */
K, ID, C  : number;                   /* ID = identification number */
                                      /* K = authentication key */
                                      /* C = database index */
H         : function;                 /* H = hash function */

messages
1. R -> T : Nt                        /* reader send Nt to tag */
2. T -> R : H(ID# Nt)#ID , Nt#K , Nt#H(C#K)  /* tag reply A, B, C, D to reader */
3. R -> T : H(ID#Nt)#P                /* reader reply to tag */

knowledge
T         : T, R, K, ID, H, C, P;     /* tag knows these values */
R         : T, R, K, H, C, P;         /* reader knows these values */

session_instances
[T:tag, R:reader, H:h];

goal
R authenticates T on Nt;              /* value used for tag authentication */
T authenticates R on Nr;              /* value used for reader authentication */
```

(b) PLAP code

**Figure 9: The codes used for the implementation on AVISPA**

**Table 5: AVISPA results for SLAP and PLAP protocols**

| | SLAP | PLAP |
|---|---|---|
| **OFMC** | UNSAFE | SAFE |
| **AtSe** | UNSAFE | SAFE |
| **SATMC** | INCONCLUSIVE | INCONCLUSIVE |
| **TA4SP** | INCONCLUSIVE | INCONCLUSIVE |

### 4.2. Random Test Experiment

The passive attacks aim to capture confidential data during the communication between the reader and the tag which actively involved in the communication. The basic model of this attack gathers a huge number of the exchanged messages and analyses these messages for discovering the pattern to understand the future messages. The main objective of this experiment is summarized as follows:

- To test the components validity and functionality of PLAP.
- To test the exchange messages of PLAP against randomness test.
- To ensure that guessing any future value from the previously exchanged sub-messages is improbable.

This experiment stored 181,440,000 values for each sub-message from the exchanged message ($A$, $B$, $C$, $D$, and $E$) during every authentication sessions [50 authentications  60 seconds  60 minutes  24 hours  7 days]. The experiment was made under the assumption that a tag can be authenticated successfully 50 times per second, which is beyond the real-life scenario. The experiment was run for seven days based on a 24/7 basis execution and stored all these exchanged messages for later analysis (based on the experimental design specification by Peris-Lopez et al. [17]).

Figure 10 shows the Python code used in calculating PLAP's sub-message $A$, $B$, $D$, and $E$ for randomness test. Sub-message $C$ was excluded from the test because sub-message $C$ is theoretically random. Sub-message $C$, calculated as $C = H(N_T \oplus N_R)$, is based on the random nonces that were generated by the tag ($N_T$) and the reader ($N_R$) from the last successful authentication. Unlike the other sub-messages, the value of sub-message $C$ is not derived from any secret key ($K$ or $P$). Therefore, sub-message $C$ is publicly safe and does not give any benefits to the adversary in performing passive attacks such as guessing and pattern analysis on sub-message $C$.

PLAP prototype was created to simulate the execution of the message. Figure 11 shows the output of the prototype for three protocol sessions. A randomization test has been carried to analyze potential hidden pattern in the sub-messages. ENT [18] and Diehard [19] were used for the randomization test. The resulted values are grouped by the cumulative $P$-values for every algorithm in the same test. All the $P$-values obtained from the Diehard

```
A = (hash (TID ^ Nr) ^ Ki) % 2**96
B = (Nt ^ Ki) % 2**96
D = Nt ^ hash (Ci^ Ki) % 2**96
E = hash (TID^Nr) ^ Pi
```

**Figure 10: Calculation of sub-message $A$, $B$, $D$, and $E$**

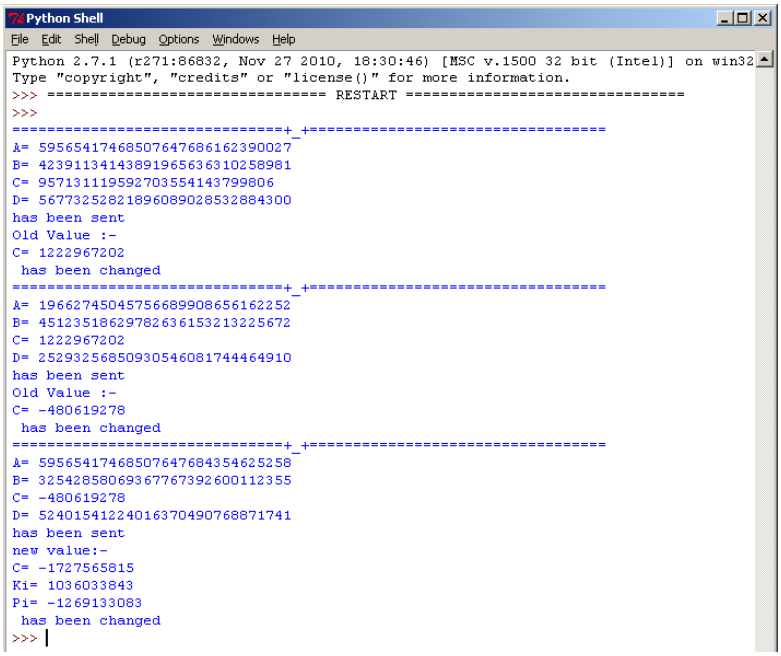



**Figure 11: The output of PLAP prototype**

tests are summarized by a formula [20] to arrive at one result. As for the ENT test, it uses the same input files as Diehard. However, the results of ENT are the measurements of the entropy, compression rate, $X^2$ statistics, arithmetic mean, Monte Carlo $\pi$ estimation, and serial correlation coefficient. The values obtained and their significance is explained in the following experimental results.

Table 6 shows the correlation coefficients for all the sub-messages' values based on the ENT test. ENT was used to prove the independency of each sub-message. Based on Bland and Altman [21], the serial correlation coefficient for each sub-message in Table 6, which is below than 0.05, indicates that these sub-messages has no relationship among each other. The uncertainty associated with a random value is measured by using the entropy rate which depends on the value closeness to a value of 8 bit/bytes. In this experiment, the entropy rate used is relatively 7.9 bit/bytes. According to Roman [22], this result indicates that the sub-messages are highly randomized. Furthermore, the arithmetic means are simply the result of adding all the bytes in the file and dividing this sum by the file length. Any

**Table 6: ENT results for sub-message $A$, $B$, $D$, and $E$**

| Test | $A$ | $B$ | $D$ | $E$ |
|---|---|---|---|---|
| Entropy | 7.992462 | 7.843656 | 7.935505 | 7.978678 |
| Compression rate | 0% | 0% | 0% | 0% |
| $X^2$ statistics | 1776228 (0.01) | 2615356 (0.01) | 7333560 (0.01) | 172134 (0.01) |
| Arithmetic mean | 127.4551 | 127.0252 | 127.5766 | 127.12345 |
| Monte Carlo $\pi$ estimation | 3.1454556713 (0.07) | 3.70675467750 (0.08) | 3.1467667567 (0.08) | 3.2345678 (0.06) |
| Serial correlation coefficient | 0.045649 | -0.006771 | 0.0014856 | - |
| $P$-value | 0.852141 | 0.546567 | 0.654778 | 0.24765 |

result that is close to the value of 127.5, it means that the tested bytes are close to random. In this experiment, the mean value for each sub-message is roughly near to 127.5, which indicates randomness.

Table 7 shows the Diehard result for each sub-message $A$, $B$, $D$, and $E$. Any $P$-value from the Diehard test that falls between $0.025 < P < 0.951$ indicates randomness of the tested data (sub-messages exchanged between the tag and the reader). As shown in Table 7, the $P$-value of each sub-message is within this range, thus proving that these sub-messages are random.

Table 8 shows the correlation among the groups of the PLAP's sub-messages ($A$, $B$, $C$, $D$, and $E$). Results presented in Table 8 show correlation values below the 0.05 threshold, which indicates that there is no evidence of correlation among the different sub-message groups. This ensures that the sub-messages are totally uncorrelated and there is no relationship within their respective pairs.

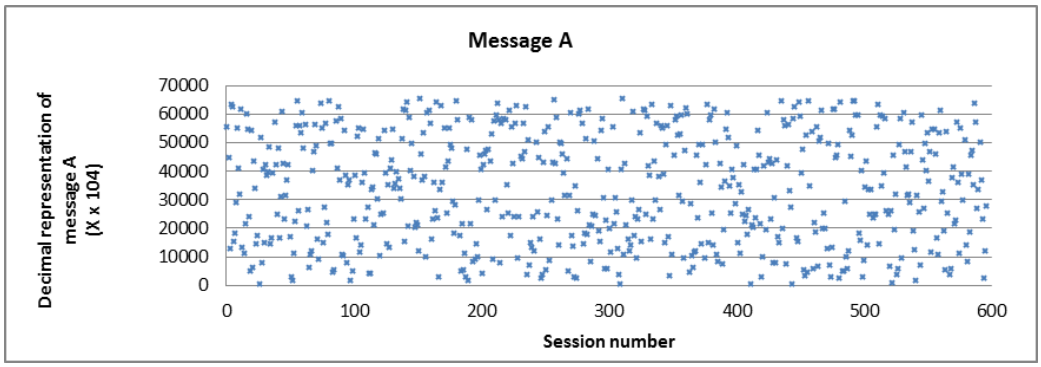
Figure 12(a)-(d) show the patttern distribution of 600 messages of sub-message $A$, $B$, $D$, and $E$, respectively. All results show that the pattern are distributed evenly and there is no similar pattern among them. Based on the randomness tests conducted, it can be concluded that it is quite impossible for an adversary to find any patterns from the exchanged messages or even predict the values in the future sessions from the previous messages.

**Table 7: Diehard results for sub-message $A$, $B$, $D$, and $E$**

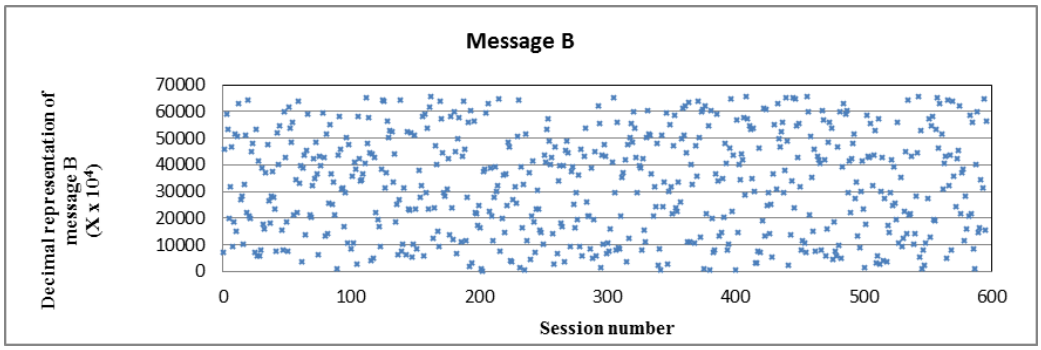
| | Single Block of Message | $A$ | $B$ | $D$ | $E$ |
|---|---|---|---|---|---|
| 1 | Birthday Spacing | 0.828970 | 0.732251 | 0.558398 | 0.839424 |
| 2 | Overlapping Permutations | 0.200375 | 0.278705 | 0.743263 | 0.511580 |
| 3 | Ranks of $31x31$ Matrices | 0.874531 | 0.787141 | 0.461387 | 0.545772 |
| 4 | Ranks of $32x32$ Matrices | 0.426338 | 0.321355 | 0.821813 | 0.712700 |
| 5 | Ranks of $6x8$ Matrices | 0.444857 | 0.798188 | 0.815494 | 0.524589 |
| 6 | Monkey Tests on 20-bit Words | 0.82251 | 0.50342 | 0.42648 | 0.45783 |
| 7 | Monkey Tests OPSO | 0.2216 | 0.1332 | 0.8280 | 0.3366 |
| 8 | Monkey Tests OQSO | 0.2086 | 0.8363 | 0.8221 | 0.4230 |
| 9 | Monkey Tests DNA | 0.6999 | 0.2862 | 0.2447 | 0.6134 |
| 10 | Count the 1's in a stream of Bytes | 0.240794 | 0.384724 | 0.921048 | 0.234402 |
| 11 | Count of 1's in specific Bytes | 0.332788 | 0.973249 | 0.509690 | 0.616074 |
| 12 | Parking Lot Test | 0.716056 | 0.399306 | 0.938878 | 0.595877 |
| 13 | Minimum Distance Test | 0.206243 | 0.518985 | 0.579136 | 0.111473 |
| 14 | Random Spheres Test | 0.411549 | 0.335431 | 0.182763 | 0.825677 |
| 15 | The Squeeze Test | 0.210103 | 0.162722 | 0.672343 | 0.544095 |
| 16 | Overlapping Sums Test | 0.875708 | 0.589925 | 0.453111 | 0.855102 |
| 17 | Runs Test | 0.599953 | 0.884229 | 0.312131 | 0.350282 |
| 18 | The Craps Test | 0.74584 | 0.613981 | 0.689736 | 0.351393 |
| | $0.025 < P < 0.951$ | | **Pass the Diehard test** | | |

**Table 8: Correlation coefficient between all sub-messages ($A$, $B$, $C$, $D$, and $E$)**

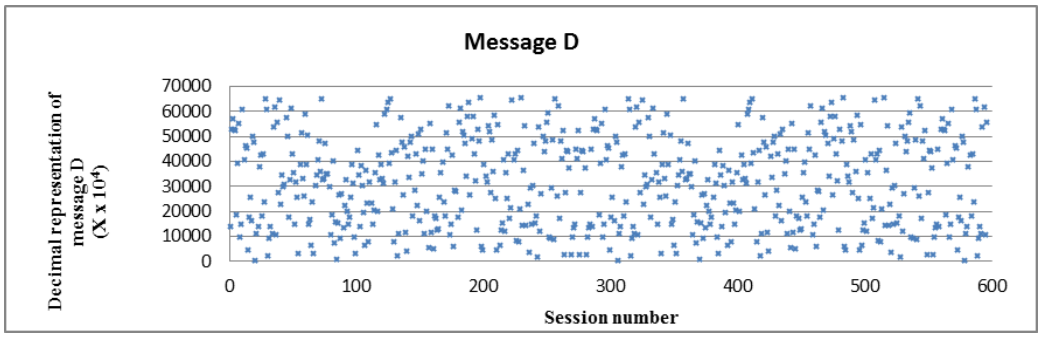
| Sub-Message | $A$ | $B$ | $C$ | $D$ | $E$ |
|---|---|---|---|---|---|
| $A$ | 1 | | | | |
| $B$ | 0.00034 | 1 | | | |
| $C$ | 0.000187 | -0.002543 | 1 | | |
| $D$ | 0.000215 | 0.000506 | -0.000241 | 1 | |
| $E$ | -0.009182 | 0.000287 | 0.000762 | 0.001817 | 1 |



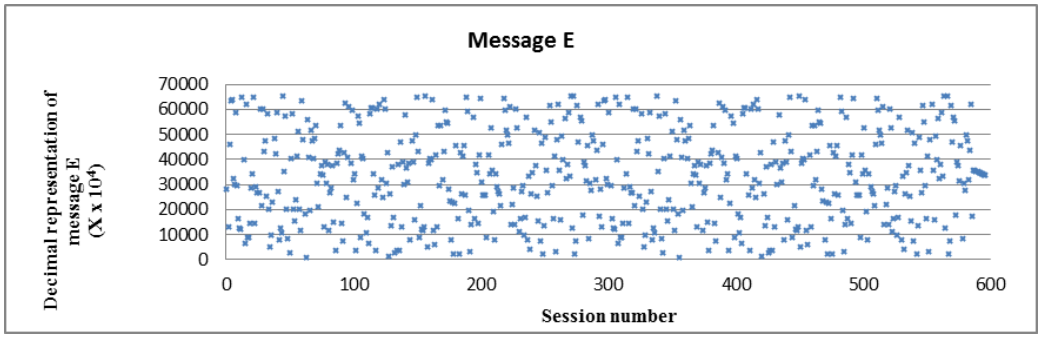

(a) Sub-message $A$




(b) Sub-message $B$




(c) Sub-message $D$




(d) Sub-message $E$

**Figure 12: Pattern distribution of 600 messages**

### 4.3. Security Analysis of PLAP

This subsection analyzes PLAP security features. The security is evaluated in term of data security and integrity, mutual authentication, anonymity of the tag, prevention of tracking attack, protection against replay attack and forward security, protection against impersonation attack, and protection from de-synchronization and DoS attack. Table 9 summarizes the security analysis of protocols, SLAP, revised SLAP [10], and PLAP.

#### 4.3.1. Data Security and Integrity of Message

In PLAP, the tag only stores the current value of $K_i$, $P_i$, and $C_i$, while the value of $TID$ remains unchanged. If an attacker is able to crack a tag and is capable of reading out its content, the attacker will still be unable to get any information about another tag. This is because messages always contain the hash of the tag's identity, in which the hash values differ for different tag. Therefore confidential information is secure, even if one of the tags is compromised.

#### 4.3.2. Mutual Authentication

The transaction identifier used in the $(i+1)^{\text{th}}$ transaction is only known by the tag and the reader. During the authentication process, this $TID$ is never sent in plain text. By using a one-way hash function an attacker will not be able to derive this identity. Therefore the communicating parties can be certain that the other partner is authentic, as an attacker cannot gain knowledge of the $TID$ during the transaction.

#### 4.3.3. Tag Anonymity

The usage of two random numbers ($N_T$, $N_R$) and the hash function in each message sent by the reader and the tag makes the messages undistinguishable from a spectator perspective. Since there is no confidential data sent in plain text, the tag can remains anonymous before a third-party.

#### 4.3.4. Prevention of Tracking Attack

In PLAP, all the information transmitted is scrambled. By utilizing the random number, fresh messages are constructed in each session. Since random-like messages lack pattern, PLAP is therefore strong against tracking attack.

**Table 9: Summary of security analysis**

| Protocol | SLAP | Revised SLAP | PLAP |
|---|---|---|---|
| Difficulty level of pattern analysis and guessing | Unproven | Unproven | Very High |
| Data integrity | Yes | Yes | Yes |
| Mutual authentication | Yes | Yes | Yes |
| Tag anonymity | Yes | Yes | Yes |
| Resistance against tracking | Yes | Yes | Yes |
| Resistance against impersonation attack | No | Yes | Yes |
| Resistance against reply attack | No | Yes | Yes |
| Resistance to de-synchronization attack | No | Yes | Yes |

#### 4.3.5. Protection Against Replay Attack and Forward Security

Replay attacks are avoided when randomly generated numbers $N_R$ and $N_T$ are used for information transmission. The keys are kept in the updated tag using the PRNG, which eliminates any possibility of reusing the record. However, sub-message $A$ and sub-message $D$ transmission may be intercepted, and the $XOR$ of these two values may compromise $TID$. Therefore there is a need for $N_T$ to complete the cycle, which is usually sent with the previous key which makes it inaccessible.

#### 4.3.6. Protection Against Impersonation Attack

In SLAP, the tag uses the message $M_2$ to authenticate the back-end server. The weakness of SLAP is that an adversary can create the valid $M_2$ without the knowledge of $ID_{i-1}$ and $ID_i$. The back-end server creates the valid $M_2$ using $N$, the random nonce, appended with the valid $ID_i$. The back-end server can create the valid $ID_i$ with the knowledge of $ID_{i-1}$. However, message $M_1$ (created by the tag) and message $M_2$ (created by the back-end server) has identical input structure $(N, ID_i)$. The only different is that $M_2$ increments the random nonce by 1, $N+1$. Since $N$ is public, with such structure, the adversary can compile a valid $M_2 = H(N+1, ID_i)$ by querying the tag with $N+1$ without the need to know $ID_i$.

The revised SLAP [10] resolves the issue by reordering $M_2$ as $M_2 = H(ID_i, N+1)$ thus makes the knowledge of $ID_{i-1}$ is an absolute requirement for the creation of $M_1$ and $M_2$. Comparable motive to the revised SLAP, in PLAP, sub-message $A$ is used for the server to authenticate the tag whereas sub-message $E$ is used for the tag to authenticate the server. By the means of these sub-messages, the tag and the server will authenticate each other mutually thus an attacker cannot impersonate the server as well as the tag.

#### 4.3.7. Protection from De-Synchronization and DoS Attack

PLAP maintains a dual key system (*old* key and *new* key). The keys' values are kept in the back-end database to prevent asynchronous updates, tempered, or lost. A checking mechanism prevents the DoS attack due to synchronization. The mechanism is designed such that the matching record in the database is identified by matching up the field $C_{old}$. The values for $K_{old}$, $P_{old}$, $C_{old}$, $K_{new}$, and $P_{new}$ will be kept the same instead of being replaced by new values.

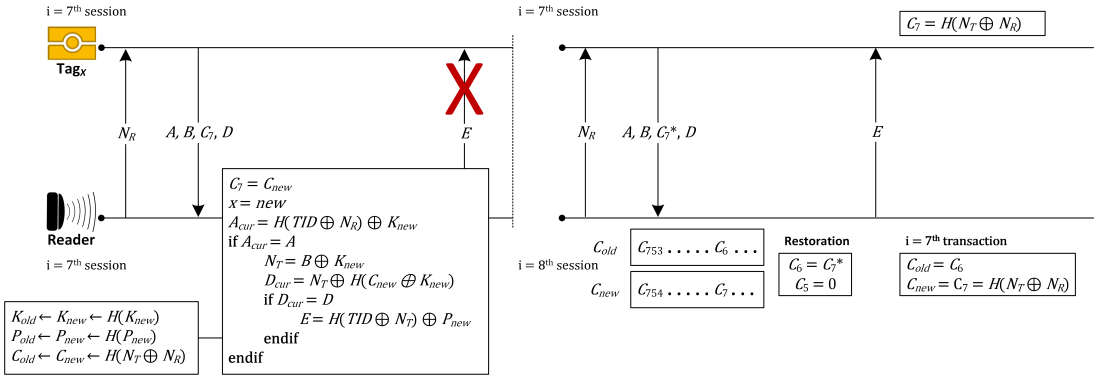



**Figure 13: The synchronization between the tag and the reader**

Figure 13 demonstrates an example of a re-synchronization process between the tag and the reader. This example assumes that the $6^{\text{th}}$ authentication session between the tag and the reader has been completed successfully whereby in the $7^{\text{th}}$ session, it assumes that the transmission of the last message ($E$) is not being received by the tag. Consequently, the tag does not update its local values. Fortunately, in the following session, PLAP has the mechanism in which the back-end database checks its synchronization state with the tag. In the case of de-synchronized state as in the aforementioned example, the reader compares the received $C_i$ value with the latest $C_{new}$ value in the database. If the values are not found, in other words de-synchronized, the reader then compares the received $C_i$ with the latest $C_{old}$ value and updates the local variable and respond with an updated $C_i$ value to the tag. Finally, the tag responses accordingly by using the received value and updates its local values. As a result, the tag and the reader are in synchronized state again.

## 5. Performance Analysis

This subsection evaluates the performance of PLAP in term of computational cost, communication cost, and storage cost. Table 10 compares the performances between PLAP and SLAP, which either original or revised SLAP [10] has exactly the same performance.

### 5.1. Computational Cost

PLAP uses hash function, PRNG, and $XOR$ operations which can easily be implemented on an RFID tag chip. The back-end server performs most of the heavy computational operations which make PLAP computationally cost effective as a whole. In the process of securing PLAP transactions, two authentication keys ($K_i$, $P_i$) were added to the PLAP. Accordingly, the number of hash function has notably increase to six hash functions in the new protocol instead of three hash functions which were found in SLAP.

### 5.2. Communication Cost

In SLAP, the messages transmitted are very compact. The length of the random number $N$ is 128 bits, the one-way function $H$ is a $SHA-1$ hash function which produces 160 bits hash digest. Therefore the length of a message in SLAP is at most 170-180 bits, depending

**Table 10: Performance comparison between SLAP and PLAP**

| | Entity | SLAP | PLAP |
|---|---|---|---|
| Number of Hash Operations | $T$ | 3 | 6 |
| | $B$ | 3 | 6 |
| Number of PRNG Operations | $T$ | - | 1 |
| | $R+B$ | 1 | 1 |
| Number of Hash Operations | $T$ | 3 | 15 |
| | $R+B$ | 3 | 14 |
| Number of Authentication Steps | - | 3 | 3 |
| Number of Hash Operations | $T$ | 160 | 672 |
| | $R+B$ | 320 | 384 |
| **Total number of bits exchanged in one complete protocol session** | | 478 | 576 |

on the number of tags in the system. In the case of an ecosystem that consists of 1024 tags, it is enough to forward the first 10 bits ($log_2 1024$) to the back-end server in order to find a given tag promptly. Therefore, for SLAP, it is estimated that 478 bits of message [128 + 10 + (2170)] is transmitted in one protocol session. On the other hand PLAP creates 576 bits [96 6] from session-start until session-termination. $N_T$ has 96 bits in the first message transmission, four other sub-messages with a total of 384 bits in the second message transmission, and another 96 bits in the third transmission.

### 5.3. Storage Requirement

In SLAP, the tag stores $ID_{i-1}$ identification number which is also known by the back-end server. On top of storing $ID_{i-1}$, the back-end server also needs to store the $ID_{i-2}$ value which has an important role in the protocol to re-establish the synchronization between the back-end server and the tag. As mentioned before in the previous subsection, the output length of the one-way hash function is 160 bits. Therefore, in SLAP, the reader is required to store 320 bits of information (per tag), whereas the tag is required to store only 160 bits of information. Meanwhile in PLAP, $TID$ has two sets of common keys ($\{K_{old}, P_{old}, C_{old}\}$, $\{K_{new}, P_{new}, C_{new}\}$), which carries a total of 672 bits [96 7] per tag at the back-end database. In comparison, PLAP stores $K_i$, $P_i$, $C_i$, and $TID$ on the tag. Each of these values has a length of 96-bit. Therefore a total of 384 bits [96 4] of rewritable memory is required on each tag.

## 6. Conclusion

PLAP is proposed in this paper, which is a provably alternative mutual authentication protocol for lightweight RFID. AVISPA security protocol verifier is used to test the implementation of PLAP and SLAP. The test proves that SLAP allows an intruder to perform server impersonation which has been addressed in the revised SLAP. The test also verifies that PLAP has the imunity against server impersonation. In addition, PLAP has been evaluated in term of security and performance. The results show that with a marginal overhead, PLAP is capable of preventing the known security attacks on RFID authentication and meets the requirements for a lightweight RFID.

Moreover, SLAP and the revised SLAP rely heavily on the hash function to endure against the passive attacks such as pattern analysis and brute-force guessing. The attack chances to succeed become higher since the protocols partially reveal the secret on each session. Although PLAP has a relatively higher requirement than the SLAPs, PLAP does not reveal any crucial secret and has been proved to be difficult in performing the passive attacks which is considered as a trade-off between security and cost. Therefore, compared to SLAP, PLAP is a safer option to preserve confidentially of the authentication key.

PLAP is intended for devices with low computational capacity which at least can perform $XOR$ operation, random number generation, and hash function. With this advantage, further design and development of PLAP is highly possible to be adapted on different communication ecosystems such as Wi-Fi, Bluetooth, near field communication (NFC), and Infrared which are commonly available in the recent fast-growing devices on mobile technology. Finally, in terms of the design, SLAP has a simplicity advantage rather than PLAP. Hence, redesigning the protocol with lower complexity has also been noted for the future consideration of this research.

## 7. ACKNOWLEDGEMENTS

The authors would like to thank the anonymous reviewers for their useful comments. This research is supported by Fundamental Research Grant Scheme (203/PKOMP/6711156) from Malaysian Ministry of Higher Education.

## 8. References

[1] S. Choi, C. Poon, "An RFID-based Anti-counterfeiting System," IAENG International Journal of Computer Science, vol. 35, no. 1, pp. 1-12, **2008**.

[2] C. Chan, H. Chow, W. Siu, H. Ng, T. Chu, H. Chan, "A Multi-Agent-based RFID Framework for Smart-object Applications," In Proceeding of the International Multiconference of Engineers and Computer Scientists (IMECS), vol. I, pp. 356-361, **2012**.

[3] A. Juels, "Strengthening EPC Tag against Clonin," In Proc. ACM Workshop Wireless Security (WiSe '05), pp. 67-76, **2005**.

[4] P. Cole, D. Ranasinghe, "Networked RFID Systems and Lightweight Cryptography," Springer-Verlag, **2008**.

[5] M. Ohkubo, K. Suzuki, S. Kinoshita, "Cryptographic Approach to 'Privacy-Friendly' Tag," RFID Privacy Workshop, MIT, Nov., **2003**.

[6] D. Henrici, P. Műller, "Hash-Based Enhancement of Location Privacy for Radio Frequency Identification Devices using Varying Identifiers," In Workshop on Pervasive Computing and Communications Security, pp. 149-153, **2004**.

[7] T. Dimitriou, "A Lightweight RFID Protocol to Protect against Traceability and Cloning Attack," In International Conference on Security and Privacy for Emerging Areas in Communication Networks-SecComm, pp. 59-66, **2005**.

[8] Z. Luo, T. Chan, J. S. Li, "A Lightweight Mutual Authentication Protocol for RFID Networks," In International Conference on e-Business Engineering, pp. 620-625, **2005**.

[9] G. Gódor, M. Antal, S. Imre, "Mutual Authentication Protocol for Low Computational Capacity RFID Systems," In Proceedings of IEEE Global Telecommunications Conference, pp. 1-5, **2008**.

[10] M. Akgün. M. U. Çaglayan, "Server Impersonation Attacks and Revisions to SLAP, RFID Lightweight Mutual Authentication Protocol," In International Conference on Systems and Networks Communications, pp.148-153, **2010**.

[11] Y. Glouche, T. Genet, O. Heen, O. Courtay, "A Security Protocol Animator Tool for AVISPA," In ARTIST-2 Workshop on Security Of Embedded Systems, Pisa (Italy), **2006**.

[12] D. A. Basin, S. Modersheim, L. Vigan, "OFMC: A Symbolic Model Checker for Security Protocols," In International Journal of Information Security, vol. 4, no. 3, pp.181-208, **2005**.

[13] M. Turuani, "The Cl-Atse Protocol Analyser," In International Conference on Term Rewriting and Applications- RTA, vol. 4098, pp. 277-286, **2006**.

[14] A. Armando, L. Compagna, "An Optimized Intruder Model for SAT-Based Model-Checking of Security Protocols," Electronic Notes in Theoretical Computer Science, vol. 125, no. 1. pp. 91-108, **2005**.

[15] Y. Boichut, P-C. Heam, O. Kouchnarenko, "Automatic Approximation for the Verification of Cryptographic Protocols," In Proc. on Automated Verification of Infinite-State Systems (AVIS'2004), joint to ETAPS'04, Barcelona (Spain), **2004**.

[16] T. Genet, F. Klay, "Rewriting for Cryptographic Protocol Verification," In Proc. Conf. on Automated Deduction, LNCS 1831, pp. 271-290, **2000**.

[17] P. Peris-Lopez, J. Hernandez-Castro, J. M. Estevez-Tapiador, A. Ribagorda, A. "M2AP: A Minimalist Mutual-Authentication Protocol for Low-Cost RFID Tags," Ubiquitous Intelligence and Computing, pp. 912-923, **2006**.

[18] J. Walker, "ENT A Pseudorandom Number Sequence test program," **2008**. http://www.fourmilab.ch/random.

[19] G. Marsaglia, "Diehard Program and Associated Documentation," **1995**. http://stat.fsu.edu/pub/diehard.

[20] W. Fleming, D. Westfall, I. De La Lande, L. Jellett, "Log-Normal Distribution Of Equieffective Doses Of Noreinephrine And Acetylcholine In Servel Tissues," Journal Of Pharmacology And Experimental Therapeutics, vol. 188, no. 339, **1972**.

[21] J. Bland, D. Altman, "Statistics Notes: Measurement Error and Correlation Coefficients," British Medical Journal (BMJ), vol.313, no.41, **1996**.

[22] S. Roman, Coding and Information Theory. Springer, **1992**.

## Authors

**Rima Hussin Embrak Alakrut** received her B.Sc. degree in Computer Science from the Seventh of April University from the city of Zawyia. Libya. She received her M.Sc. degree in Computer Science from Universiti Sains Malaysia (USM) in 2011. Her research interest is in the field of Computer Network.

**Azman Samsudin** is a lecturer at the School of Computer Sciences, Universiti Sains Malaysia (USM). He received the B.Sc. degree in Computer Science from University of Rochester, USA, in 1989. He obtained his M.Sc. and Ph.D. degrees in Computer Science from University of Denver, USA, in 1993 and 1998, respectively. His current research interests are in the fields of Cryptography, Interconnection Switching Networks, and Parallel Distributed Computing.

**Alfin Syafalni** received the B.Sc. degree in Computer Science from Universiti Sains Malaysia in 2010 and currently pursuing a M.Sc. degree at the same university. In 2009, he underwent the internship program with the Malaysian Institute of Microelectronics Systems (MIMOS) working on IMS implementation and its application. His research interests are on Networking, Multimedia, and Information Security.