Scalability and Security Conflict for RFID Authentication Protocols

Imran Erguler[1,2], Emin Anarim[2]

[1] National Research Institute of Electronics and Cryptology, TUBITAK-UEKAE PO Box 74, 41470, Gebze, Kocaeli, Turkey
[2] Electrical-Electronics Engineering Department, Bogazici University, 34342 Bebek, Istanbul, Turkey
ierguler@uekae.tubitak.gov.tr, anarim@boun.edu.tr

Abstract. Many RFID authentication protocols have been proposed to preserve security and privacy. Nevertheless, most of these protocols are analyzed and it is shown that they can not provide security against some RFID attacks. Moreover, some of the secure ones are criticized, because they suffer from scalability at the reader/server side as in tag identification or authentication phase they require a linear search depending on number of tags in the system. Recently, new authentication protocols have been presented to solve scalability issue, i.e. they require constant time for tag identification with providing security. In this paper, we analyze two of these new RFID authentication protocols SSM (very recently proposed by Song and Mitchell) and LRMAP (proposed by Ha et al.) and to the best of our knowledge, they have received no attacks yet. These schemes take $O(1)$ work to authenticate a tag and are designed to meet the privacy and security requirements. The common point of these protocols is that normal and abnormal states are defined for tags. In the normal state, server authenticates the tag in constant time, while in the abnormal state, occurs rarely, authentication is realized with linear search. We show that, however, these authentication protocols do not provide untraceability which is one of their design objectives. We also discover that the SSM protocol is vulnerable to a desynchronization attack, that prevents a legitimate reader/server from authenticating a legitimate tag. Furthermore, in the light of these attacks, we conclude that allowing tags to be in different states may give clue to an adversary in tracing the tags, although such a design is preferred to achieve scalability and efficiency at the server side.

1 Introduction

RFID systems are expected to be the main communication devices in ubiquitous environment with many applications in manufacturing, supply chain management and inventory control. Because of low production costs and small size, RFID technology is envisioned as a replacement for traditional identification methods such as bar codes. A typical RFID system has three components: tags, one or more readers, and a backend server.

Since memory and computation power of a low-cost RFID tag is limited, it is not feasible to implement computationally intensive cryptographic algorithms. It is an interesting task to design authentication protocols for low-cost RFID tags to resist all possible attacks and threats with obeying RFID implementation constraints. Solving this delicate task has recently aroused interest of security community and many authentication protocols have been proposed for RFID security. A considerable part of the research has provided solutions to the anonymous authentication problem in RFID. However, currently available solutions either do not provide security and privacy [1–10] or suffer from scalability issues as the number of tags in the system is very large [5, 11]. The main reason leading to scalability and security conflict is the hardware constraints on RFID tags, which has so far limited implementation of cryptography in tags to symmetric-key algorithms. The symmetric-key approaches with the anonymous setting result in the difficulty that the server must first decide which secret should be used to identify/authenticate the tag. As a consequence, the server must perform a brute force search in its database to identify a tag. That is, for each tag entry in the database, it computes a symmetric cryptographic operation with the corresponding tag's secret and check whether or not the result matches with the received result produced by the present tag. Such a tedious search procedure will cause scalability issues as the tag population increases.

Recently, new RFID protocols have been proposed to reduce computational load on the back-end database, i.e. solving scalability problem with claiming that they provide security requirements [9, 10, 12, 13]. Usually, these protocols use look-up tables to find the match in the search process, so they need only $O(1)$ effort to identify a tag.

In this study, we investigate security performance of two new protocols which have received no attacks yet. The first protocol has been very recently proposed by Song and Mitchell [12] denoted as SSM (Scalable Song Mitchell protocol) and the second protocol, LRMAP, has been presented by Ha et al. [13]. The common point of these two protocols is that more than one state is defined for RFID tags, such as the state for regular cases and the state for irregular cases respectively. In the regular state, server authenticates the tag in constant time, while in the irregular state authentication is realized with classic exhaustive search.

For both protocols it has been shown that they guarante untraceability, authentication, and robustness against replay and spoofing attacks.

In this paper, we present different attacks to show that the two protocols do not achieve their design objective of untraceability. Our attacks mainly benefit from the fact that RFID tags are allowed to be in different states according to the protocol descriptions which leads to a hint in distinguishing the tags from the perspective of an adversary. Besides, we point out that the SSM scheme has a weakness in the secret update procedure and describe a desynchronization attack on it. The rest of this paper is organized as follows. Section 2 reviews the SSM and the LRMAP protocols. In Section 3 we describe the attacks for each scheme, which is followed by our conclusions in Section 4.

2 Review of the Protocols

In this section, we briefly review the protocols SSM and LRMAP. We use the following notations to simplify the descriptions.

$\mathcal{T}$	RFID tag or transponder
$\mathcal{R}$	RFID reader or transceiver
$\mathcal{DB}$	The back-end database
ID	Identity of a tag, L bits
HID	Hashed value of ID, L bits
PID	Previous identity of a tag used in previous session, L bits
r_R	Random nonce generated by reader $\mathcal{R}$
r_T	Random nonce generated by tag $\mathcal{T}$
$SYNC$	State of $\mathcal{T}$
$H()$	One-way hash function
$e(), f(), g()$	Keyed one-way hash functions of SSM
$SecReq$	Secret update request message
$L(x)$	Left half of input message x
$R(x)$	Right half of input message x
$\|\|$	Concatenation operator
N	Number of tags

2.1 The SSM Protocol

In 2009, Song and Mitchell proposed a scalable RFID authentication protocol [12] to outcome the scalability problem mentioned in the previous section. For this model, $\mathcal{DB}$ stands for the back-end server and the reader. Initially, secret s_i is a string of l bits assigned to $\mathcal{T}_i$ and $k_i = H(s_i)$ is computed by the server. In addition, $\mathcal{DB}$ chooses a random l-bit string

x_0, and computes the hash-chain values $x_i = e_k(x_{i-1})$ for $1 \leq i \leq m$, where the values x_i are used as tag identifiers and m is the length of the hash-chain. In this scheme, $\mathcal{DB}$ stores secrets s_i-k_i for each tag $\mathcal{T}_i$ as well as the most recent secrets $\hat{s}_i$-$\hat{k}_i$ and the identifiers $x_0, x_1, \cdots, x_m$ as the entries for each tag in its look-up table. On the other hand, each $\mathcal{T}_i$ stores k, x and x_m, where x is initially set to x_0. A step by step description of the SSM is given below:

- $\mathcal{DB}$ generates a random l_r-bit string r_R, and sends it to $\mathcal{T}$.
- When $\mathcal{T}$ receives r_R, it compares its stored values of x and x_m.
  - If $x \neq x_m$, then $\mathcal{T}$ calculates $M_T = f_k(r_R||x)$ and updates its identifier x to $e_k(x)$. $\mathcal{T}$ transmits r_R, x and M_T to $\mathcal{DB}$. If the updated x is equal to x_m, $\mathcal{T}$ waits for $\mathcal{DB}$ response, keeping r_R and M_T in short term memory.
  - If $x = x_m$, $\mathcal{T}$ generates a random number r_T and computes $M_1 = f_k(r_R||r_T)$ and $M_2 = r_T \oplus x$. Then, the tag sends r_R, M_1 and M_2 back to $\mathcal{DB}$ with a request for an update of the shared secrets as *SecReq*. $\mathcal{T}$ waits for the server response, keeping r_R, r_T and M_1 in its short term memory.
- When $\mathcal{DB}$ receives the messages one of the following cases is performed:
  - If $\mathcal{DB}$ receives x and M_T , it executes the following steps: Firstly, $\mathcal{DB}$ searches its look-up table for a value x_i equal to the received value of x. If such a value is found, it identifies the tag. Otherwise, the session terminates. Next, $\mathcal{DB}$ checks that $f_k(r_R||x_{i-1})$ equals the received value of M_T , where k is the key belonging to the identified tag $\mathcal{T}$. If this verification succeeds, then $\mathcal{DB}$ authenticates $\mathcal{T}$. Otherwise, the session terminates.
    - ∗ If $x \neq x_m$, then the authentication session terminates successfully.
    - ∗ However, if $x = x_m$, then the server starts a regular secret update process. Firstly, $\mathcal{DB}$ chooses a random l-bit string s' and an integer m', and computes a key $k' = H(s')$ and a sequence of m' identifiers $x'_i = e_{k'}(x'_{i-1})$ for $1 \leq i \leq m'$, where x'_0 is set to x. Then, $\mathcal{DB}$ computes $M_S = g_k(r_R||x||M_T) \oplus (s||k'||x'_{m'})$, and sends r_R and M_S to $\mathcal{T}$. Finally, $\mathcal{DB}$ updates the set of stored values for $\mathcal{T}$ from $(\hat{s}, \hat{k}, s, k, x_0, x_1, \cdots, x_m)$ to $(s, k, s', k', x, x'_1, \cdots, x'_{m'})$.
  - If $\mathcal{DB}$ receives r_R, M_1, M_2 and *SecReq* from $\mathcal{DB}$, then the server starts an irregular secret update process. Firstly, $\mathcal{DB}$ searches its look-up table for a value $x = x_m$ or $x = x_0$ for which $M_1 =$

$f_k(r_R||(M_2 \oplus x))$. If such a value is found, $\mathcal{DB}$ authenticates the tag. Otherwise, the session terminates.

* If $x = x_m$, it means that although $\mathcal{T}$ sent $x = x_m$ to $\mathcal{DB}$ in the previous session, $\mathcal{DB}$ did not receive it correctly. Hence, neither server nor tag have updated their shared secrets. In this case, $\mathcal{DB}$ performs the following steps. $\mathcal{DB}$ chooses a random l-bit string s' and an integer m', and computes a key $k' = H(s')$ and a sequence of m' identifiers $x'_i = e_{k'}(x'_{i-1})$ for $1 \le i \le m'$, where x'_0 is set to x. After $\mathcal{DB}$ computes $r_T = M_2 \oplus x$ and $M_S = g_k(r_R||r_T||M_1) \oplus (s||k'||x'_{m'})$, and sends r_R and M_S to $\mathcal{T}$. Last, $\mathcal{DB}$ updates the set of stored values for $\mathcal{T}$ from $(\hat{s}, \hat{k}, s, k, x_0, x_1, \cdots, x_m)$ to $(s, k, s', k', x, x'_1, \cdots, x'_{m'})$.
* If $x = x_0$, it means that M_S did not reach $\mathcal{T}$ correctly in the previous session, and thus $\mathcal{T}$ did not update its secrets, although $\mathcal{DB}$ did. In this case, the following steps are done by the server. $\mathcal{DB}$ computes $r_T = M_2 \oplus x$ and $M_S = g_k(r_R||r_T||M_1) \oplus (\hat{s}||k||x_m)$, and sends r_R and M_S to $\mathcal{T}$

– If $\mathcal{T}$ receives r_R and M_S, it calculates $(s||k'||x'_{m'})$ as the following: If M_1, M_2 and *SecReq* is sent by $\mathcal{T}$ in the first step, then $(s||k'||x'_{m'}) = M_S \oplus g_k(r_R||r_T||M_1)$, however if M_T is sent in the first step then $(s||k'||x'_{m'}) = M_S \oplus g_k(r_R||x||M_T)$ is evaluated. Next, if $H(s)$ is equal to k, $\mathcal{T}$ authenticates $\mathcal{DB}$ and updates k and x_m to k' and $x'_{m'}$, respectively. The secret update session then terminates successfully. Otherwise, the session is ended.

The protocol is shown in Figure 1.

2.2 The LRMAP

LRMAP is a mutual authentication protocol proposed by Ha et al. [13]. The major advantage of the scheme compared to other protocols is that the LRMAP reduces the computational load on the back-end database by putting the tags in two different states. That is, if the tag is synchronized state the protocol only requires 3 hash operations in the database even number of tags in the system is large. In the case of desynchronization, the recovery time in desynchronization state is $N + 3$ hash operations on average, where N denotes the number of tags. However, since desynchronization of a tag is a special and unusual state, the normal synchronization state only requires 3 hash operations.

Data Base / Reader $[\hat{s}, \hat{k}, s, k, x_0, \cdots, x_i \cdots, x_m]$		**Tag** $[k, x, x_m]$
Generate r_R	r_R $\longrightarrow$	**If** $x \neq x_m$, $M_T = f_k(r_R \vert\vert x)$ $x \leftarrow e_k(x)$
	$\{r_R, x, M_T\}$ $\longleftarrow$	
Case 1: Search for $x_i = x$ in the DB Check $M_T = f_k(r_R \vert\vert x_{i-1})$ **Case 2:** **If** $x = x_m$, $M_S = g_k(r_R \vert\vert x \vert\vert M_T) \oplus (s \vert\vert k' \vert\vert x'_{m'})$ Update secrets for Tag $\hat{s} \leftarrow s, \hat{k} \leftarrow k, s \leftarrow s', k \leftarrow k', x_0 \leftarrow x$ $x_i(1 \leq i \leq m) \leftarrow x'_i(1 \leq i \leq m')$	$\{r_R, M_S\}$ $\longrightarrow$	$(s \vert\vert k' \vert\vert x'_{m'}) = M_S \oplus g_k(r_R \vert\vert x \vert\vert M_T)$ **If** $H(s) = k$, $k \leftarrow k', x_m \leftarrow x'_{m'}$
		If $x = x_m$, Generate r_T $M_1 = f_k(r_R \vert\vert r_T)$ $M_2 = r_T \oplus x$
Case 3: Search for $x = x_m$ or x_0 in the ***DB*** for which $M_1 = f_k(r_R \vert\vert (M_2 \oplus x))$ $r_T = M_2 \oplus x$ **If** $x = x_m$, $M_S = g_k(r_R \vert\vert r_T \vert\vert M_1) \oplus (s \vert\vert k' \vert\vert x'_{m'})$ **If** $x = x_0$, $M_S = g_k(r_R \vert\vert r_T \vert\vert M_1) \oplus (\hat{s} \vert\vert k \vert\vert x_m)$	$\{r_R, M_1, M_2, SecReq\}$ $\longleftarrow$	
Update secrets for Tag $s \leftarrow s', k \leftarrow k', x_0 \leftarrow x$ $x_i(1 \leq i \leq m) \leftarrow x'_i(1 \leq i \leq m')$	$\{r_R, M_S\}$ $\longrightarrow$	$(s \vert\vert k' \vert\vert x'_{m'}) = M_S \oplus g_k(r_R \vert\vert r_T \vert\vert M_1)$ **If** $H(s) = k$, $k \leftarrow k', x_m \leftarrow x'_{m'}$

Fig. 1. The SSM Protocol

The back-end database $\mathcal{DB}$ manages the ID, hashed values HID, and PID for each $\mathcal{T}$ in the database field. According to the state of the tag, the $\mathcal{DB}$ finds the ID for the current session or PID used for the previous session by comparing the received message with the HID and PID.

A step by step description of the LRMAP is given below:

- $\mathcal{R}$ challenges $\mathcal{T}$ with a random nonce r_R.
- $\mathcal{T}$ chooses a random nonce r_T and computes P differently according to the state of $SYNC$. If $SYNC = 0$, then $P = H(ID)$, otherwise $P = H(ID||r_T)$. Next computes $Q = H(ID||r_T||r_R)$ and sets $SYNC = 1$. $\mathcal{T}$ responds with $\{P, L(Q), r_T\}$.
- $\mathcal{R}$ delivers the messages from $\mathcal{T}$ to $\mathcal{DB}$ with r_R.
- $\mathcal{DB}$ firstly searches P with the HID values stored in the database. If the values match, $\mathcal{DB}$ regards the ID as the identity of $\mathcal{T}$. This is a general case when the previous session is closed normally. If $\mathcal{DB}$ cannot find any match in the first searching case, then computes $H(ID||r_T)$ compares it with P. However, if $\mathcal{DB}$ still cannot find the ID of $\mathcal{T}$ in the second search, it then computes $H(PID||r_T)$ and compares it with P. $\mathcal{DB}$ gets a match in the third search process when $\mathcal{R}$'s last messages were blocked in the previous session, that is, S$YNC = 1$ and $\mathcal{DB}$ updated the ID, but ID of the tag was not updated. If $\mathcal{DB}$ finds a match in any of the three searching cases, PID is updated to ID for the first or second search or set to PID for the third search. Then it calculates $Q' = H(PID||r_T||r_R)$ and checks whether or not $L(Q')$ is equal to $L(Q)$. If it holds, $\mathcal{DB}$ sends $R(Q')$ to $\mathcal{R}$ and sets $ID = H(PID||r_R)$ and $HID = H(ID)$.
- $\mathcal{R}$ forwards received $R(Q')$ to $\mathcal{T}$.
- If $R(Q') = R(Q)$, then $\mathcal{T}$ updates its ID as $ID = H(ID||r_R)$ and sets $SYNC = 0$.

The protocol is depicted in Figure 2.

3 The Attacks

In this section, we present concrete attacks to both SSM and LRMAP protocols. In our attack models, it is assumed that the adversary can eavesdrop, block, modify, and inject messages in any communication between a reader and a tag. We apply tracking attacks on these protocols and show that they do not achieve their design objective of untraceability. Additionally, we describe a denial of service (DoS) attack for SSM such that an adversary can permanently desynchronize the interactions between a server and a tag.

Data Base / Reader $[ID, HID, PID]$		**Tag** $[ID, SYNC]$
Generate r_R	r_R $\longrightarrow$	Generate r_T **If** $SYNC = 0$, $P = H(ID)$ **else** $P = H(ID\|\|r_T)$ $Q = H(ID\|\|r_T\|\|r_R), SYNC = 1$
	$\{P, L(Q), r_T\}$ $\longleftarrow$	
If $P = H(ID)$, $PID = ID$ **else if** $P = H(ID\|\|r_T)$, $PID = ID$ **else if** $P = H(PID\|\|r_T)$, $PID = PID$ $Q' = H(PID\|\|r_T\|\|r_R)$ **If** $L(Q') = L(Q)$, $ID = H(PID, r_R)$, $HID = H(ID)$		
	$R(Q')$ $\longrightarrow$	**If** $R(Q') = R(Q)$ $ID = H(ID\|\|r_R)$, $SYNC = 0$

Fig. 2. The LRMAP Protocol

3.1 Tracking Attack for SSM

Intuitively, a protocol satisfies untraceability if an adversary is not able to recognize a tag he has previously observed [14]. Issue of untraceability is treated formally in security models, such as by Avoine [15] and Juels-Weis [16]. According to [16] as a formal definition, untraceability can be defined in terms of privacy experiments. The aim of the adversary in this experiment is to distinguish between two different tags within the limits of its computational power and functionality-call bounds. Instead of reproducing the detailed untraceability model definitions of the [16], we use the terms given in [14]. The privacy experiment consists of two phases: The learning phase and the challenge phase. In the former, the adversary $\mathcal{A}$ may initiate a communication with the reader $\mathcal{R}$ (ReaderInit) or tags $\mathcal{T}$ (TagInit). Then he may interact with them according to the corresponding protocol steps. In the challenge phase, the adversary selects two tag candidates $\mathcal{T}_i$ and $\mathcal{T}_j$ to be tested. Then he chooses one of these tags randomly, called $\mathcal{T}^*$ and $\mathcal{A}$ is given access to this tag. The adversary may again interact with the reader and the tags. Eventually, $\mathcal{A}$ terminates the test and decides whether the selected tag is $\mathcal{T}_i$ or $\mathcal{T}_j$. If the adversary has a non-negligible advantage in successfully guessing the selected tag, then he succeeds in attack and the protocol is not untraceable.

According to the SSM protocol specification: If $x \neq x_m$ at the tag side, then $\mathcal{T}$ transmits r_R, x and M_T to $\mathcal{DB}$. However, if $x = x_m$, $\mathcal{T}$ responds to the reader with r_R, M_1, M_2 and *SecReq*. Thus, response types of the tag gives a hint about the state of a tag and this is the main idea behind the attack.

It is very likely that for most of the tags $x \neq x_m$, because this corresponds regular case. Nevertheless, $x \neq x_m$ for a tag can be assured by executing the *Algorithm Tag Reset* given below. In this algorithm, an adversary queries a tag until observing the event that the tag has $x = x_m$ and responds with r_R, M_1, M_2 and *SecReq* messages. Then, a successful authentication and irregular secret update process between the server and the tag is observed. After these steps the adversary is sure that in the next query of the tag it will not wait for a secret update, because $m \geq 1$ i.e. $x \neq x_m$.

Algorithm Tag Reset

- $\mathcal{A}$ *initiates communication with* $\mathcal{T}_i$ *using* TagInit.
- $\mathcal{A}$ *transmits some random nonce* r_A *to* $\mathcal{T}_i$.
- $\mathcal{A}$ *repeats the above two steps till* $\mathcal{T}_i$ *responds with SecReq*.
- $\mathcal{A}$ *initiates communication with* $\mathcal{DB}$ *using* ReaderInit *and gets* r_R.
- $\mathcal{A}$ *initiates communication with* $\mathcal{T}_i$ *using* TagInit.
- $\mathcal{A}$ *transmits* r_R *to* $\mathcal{T}_i$.
- $\mathcal{A}$ *delivers* $\mathcal{T}_i$ *response* $\{r_R, M_1, M_2, SecReq\}$ *to* $\mathcal{DB}$.
- $\mathcal{A}$ *transmits* $\mathcal{DB}$ *response* $\{r_R, M_S\}$ *to* $\mathcal{T}_i$.

For the privacy experiments, the adversary follows the attack as described below. We suppose for the selected tags $x \neq x_m$ (This can be realized easily by running the *Algorithm Tag Reset* as mentioned above). Thus, in this case for the first query of the tags, they will not request a secret update from the server. In the learning phase of the attack, two tags $\mathcal{T}_i$ and $\mathcal{T}_j$ are selected and tag $\mathcal{T}_i$ is put into state such that $x = x_m$. This can be done by simultaneously querying $\mathcal{T}_i$ until observing the responses $r_R, M_1, M_2, SecReq$.

Learning Phase

- $\mathcal{A}$ *randomly chooses a pair of distinct tags* $\mathcal{T}_i$ *and* $\mathcal{T}_j$.
- $\mathcal{A}$ *initiates communication with* $\mathcal{T}_i$ *using* TagInit.

- $\mathcal{A}$ *transmits some random nonce* r_A *to* $\mathcal{T}_i$.
- $\mathcal{A}$ *repeats the previous two steps till* $\mathcal{T}_i$ *responds with SecReq.*

In the challenge phase, the adversary only the queries the selected tag once and observes the response. Then he checks whether the tag response includes *SecReq* message or not: If the selected tag is $\mathcal{T}_i$, then it answers the query with r_R, M_1, M_2 and *SecReq* messages, since $x = x_m$ for $\mathcal{T}_i$. On the other hand if the selected tag is $\mathcal{T}_j$, then it responds with r_R, x and M_T due to $x \neq x_m$.

Challenge Phase

- $\mathcal{A}$ *takes* $\mathcal{T}_i$ *and* $\mathcal{T}_j$ *as its challenge candidates.*
- $\mathcal{A}$ *transmits some random nonce* r_A *to the selected tag* T^*.
- $\mathcal{A}$ *observes* $\mathcal{T}^*$ *response.*
- *If* $\mathcal{T}^*$ *response includes SecReq message,* $\mathcal{A}$ *decides* $\mathcal{T}^* = \mathcal{T}_i$. *Otherwise guesses* $\mathcal{T}^* = \mathcal{T}_j$.

3.2 Denial of Service Attack for SSM

In this part, it is shown that the SSM protocol is vulnerable to a denial of service attack, in which an adversary can update a tag's secret to a random value. Consequently, the tag is desynchronized with the server and authentication of the tag is prevented. Our attack can be launched to any tag that is in state $x = x_m$ and requesting secret update. Note that an attacker can easily put any tags to this state by running the *Learning Phase* of the previous attack. As the $\mathcal{DB}$ responds with $\{r_R, M_S\}$ pair to secret update request of $\mathcal{T}$, an active adversary can intercept and block the message from reaching the tag. Then he forges a second message $\{r_R, \hat{M}_S\}$ such that $\hat{M}_S = M_S \oplus (0||r_1||r_2)$ and r_1, r_2 are l-bit strings other than zero. Next, the adversary sends the modified value, $\{r_R, \hat{M}_S\}$, to the tag. $\mathcal{T}$ firstly computes $\hat{M}_S \oplus g_k(r_R||r_T||M_1)$ and obtains $(s||\hat{k}||\hat{x}_m)$, where $\hat{k} = k' \oplus r_1$ and $\hat{x}_m = x'_{m'} \oplus r_2$. The modified secrets are accepted by the tag, because $H(s) = k$ is verified. Thus, at the end of the attack execution, $\mathcal{DB}$ and $\mathcal{T}$ update their secrets to different values. The server stores $(s, k, s', k', x, x'_1, \cdots, x'_{m'})$, while the tag stores $\hat{k}$ and $\hat{x}_m$. In the next query of the tag, it sends x and M_T to the server, where $x = e_{\tilde{k}}(x)$ and $M_T = f_{\hat{k}}(r_R||x)$. When $\mathcal{DB}$ receives x, it will not find a match in the look-up table such that a value x_i equal to the received value of x. Furthermore, $\hat{x}_m$ is not in the keyed hash chain list of $e_{\hat{k}}(x)$, so the tag will not request a secure update in the future queries neither. Therefore, the reader and tag will be in a desynchronized state and future authentication of the tag becomes impossible. The steps of this desynchronization attack are given below:

DoS Attack

- $\mathcal{A}$ *takes* $\mathcal{T}_i$ *as its target tag.*
- $\mathcal{A}$ *initiates communication with* $\mathcal{T}_i$ *using* TagInit*.*
- $\mathcal{A}$ *transmits some random nonce* r_A *to* $\mathcal{T}_i$*.*
- $\mathcal{A}$ *repeats the previous two steps till* $\mathcal{T}_i$ *responds with SecReq.*
- $\mathcal{A}$ *initiates communication with* $\mathcal{DB}$ *using* ReaderInit *and gets* r_R*.*
- $\mathcal{A}$ *initiates communication with* $\mathcal{T}_i$ *using* TagInit*.*
- $\mathcal{A}$ *transmits* r_R *to* $\mathcal{T}_i$*.*
- $\mathcal{A}$ *delivers* $\mathcal{T}_i$ *response* $\{r_R, M_1, M_2, SecReq\}$ *to* $\mathcal{DB}$*.*
- $\mathcal{A}$ *blocks* $\mathcal{DB}$ *response* $\{r_R, M_S\}$ *from reaching to* $\mathcal{T}_i$*.*
- $\mathcal{A}$ *forges a second message* $\{r_R, \hat{M}_S\} : \hat{M}_S = M_S \oplus (0||r_1||r_2)$*.*
- $\mathcal{A}$ *sends the modified value,* $\{r_R, \hat{M}_S\}$*, to* $\mathcal{T}_i$*.*
- $\mathcal{T}_i$ *computes* $\hat{M}_S \oplus g_k(r_R||r_T||M_1)$ *and gets* $(s||\hat{k}||\hat{x}_m)$*, where* $\hat{k} = k' \oplus r_1$ *and* $\hat{x}_m = x'_{m'} \oplus r_2$*.*
- $\mathcal{T}_i$ *verifies* $H(s) = k$ *and updates the secrets as* $\hat{k}$ *and* $\hat{x}_m$*, while they are stored as* k' *and* $x'_{m'}$ *at* $\mathcal{DB}$*. Thus* $\mathcal{DB}$ *will not able to identify or authenticate* $\mathcal{T}_i$ *in the future sessions*

3.3 Timing Attack for LRMAP

According to the LRMAP protocol specification, if $SYNC$ of the tag is 0, the tag sets $P = H(ID)$, otherwise computes $P = H(ID||r_T)$. Then it responds to the reader with $\{P, L(Q), r_T\}$. On the $\mathcal{DB}/\mathcal{R}$ side, $\mathcal{DB}$ firstly compares the received P with the HID values stored in the database. If a match exists, the $\mathcal{DB}$ regards the ID as the identity of the tag and concludes $SYNC = 0$ for the present tag. This is the case when the previous session is closed normally. This process requires one table-lookup and reader returns in constant time. On the other hand, if the $\mathcal{DB}$ cannot find the HID in the first searching case, then it computes $H(ID_i||r_T)$ value for $1 \leq i \leq N$ until a match is found between P and the computed value. However, if the $\mathcal{DB}$ still cannot find the ID of tag in above two cases, then it computes $H(PID_i||r_T)$ value for $1 \leq i \leq N$ and compares it with P till the match is found. After a match is obtained in any of three cases, the reader responds with $R(Q')$. Notice that if $SYNC = 0$, then $\mathcal{R}$ returns in constant time since first search only requires one table lookup. However if $SYNC = 1$, $\mathcal{DB}$ makes about $N/2$ hash operations for the second search and $N+N/2$ hash operations for the third search. Thus, an adversary can potentially distinguish between tags with $SYNC=0$ and tags with $SYNC=1$ by timing server responses. A tag with $SYNC=0$ only requires a server to perform a fast table look-up, whereas a tag with $SYNC=1$ requires it to perform an exhaustive search. In fact, this is the main idea behind our attack.

Assume first search costs τ_1 time and each hash operation requires τ_2 time. Let t_1, t_2 and t_3 represent the average elapsed time between the second and third message flow for the cases; the single search, two searches (1st, 2nd search) and three searches (1st, 2nd, 3rd search) respectively. Also suppose θ stands for other time costs such as time loss due to communication layer etc. From this fact, the response time of the reader on average can be defined as:

1- If $SYNC=0$, only the first search is done in $\mathcal{DB}$, so $t_1=\tau_1+\tau_2+\theta$.

2- If $SYNC=1$ and match is in 2nd search, $t_2=\tau_1+(\frac{N}{2}+1)\cdot\tau_2+\theta$.

3- If $SYNC=1$ and match is in 3rd search, $t_3=\tau_1+(\frac{3N}{2}+1)\cdot\tau_2+\theta$.

The attack may have a training phase to estimate values of t_1, t_2 and t_3. For example, Table 1 gives some practical values for t_1, t_2 and t_3. In this example, it is assumed that the system relies on a single computer which takes 2^{-23} seconds to carry out a hash operation and the number of tags in the system is 2^{20}. According to the these values, one can see that there is a dramatic difference between t_1, t_2 and t_3.

It is very likely that most of the tags have $SYNC=0$. In fact, the protocol is designed with the assumption that most of the tags will be in this state. Hence, the adversary can estimate the approximate value of t_1, by observing successful authenticated protocols between the reader and tags and noting the time intervals that the reader respond in the third message flow. Also, to approximate t_2, the adversary firstly puts a tag into $SYNC=1$. He can realize this by challenging the tag and ending the protocol before sending the third message. Then the adversary observes a successful authentication with the reader by only considering time responses. Lastly, the adversary observes a successful authentication with the reader, to estimate t_3, but now prevents the third flow from reaching the tag. It puts the tag out of the synchronization and $SYNC=1$, because while the DB updates the ID, the tag does not. Next, the adversary allows the reader and the tag to run the protocol again without intervening them and records the elapsed time for the reader response. Note that, for this case, the reader realizes the third search, since the tag is desynchronized from the previous action. Thus, the adversary obtains information about t_3. The adversary may repeat the above steps for some number of times to get expected values of t_1, t_2 and t_3.

Table 1. Practical values for t_1, t_2 and t_3. It is assumed that $N = 2^{20}$ and the server has capability make one hash operation in 2^{-23} seconds. Since τ_1 and θ are common for all of three parameters, they are skipped in calculation .

Parameter	Time (millisecond)
t_1	0.0001
t_2	62.5
t_3	187.5

For the privacy experiments, the adversary can follow the attack as follows. In the learning phase, two tags $\mathcal{T}_i$ and $\mathcal{T}_j$ are selected and tag $\mathcal{T}_i$ is put into state $SYNC = 1$. This can be done as mentioned above.

Learning Phase

- $\mathcal{A}$ *randomly chooses a pair of distinct tags* $\mathcal{T}_i$ *and* $\mathcal{T}_j$.
- $\mathcal{A}$ *initiates communication with* $\mathcal{T}_i$ *using* TagInit.
- $\mathcal{A}$ *transmits some random nonce* r_A *to* $\mathcal{T}_i$.
- $\mathcal{A}$ *terminates the protocol.*

In the challenge phase, the adversary only observes a successful authentication between the legitimate reader and the tag and records time duration between the second and the third message flow, call it t'. If $t' \approx t_2$, the tag's $SYNC = 1$, hence the selected tag is $\mathcal{T}_i$. On the other hand, if $t' \approx t_1$ then the tag's $SYNC = 0$, hence the selected tag is $\mathcal{T}_j$.

Challenge Phase

- $\mathcal{A}$ *takes* $\mathcal{T}_i$ *and* $\mathcal{T}_j$ *as its challenge candidates.*
- $\mathcal{A}$ *initiates communication with* $\mathcal{R}$ *using* ReaderInit *and gets* r_R.
- $\mathcal{A}$ *transmits* r_R *to the selected tag* $\mathcal{T}^*$.
- $\mathcal{A}$ *delivers* $\mathcal{T}^*$ *response* $\{P, L(Q), r_T\}$ *to* $\mathcal{R}$.
- $\mathcal{A}$ *measures elapsed time,* t'*, between* 2nd *and* 3rd *message flow.*
- *If* $t' \approx t_2$ *and* $t' \gg t_1$, $\mathcal{A}$ *decides* $\mathcal{T}^* = \mathcal{T}_i$. *Otherwise guesses* $\mathcal{T}^* = \mathcal{T}_j$.

Remark. A similar attack can be also applied on SSM protocol. An adversary $\mathcal{A}$ can easily put a tag $\mathcal{T}$ into desynchronized state by executing the Learning Phase in Section 3.1. Then $\mathcal{A}$ can distinguish between synchronized and desynchronized tags by timing server responses, because a synchronized tag only requires a server to perform a fast table look-up, whereas a desynchronized tag requires it to perform an exhaustive search.

3.4 Timing Attack II for LRMAP

In this part, we enhance the previous attack and use it to distinguish among three tags by putting them into three different states. In the learning phase, three tags $\mathcal{T}_i$, $\mathcal{T}_j$ and $\mathcal{T}_k$ are selected. $\mathcal{T}_i$ is put into state $SYNC = 1$ as described in the previous attack and $\mathcal{T}_j$ is put into desynchronized and state $SYNC = 1$, by blocking third message flow in a normal session between the reader and $\mathcal{T}_j$.

Learning Phase

- $\mathcal{A}$ *randomly chooses a pair of distinct tags* $\mathcal{T}_i$, $\mathcal{T}_j$ *and* $\mathcal{T}_k$.
- $\mathcal{A}$ *initiates communication with* $\mathcal{T}_i$ *using* TagInit.
- $\mathcal{A}$ *transmits some random nonce* r_A *to* $\mathcal{T}_i$.
- $\mathcal{A}$ *terminates the protocol.*
- $\mathcal{A}$ *initiates communication with* $\mathcal{R}$ *using* ReaderInit *and gets* r_R.
- $\mathcal{A}$ *initiates communication with* $\mathcal{T}_j$ *using* TagInit.
- $\mathcal{A}$ *transmits* r_R *to* $\mathcal{T}_j$.
- $\mathcal{A}$ *relays* $\mathcal{T}_j$*'s response* $\{P, L(Q), r_T\}$ *to the reader.*
- $\mathcal{A}$ *breaks the protocol.*

In the challenge phase, the adversary only observes a successful authentication between the legitimate reader and the tag and records time duration between the second and the third message flow, call it t'. If $t' \approx t_3$, the tag's $SYNC = 1$ and it is in desynchronized state, hence the selected tag is $\mathcal{T}_j$. On the other hand, if $t' \approx t_2$ then the tag's $SYNC = 1$ but in synchronized, so the selected tag is $\mathcal{T}_i$. Otherwise, if $t' \approx t_1$ then the tag's $SYNC = 0$, hence the selected tag is $\mathcal{T}_k$.

Challenge Phase

- $\mathcal{A}$ *takes* $\mathcal{T}_i$, $\mathcal{T}_j$ *and* $\mathcal{T}_k$ *as its challenge candidates.*
- $\mathcal{A}$ *initiates communication with* $\mathcal{R}$ *using* ReaderInit *and gets* r_R.
- $\mathcal{A}$ *transmits* r_R *to the selected tag* $\mathcal{T}^*$.

– $\mathcal{A}$ *delivers* $\mathcal{T}^*$ *response* $\{P, L(Q), r_T\}$ *to* $\mathcal{R}$.
– $\mathcal{A}$ *measures elapsed time,* t', *between* 2*nd and* 3*rd message flow.*
– *If* $t' \approx t_3$, $\mathcal{A}$ *decides* $\mathcal{T}^* = \mathcal{T}_j$. *On the other hand, if* $t' \approx t_2$, $\mathcal{A}$ *guesses* $\mathcal{T}^* = \mathcal{T}_i$. *Otherwise if* $t' \approx t_1$, $\mathcal{A}$ *decides* $\mathcal{T}^* = \mathcal{T}_k$.

4 Concluding Remarks

Scalability is a desirable property for an RFID protocol such that it should be able to handle large numbers of tags without an undue computational load. To achieve this, new schemes have been proposed with requiring only $O(1)$ effort to identify a tag. These models usually allow the tags to be in different states: For the regular cases, a normal state is defined and in this state server authenticates the tag in constant time. On the other hand, for irregular cases, an abnormal state is defined and in this state the server needs to perform a linear search with complexity $O(N)$. Although allowing the tags to be in different states reduces the computational complexity at the backend server, it gives also a hint to an adversary who aims to distinguish the tags by connecting relations between tag/server responses and tag states. In this study, we analyze two of these protocols as the SSM and the LRMAP protocols. We show that both of the SSM and LRMAP protocols cannot achieve untraceability by describing the tracking and timing attacks that give the adversary a non-negligible advantage of guessing the selected tag. It can be easily shown that similar protocols are vulnerable to the presented attacks or some their modified versions, so this is a security/scalability conflict for RFID authentication protocols.

References

1. Chien H. Y., Chen C. H.: Mutual Authentication Protocol for RFID Conforming to EPC Class 1 Generation 2 Standards, Computers Standards Interfaces, vol. 29, no. 2, pp. 254-259. (2007).
2. Ohkubo M., Suzki K., Kinoshita S.: Cryptographic Approach to Privacy-friendly Tags, In RFID Privacy Workshop, MIT, MA, USA, November 2003. http://www.rfidprivacy.us/2003/agenda.php.
3. Rhee K., Kwak J., Kim S., Won D.: Challenge-Response Based on RFID Authentication Protocol for Distributed Database Environment, In: SPC05. LNCS, vol. 3450. pp. 70−84. Springer, Heidelberg (2005)
4. Duc D., Park J., Lee H., Kim K: Enhancing Security of EPCglobal GEN-2 RFID Tag Against Traceability and Cloning, In: Symposium on Cryptography and Information Security 2006.
5. Song B., Mitchell C.: RFID Authentication Protocol for Low-cost Tags, In WiSec 08: Proceedings of the first ACMConference on Wireless Network Security, pp. 140−147. ACM Press (2008).

6. Dimitriou T.: A Lightweight RFID Protocol to Protect against Traceability and Cloning Attacks, In Conference on Security and Privacy for Emerging Areas in Communication Networks SecureComm 2005, pp. 59−66. IEEE (2005).
7. Henrici A., Muller P.: Hash-based Enhancement of Location Privacy for Radio-frequency Identification Devices Using Varying Identifiers, International Workshop on Pervasive Computing and Communication Security PerSec 2004, pp. 149−153. IEEE Computer Society (2004).
8. Molnar D., Wagner D.: Privacy and Security in Library RFID, Issues, practices, and architectures. In B. Pfitzmann and P. Liu, editors, Conference on Computer and Communications Security ACM CCS, pp. 210− 219, Washington, DC, USA, October 2004. ACM Press.
9. Ha, J., Moon, S.J., Gonzlez Nieto, J.M. and Boyd, C., Low-cost and Strong-security RFID authentication protocol. In: EUC Workshops 2007. LNCS, vol. 4809. pp. 795−807. Springer, Heidelberg (2007)
10. Tsudik G.: A Family of Dunces: Trivial RFID identification and authentication protocols, In N. Borisov and P. Golle, editors, Privacy Enhancing Technologies, 7th International Symposium PET 2007. LNCS, vol. 4776. pp. 45−61. Springer, Heidelberg (2007).
11. Cai S., Li Y., Li T., Deng R: Attacks and Improvements to An RFID Mutual Authentication Protocol and Its Extensions. In: Proceedings of the second ACM-Conference on Wireless Network Security, ACM Press (2009).
12. Song B., Mitchell C.: Scalable RFID Pseudonym Protocol. In: Network System Security 19-21 October 2009, IEEE Computer Society Press, pp. 216 − 224.
13. Ha, J., Ha, J., Moon, S., Boyd, C.: LRMAP: Lightweight and Resynchronous Mutual Authentication Protocol for RFID System. In: Stajano, F., Kim, H.-J., Chae, J.-S., Kim, S.-D. (eds.) ICUCT 2006. LNCS, vol. 4412, pp. 80−89. Springer, Heidelberg (2007).
14. Deursen, T.v., Radomirovic, S.: Security of RFID protocols A Case Study. Electronic Notes in Theoretical Computer Science, vol. 224, pp. 41−52. Elsevier (2009)
15. Avoine, G.: Adversarial Model for Radio Frequency Identificatin. Cryptology ePrint Archieve, Report 2005/049 (2005), http://eprint.iacr.org
16. Juels, A. and S. A. Weis: Defining Strong Privacy for RFID, in: PerCom Workshops, pp. 342−347, (2007).