On the Security of Chien's Ultralightweight RFID Authentication Protocol

Hung-Min Sun, Wei-Chih Ting, and King-Hang Wang

Department of Computer Science,
National Tsing Hua University, Hsinchu, Taiwan
hmsun@cs.nthu.edu.tw
{sd,khwang0}@is.cs.nthu.edu.tw

Abstract. Recently, Chien proposed an ultralightweight RFID authentication protocol to prevent all possible attacks. However, we find two de-synchronization attacks to break the protocol.

Key words: RFID, cryptanalysis, identification protocols

1 Introduction

RFID systems will soon be widely deployed. Currently, they are not secure enough, and hence researchers have proposed various solutions as introduced by Chien in [1]. Chien classified these protocols into four classes. They are *full-fledged*, *simple*, *lightweight*, and *ultralightweight* protocols. The first uses cryptographic functions or public key algorithms to provide mutual authentication between the reader and the tag. The second requires a random number generator and a hash function on each tag. The third uses CRC functions instead of hash functions. The fourth class only needs simple operations, such as XOR, AND, OR, etc. Recently, Chien [1] proposed a new ultralightweight protocol, called SASI, which provides mutual authentication, tag anonymity, data integrity, and forward security. It was designed to resist de-synchronization attack, replay attack, and man-in-the-middle attack. However, we find two de-synchronization attacks to break the protocol.

2 SASI protocol

In this section, we review Chien's protocol. There are three entities in the scheme: tag, reader, and backend database. It is assumed that the reader and the database shares a secure channel, but the channel between the reader and the tag is insecure. The tag is initialized with a static identification (ID), a pseudonym (IDS) which is used as the search index in the database, and two secret keys $K1$ and $K2$. The length of each variable is 96 bits. These variables are also stored in the database.

Let R denote the reader and T denote the tag. The symbol '$\oplus$' refers to bitwise exclusive-or, '+' refers to addition under $mod\ 2^{96}$, and '$\vee$' refers to bitwise-or. $Rot(x, y)$ stands for left rotating x according to y's bits. More precisely, for $y = y_{96}y_{95}y_{94}...y_2y_1$, each input bit y_i, where $1 \leq i \leq 96$, is examined and processed by: if $y_i = 1$, x is left rotated one bit; otherwise, do nothing. (Note that $Rot(x, y)$ is not clearly defined in [1]. We confirm it in [2]). In fact, $Rot(x, y)$ acts as an $w(y)$-bit left rotation on x, where $w(y)$ denotes the Hamming weight of y. The protocol works as follows:

1. $R \rightarrow T : hello$
2. $T \rightarrow R : IDS$

3. The reader uses IDS to find a matched record in the database and access the corresponding secret information ID, $K1$, and $K2$ for the tag. If the IDS is not found in the database, the reader will request the old IDS by Step 2 again.
4. The reader chooses two random numbers $n1, n2$ and sends:
   $R \rightarrow T : A||B||C$, where
   $A = IDS \oplus K1 \oplus n1$,
   $B = (IDS \vee K2) + n2$,
   $\overline{K}1 = Rot(K1 \oplus n2, K1)$,
   $\overline{K}2 = Rot(K2 \oplus n1, K2)$,
   $C = (K1 \oplus \overline{K}2) + (K2 \oplus \overline{K}1)$.
5. T extracts $n1$ and $n2$ from A and B. Then it computes C. If C matches with the one in Step 4, then it updates its IDS, $K1$, and $K2$ as follows:
   (a) $IDS_{old} = IDS; K1_{old} = K1; K2_{old} = K2$,
   (b) $IDS_{next} = (IDS + ID) \oplus (n2 \oplus \overline{K}1)$,
   (c) $K1_{next} = \overline{K}1; K2_{next} = \overline{K}2$.
6. $T \rightarrow R : D$, where
   $D = (\overline{K}2 + ID) \oplus ((K1 \oplus K2) \vee \overline{K}1)$.
7. R computes D. If D matches with the one in Step 6, R updates its IDS, $K1$, and $K2$.

Note that in Step 7, if D passes the verification, the database will update the variables IDS, $K1$, and $K2$ with the value of IDS_{next}, $K1_{next}$, and $K2_{next}$ respectively. The old values of the variables are discarded [2].

3 The First Attack

We assume that there is a synchronized tag in which $(IDS_{next}, K1_{next}, K2_{next})$ equals to $(IDS, K1, K2)$ stored in the database. We denote these variables as $(IDS_1, K1_1, K2_1)$. Now, suppose the reader goes to read the tag. The attacker records the messages (A, B, C) as (A', B', C'). At the end of the protocol, the attacker interrupts the message D so that the reader will not update its variables. However, the tag will update its variables as follows:

a) $(IDS_{old}, K1_{old}, K2_{old}) = (IDS_1, K1_1, K2_1)$,
b) $(IDS_{next}, K1_{next}, K2_{next}) = (IDS_2, K1_2, K2_2)$.

Next, we allow the reader and the tag to run the protocol again without intervening them. Because IDS_2 is not found in the database, both the reader and the tag use IDS_1 to communication. Thus, the database will update its variable list to $(IDS_3, K1_3, K2_3)$. In the tag, the values of $(IDS_{old}, K1_{old}, K2_{old})$ are now updated to $(IDS_1, K1_1, K2_1)$ and $(IDS_{next}, K1_{next}, K2_{next})$ are now updated to $(IDS_3, K1_3, K2_3)$.

Finally, when the reader leaves the reading range of the tag, the attacker imitates as a valid reader to query the tag. The tag will reply IDS_{next}, which is IDS_3. The attacker pretends that he cannot find IDS_{next} and requests the old IDS. The tag will response IDS_{old}, which has the value IDS_1. The attacker now replays the recorded message A_1, B_1, C_1 to the tag. Since these values were computed by a valid reader with IDS_1 previously, the tag will treat the attacker as a valid reader and update its variables again as:

a) $(IDS_{old}, K1_{old}, K2_{old}) = (IDS_1, K1_1, K2_1)$,
b) $(IDS_{next}, K1_{next}, K2_{next}) = (IDS_2, K1_2, K2_2)$.

Now, they are desynchronized since the values stored in the database are $(IDS_3, K1_3, K2_3)$, which are completely different from the values stored in the tag.

4 The Second Attack

We assume that there is a synchronized tag with the above settings. The attacker eavesdrops on a successful session between the tag and the reader, and records the values (A, B, C) as (A_1, B_1, C_1). At the same time, the database updates its variable list to $(IDS_2, K1_2, K2_2)$. In the tag, the values of $(IDS_{old}, K1_{old}, K2_{old})$ are $(IDS_1, K1_1, K2_1)$ and $(IDS_{next}, K1_{next}, K2_{next})$ are $(IDS_2, K1_2, K2_2)$.

When the reader leaves the reading range of the tag, the attacker initiates the protocol and requests IDS_1 by claiming a mismatching for IDS_2.

Thus, the tag will reply with IDS_1. The attacker's goal is to forge a tuple (A'_1, B'_1, C'_1) that is accepted by the tag. The attack makes $A'_1 = A^*_1$ where A^*_1 is to flip the k-th bit in A_1, $B'_1 = B_1$, and $C'_1 = C^*_1$ where C^*_1 is to flip the most significant bit (MSB) of C_1. Then, the attacker replies the tag with (A'_1, B'_1, C'_1).

Note that in the protocol of SASI, flipping the k-th bit in A leads to the k-th bit in $n1$ be flipped if IDS and K remain unchanged. Therefore, the k-th bit in $K2 \oplus n1$ will flip.

If the flipped bit is coincidentally rotated to the MSB in $\overline{K}2$, then C will be changed in the MSB. This is because the addition overflowing bit under $mod\ 2^{96}$ would be discarded. Therefore, $x + y$ only differs from $x^* + y$ in the MSB if x^* only differs from x in the MSB. More precisely, there are eight cases. Let us use X, Y, and C_{MSB} to denote the MSBs of $(K1 \oplus \overline{K}2)$, $(K2 \oplus \overline{K}1)$, and C respectively. Let *carry* represent whether the sum of the rest bits of the two operands generates a carry bit. We have C'_{MSB} denote the MSB of C after we flip X. The truth table is shown in Table 1.

Table 1. Truth table

X	Y	$carry$	$\overline{X}$	C_{MSB}	C'_{MSB}
0	0	0	1	0	1
0	0	1	1	1	0
0	1	0	1	1	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	0	0	1
1	1	1	0	1	0

In this way, C'_{MSB} always flips and C^*_1 from the attacker will pass the verification process of the tag. Since the rotation is controlled by the Hamming weight of $K2_1$, the attacker can obtain an authenticated tuple (A'_1, B'_1, C'_1) by at most 96 trials for all possible values of k. We also note that an authenticated tuple can be confirmed if there is a response D' from the tag in Step 6. In fact, D' differs from D in the MSB, too. Once an authenticated tuple (A'_1, B'_1, C'_1) is accepted by the tag, the tag will update $(IDS_{next}, K1_{next}, K2_{next}) = (IDS_2, K1_2, K2^*{}_2)$, where $K2^*_2$ has the k-th bit flipped in $K2_2$.

In the next time, when the reader tries to read the tag, the tag replies IDS_2. This value can be found in the database, but the reader will be rejected by the tag, since the key $K2_{next}$ stored in the tag is no longer synchronized with the database. This makes them de-synchronized.

5 Discussion and Conclusion

In order to prevent the first attack, it is possible to store two copies of variables in the database.

In this way, the old IDS, i.e., IDS_{old}, can be found in the database so that the first attack can not work. However, this approach is still vulnerable to the second attack. In the second attack, IDS_{next} in the tag is the same as IDS_{next} in the database. However, the reader cannot be authenticated due to the difference in $K2_{next}$.

We often find security loopholes in authentication protocols without hash functions. It is still a hard challenge to design an ultralightweight secure authentication protocol.

Acknowledgment

We thank Prof. Chien for clarifying the design.

References

1. H.-Y. Chien, "SASI: A new ultralightweight rfid authentication protocol providing strong authentication and strong integrity," *IEEE Transactions on Dependable and Secure Computing*, vol. 4, no. 4, pp. 337–340, 2007.
2. H.-Y. Chien, private communication.