An Efficient Offline Delegation Protocol in Mobile RFID Environment

Jia-Ning Luo
Ming Chuan University, Information & Telecommunications Engineering, Taoyuan County, Taiwan 333
Email: deer@mail.mcu.edu.tw

Ming-Hour Yang*
Chung Yuan Christian University, Information & Computer Engineering, Taoyuan County, Taiwan 320
*Corresponding Author, Email: mhyang@cycu.edu.tw

Abstract—In this paper, we propose a new protocol to allow delegation transfer between offline mobile readers in the mobile RFID (Radio Frequency Identification) environment. A mobile reader can grant the access rights of a specific tag to another reader. Besides, our protocol is efficient and secure against most current network threats, such as replay attacks, Man-in-the-Middle (MITM) attacks, denial of service (DoS) attacks caused by asynchronous update, guessing attacks, and counterfeit tags. It also guarantees forward/backward secrecy, data privacy, and location privacy.

Index Terms—Mobile RFID; Offline Delegation; Delegation Transfer

I. INTRODUCTION

Radio frequency identification (RFID) has been widely applied in logistics management, entrance control, smart appliances, medical control, and e-wallet services. An RFID framework consists of back-end servers, database, RFID readers, and tags. RFID tags include active tags and passive tags. An active RFID tag contains a battery, and a passive tag is powered by radio wave energy from an RFID reader. Most RFID tags used in supply chains are passive ones. Because of the limitation of power consuming and gate counts, a passive RFID tag can only do simple computation and have limited storage capacity. Most of its data is stored on the back-end database servers.

In recent years RFID technology has seen the integration of RFID and mobile devices [2] [7] [8]. An RFID-embedded cellphone can access a tag and retrieve its data from the back-end server through wireless or 3G networks [14].

In a mobile RFID environment, even though a mobile reader can communicate with its back-end server through telecommunications, weak reception or poor network quality can usually affect its performance [3] [4] [5] [9] [11] [15]. Among RFID’s research, offline delegation transfer allows users who have been authorized to access a specific tag to transfer a part of their authorization to others without Internet connections [1] [10] [12] [16]. To perform offline delegation, a back-end server has to delegate authority of a specific RFID tag to an RFID reader, and the delegated reader is allowed to access a specific tag in the future even in an offline environment [10] [14] [16].

Fouladgar et al. [5] propose an online delegation and ownership transfer protocol. Lee et al.’s scheme [9] uses timestamps to manage authorization, but in Lee’s protocol, if one user updates a tag’s timestamp, others will have to use the new timestamp to access the tag. When a tag reaches its maximum timestamp update times, all the delegated readers have to connect to their back-end server for new delegation. Yang [15] also proposes an offline delegation protocol that stores an access control list (ACL) on a tag to limit a reader’s access right. He also deploys timestamps in his offline delegation scheme. His back-end server can delegate the access right to a tag, but the access right cannot be transferred from one reader to another.

For these reasons, we propose a new scheme to perform offline delegation transfer. In our protocol, the delegation of a specific tag is within control. A mobile RFID reader is allowed limited access times of a specific tag by its back-end server. The tag can verify and decrease the access counter in each access. After the counter expires, the reader has to request a new delegation from the back-end server. The other reader can request delegation from the reader that owns the tag. A delegation transfer is operated offline without the involvement of the back-end server. After the delegation transfer, the access times of the tag are transferred to the new reader, and the access counter of the previous reader decreases by the same amount. The rest of the paper is organized as follows: section 2 details our proposed protocol; section 3 analyzes the security issues; section 4 deals with performance evaluation and comparison; section 5 concludes our scheme.

II. OFFLINE DELEGATION PROTOCOL

Our offline delegation transfer protocol includes an initial stage and three protocols: (1) offline reader-tag mutual authentication protocol; (2) readers’ delegation protocol and (3) offline delegation transfer protocol, as shown in Figure 1. At the initial stage, mobile readers have been delegated authority to a specific tag and authenticate each other. In the second part, the readers identify and
access the tag offline. In the third part a reader delegate his access rights to another reader.

![Diagram of offline delegation transfer]

**Figure 1.** Offline coupon delegation transfer scheme

Notations uses in this paper are listed in Table 1.

**Table 1. Notations**

<table>
<thead>
<tr>
<th>TID&lt;sub&gt;m&lt;/sub&gt;</th>
<th>Tag m’s identifier.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RID&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Reader i’s identifier (target reader in delegation transfer)</td>
</tr>
<tr>
<td>Secret&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Secret key shared between m and its back-end server.</td>
</tr>
<tr>
<td>TS&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Tag m’s timestamp.</td>
</tr>
<tr>
<td>RS&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Reader’s current timestamp.</td>
</tr>
<tr>
<td>TC&lt;sub&gt;i&lt;/sub&gt;, TC&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Counters that indicate reader i’s and reader j’s maximum query times to tag m. (on reader’s part)</td>
</tr>
<tr>
<td>RC&lt;sub&gt;i&lt;/sub&gt;, RC&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Counters that indicate reader i’s and reader j’s maximum query times to tag m. (on back-end server part)</td>
</tr>
<tr>
<td>Rlist&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Tag m’s access control list.</td>
</tr>
<tr>
<td>Tlist&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Reader i’s authorization table.</td>
</tr>
<tr>
<td>RK&lt;sub&gt;i&lt;/sub&gt;, RK&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Reader i’s and reader j’s shared keys with tag m, respectively.</td>
</tr>
<tr>
<td>DM&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Delegation message, generated by back-end server for reader i to access tag m.</td>
</tr>
<tr>
<td>DC&lt;sub&gt;i&lt;/sub&gt;</td>
<td>Requested query times in delegation transfer.</td>
</tr>
<tr>
<td>DT</td>
<td>Delegation transfer flag.</td>
</tr>
<tr>
<td>SK&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Session key, used to encrypt/decrypt a delegation message.</td>
</tr>
<tr>
<td>r&lt;sub&gt;i&lt;/sub&gt;, r&lt;sub&gt;m&lt;/sub&gt;</td>
<td>Nonce generated by back-end server, used in reader-tag communications, before reader i and tag m.</td>
</tr>
<tr>
<td>r&lt;sub&gt;i&lt;/sub&gt; = H(2&lt;sup&gt;r&lt;/sup&gt;)</td>
<td>Random numbers.</td>
</tr>
<tr>
<td>H(·)</td>
<td>Hash function.</td>
</tr>
<tr>
<td>LE(·)</td>
<td>Lightweight symmetric key encryption algorithm.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**A. Initial Stage**

In the initial stage, when an RFID reader i wants to access a tag, it should send a delegation request to the back-end server, and the server will return a tag-list Tlist<sub>i</sub> that contains all the tags owned by the reader. See Figure 2. The server randomly generates a session key RK<sub>i</sub> and a nonce r<sub>i</sub>. After concatenating Secret<sub>m</sub>, RID<sub>i</sub> and r<sub>i</sub>, and generate the session key SK<sub>m</sub>, a delegation message DM<sub>i</sub> contains reader’s and tag’s identifiers, system’s timestamp RS<sub>m</sub>, session key RK<sub>m</sub>, and reader’s maximum query times RC<sub>i</sub>. Then the server sends Tlist<sub>i</sub> to the reader.

**B. Offline Reader-Tag Mutual Authentication Protocol**

After a reader gets the access right of a specific tag from the back-end server, the reader and the tag should do mutual authentication to identify each other, which is performed offline. Figure 3 details the steps of our reader-tag mutual authentication.

![Diagram of mutual authentication]

**Figure 2. Initial stage**

**Figure 3. Mutual authentication protocol**

**Step 1:** Reader i sends a request message RequestAuthentication to tag m, along with its own identifier RID<sub>i</sub>, a nonce r<sub>i</sub>, and a delegation message DM<sub>i</sub>.

**Step 2:** Tag m generates a session key SK<sub>m</sub> and decrypts DM<sub>i</sub> to derive reader’s identifier RID<sub>i</sub>, tag’s identifier TID<sub>m</sub>, reader i’s current timestamp RS<sub>m</sub>, the key RK<sub>m</sub>, and reader i’s delegated query times RC<sub>i</sub>. Tag m compares RID<sub>i</sub> and TID<sub>m</sub> with RID<sub>i</sub> and TID<sub>m</sub>, respectively. It also checks if RS<sub>m</sub> ≥ TS<sub>m</sub>. If RS<sub>m</sub> < TS<sub>m</sub>, the reader’s delegated authority expires. If RS<sub>m</sub> ≥ TS<sub>m</sub>, tag m has to update its timestamp, i.e. TS<sub>m</sub> = RS<sub>m</sub>, and to clear its access control list Rlist<sub>m</sub>. Then tag m stores RID<sub>i</sub>, RK<sub>m</sub> and RC<sub>i</sub> on its Rlist<sub>m</sub>. If reader i is valid, tag m sends a message M<sub>1</sub> to the reader, as shown in figure 3. If reader i fails to pass the authentication, tag m returns a random number.

**Step 3:** When receiving M<sub>1</sub>, reader i derives TID<sub>m</sub> and r<sub>i</sub> and compare them to TID<sub>m</sub> and r<sub>i</sub>.

**C. Reader’s Delegation Protocol**

In our protocol, the delegation of a specific tag is limited. Each time when a reader accesses a tag, a counter decrease. After the counter expires, the reader has to
request a new delegation from the back-end server. We use two counters TC and RC to restrict delegated readers’ access to a tag, as shown in Figure 4. Detailed steps are as follows:

1. \( \text{Generate } \); \( \text{Tag } \# \)

2. \( \text{Derive } \).

3. \( \text{Request } \).

Figure 4. Steps of reader’s access to tag

Step 1: Reader i constructs \( M_2 \) by using \( RK^m_1 \) to encrypt \( RID_i \) and a random number \( r_2 \). Then it sends \( \text{Request}_{\text{read}} \) its own identifier \( RID_i \), and \( M_2 \) to the tag \( m \).

Step 2: Tag \( m \) checks if \( RID_i \) is stored on \( \text{Rlist}_m \), and looks up the \( RK^m_1 \) to decrypt \( M_2 \) to obtain \( ID_i \) and \( r_2 \). If \( TID_m \) matches \( TID_m \), tag \( m \) checks if reader \( i \) has run out of its query times. If \( TC^{m}_m \) > 0, tag \( m \) allows reader \( i \)’s access. It generates a random number \( r_3 \), and uses \( RK^m_2 \) to encrypt \( TID_m \), \( TC^{m}_m \), \( r_2 \) and \( r_3 \). Then tag \( m \) returns the encrypted message \( M_3 \) to reader \( i \). If a reader fails to pass any of the verification, tag \( m \) returns \( r_3 \) as \( M_3 \) to reader \( i \).

Step 3: Reader \( i \) uses \( RK^m_2 \) to decrypt \( M_3 \) and then verifies whether the received \( r_3 \) and \( TID_m \) match its own \( r_2 \) and \( TID_m \). If they match, reader \( i \) can confirm \( M_3 \) is not replayed and the sender of \( M_3 \) is tag \( m \). Next, reader \( i \) updates its counter, i.e. \( RC^{m}_m = TC^{m}_m - 1 \). Last, reader \( i \) uses \( RK^m_3 \) to encrypt the updated \( RC^{m}_m \) and \( r_3 \). The encrypted message is returned to tag \( m \) as \( M_4 \). If \( M_3 \) cannot be verified, reader \( i \) returns \( r_2 \) as \( M_4 \) to tag \( m \).

Step 4: Tag \( m \) decrypts \( M_4 \) with \( RK^m_3 \). Then tag \( m \) verifies whether the received \( r_3 \) matches its \( r_3 \), and whether the received \( RC^{m}_m \) has been updated, i.e. \( RC^{m}_m = TC^{m}_m - 1 \). If they are verified, tag \( m \) approves reader \( i \)’s access and then updates its own counter \( TC^{m}_m \), i.e. \( TC^{m}_m = RC^{m}_m \).

D. Offline Delegation Transfer Protocol

In our offline delegation transfer protocol, the access right of a reader can be transferred to another reader through tag \( m \)’s authorization. For example, the initial query times of readers \( i \) and \( j \) are 3 and 18 respectively, i.e. \( TC^{i}_i = 3 \) and \( TC^{j}_j = 18 \). Reader \( j \) uses the offline delegation transfer protocol to transfer 10 query times to reader \( i \); at last the query time of reader \( i \) and \( j \) are 13 and 8 respectively, i.e. \( TC^{i}_i = 13 \) and \( TC^{j}_j = 8 \), see Figure 5.

Our offline delegation transfer requires a tag and two mobile readers (one to request delegation transfer, the other to transfer authorized query times). Both reader \( i \) and reader \( j \) have to be delegated authority in advance to access tag \( m \). We assume that reader \( i \) has run out of its delegated query times. It needs to request delegation transfer from reader \( j \), so that it can acquire some extra query times to access the tag. One particular requirement in our delegation transfer is that the transfer must be performed under the same timestamp. Only in doing so can tag \( m \) have full record of all delegated readers’ information. This requirement also empowers our back-end server to use timestamps to restrict readers’ delegated authority.

Figure 5. Message flow of offline delegation transfer

Reader \( j \) provides tag \( m \) with the information required for delegation transfer. Once the tag approves the request, it will update its own access control list and then the transfer of delegation is done. The transfer includes 6 steps, as shown in Figure 6 and Figure 7.

Figure 6. Offline delegation transfer (1)

Step 1: Reader \( i \) sends to reader \( j \) a transfer request \( \text{Request}_{\text{Reader DT}} \), its own identifier \( RID_i \), tag \( m \)’s identifier \( TID_m \), and requested query times \( DC^{m}_m \).

Step 2: Reader \( j \) generates a delegation transfer request \( \text{Request}_{\text{Tag DT}} \), and uses \( RK^m_0 \) to encrypt \( r_s \), \( RID_i \), \( TID_m \) and \( DC^{m}_m \). Next it sends the encrypted message \( M_5 \), the request \( \text{Request}_{\text{Tag DT}} \), and its own identifier \( RID_j \) to tag \( m \).
Step 3: If $RIT_d$ is on tag $m$’s $RList_m$, the tag uses $RK_i^m$ to decrypt $M_8$ and obtains $TID_m^i$, $RID_j^i$, $DC_i^m$ and $r_4$. Then the tag checks if $TID_m^i = TID_m^j$ and if $TC_i^m \geq DC_i^m$. It has to make sure that tag $m$ is the tag that the two readers try to access, and that reader $j$ still has enough query times for delegation transfer. If $RIT_d$ is not on $RList_m$, $DC_i^m$ will be set zero. At last, tag $m$ approved the delegation transfer by constructs $M_4$ by using $RK_j^m$ to encrypt $r_5$ and $RC_i^m$ and it sets the flag $DT$ to $DT_{confirm}$. Then $M_4$ is returned to tag $m$ and $DT$ is sent to reader $i$. If the received data in $M_4$ is not valid, reader $j$ returns $r_3$ as $M_2$ to tag $m$ and checks $DC_i^m$. If $DC_i^m = 0$, it means $RIT_d$ is not on $RList_m$ and delegation transfer cannot be performed and $DT$ will be set as $DT_{RID, Error}$ to indicate the illegitimacy of $RIT_d$. If $DC_i^m$ is not zero, reader $j$ will set $DT$ as $DT_{Failed}$ to signal the failure of delegation transfer, which may result from message loss or false messages.

Step 4: Reader $j$ decrypts $M_4$ with $RK_i^m$, and obtains $TID_m^i$, $TC_j^k$, $RID_j^i$, $DC_i^m$, $r_4$ and $r_5$. If they are valid, reader $j$ updates its maximum query times, i.e. $RC_i^m = TC_i^m - DC_i^m$. Then it constructs $M_4'$ by using $RK_j^m$ to encrypt $r_5$ and $RC_i^m$ and it sets the flag $DT$ to $DT_{confirm}$. Then $M_4'$ is returned to tag $m$ and $DT$ is sent to reader $i$. If the received data in $M_4'$ is not valid, reader $j$ returns $r_3$ as $M_2$ to tag $m$ and checks $DC_i^m$. If $DC_i^m = 0$, it means $RIT_d$ and delegation transfer cannot be performed and $DT$ will be set as $DT_{RID, Error}$ to indicate the illegitimacy of $RIT_d$. If $DC_i^m$ is not zero, reader $j$ will set $DT$ as $DT_{Failed}$ to signal the failure of delegation transfer, which may result from message loss or false messages.

Step 5: If reader $i$ receives $DT_{RID, Error}$ or $DT_{Failed}$, delegation transfer stops. If reader $i$ receives $DT_{confirm}$ from reader $j$, it will wait for tag $m$’s approval message $M_6$ so that it can update its query times $RC_i^m$.

Step 6: Tag $m$ decrypts $M_2$ with $RK_i^m$ and obtains the updated $RC_i^m$ and $r_5$. If $r_4$ and $r_5$ match and $RC_i^m = TC_i^m - DC_i^m$, tag $m$ updates $RList_m$’s $TC_j^m$ and $TC_i^m$. That is, $TC_j^m = RC_i^m$, $TC_i^m = TC_i^m + DC_i^m$. Then the tag constructs $M_6$ by using $RK_j^m$ to encrypt the updated $TC_i^m$, $TID_m^i$, $r_4$ and $r_5$. $M_6$ is forwarded to tag $i$ through tag $j$. If $M_2$ is not valid, tag $m$ terminates the protocol. Otherwise reader $i$ decrypts $M_6$ with $RK_i^m$ and obtains $TID_m^i$ and $TC_j^m$. If $TID_m^i$ and $TID_m^j$ match, reader $i$ updates its maximum query times, i.e. $RC_i^m = TC_i^m$, and offline delegation transfer is done.

### III. Security Analysis

Our protocol is designed to perform delegation transfer between offline readers. Without the involvement of back-end servers, we use timestamps and access control lists, e.g. $RList_m$ and $TList_i$, to control readers’ delegated authority. Apart from these, we deploy hash functions to generate keys, add random numbers into our messages, and use lightweight symmetric key encryption [6] [13] in our communications. All of these are intended to enhance our security and to lower the chance of attacks. Therefore, our security analysis will particularly focus on mutual authentication, replay attacks, MITM attacks, forward secrecy, backward secrecy, guessing attacks, location privacy, data privacy, and DoS attacks in asynchronous update.

#### A. Mutual Authentication

Our reader-tag communications are all encrypted with symmetric keys. Only with reader-tag shared keys, e.g. $RK_i^m$ and $RK_j^m$, can the two decrypt each other’s messages and hence authenticate each other. Even though adversaries intercept our delegation messages, their readers cannot pass our authentication because the messages contain delegated readers information. If they try to use an intercepted message to perform authentication, they do it simply for a legitimate reader whose identifier is in the message. Besides, without the keys they cannot even decrypt the intercepted messages.

#### B. Replay Attacks

Since we use nonce in our reader-tag communications, the messages will change in every session. Thus, attackers are unable to replay any legitimate messages to pass our authentication. However, during our offline mutual authentication, there is only one $r_i^m$ from a back-end server under the same timestamp. Here, malicious users may take this opportunity to replay reader $i$’s message to tag $m$. But since tag $m$ only stores the information in the first successful authentication, replayed messages are unable to change anything. Also, tag $m$’s responses contain a nonce $r_i$, which change in each communication. So, despite the possibility of replay attacks in this part, they cannot change anything.

#### C. MITM Attacks

First, our reader-tag messages are protected with encryption algorithms. Second, in our delegation transfer or when a delegated reader accesses a tag, the update of maximum query times requires the tag’s reconfirmation with the delegated reader. Since adversaries are unable to launch replay attacks and they do not have $RK_i^m$ or $RK_j^m$, they are unable to counterfeit tags or to launch MITM attacks, either.

#### D. Forward Secrecy

Because we use timestamps to control our delegated authority, once the timestamp is updated old delegation messages are all cleared. Old information will not be exposed to new users. Hence, forward secrecy is secured.
E. Backward Secrecy

Our delegation message $DM_i^m$ is protected by a session key $SK_i^m$. If Tag $m$ wants to read the message, it needs to generate the key by hashing $\text{Secret}_{m_i}, RID_i$ and $r_i^m$. Thus, even though attackers can obtain $SK_i^m$, they are unable to break $\text{Secret}_{m_i}$ because the session key is a hash value. Attackers cannot trace back the elements of a hash function from a hash value. Once $\text{Secret}_{m_i}$ changes in the next session, $SK_i^m$ changes as well. Adversaries will not be able to track messages in following sessions. Hence, backward secrecy is secured.

F. Guessing Attacks

Though our communications are all secured with keys, malicious users may eavesdrop on our messages and then launch guessing attacks. Therefore, we use random numbers as responses when a reader or a tag fails to pass our authentication. This can effectively lower the chance of guessing attacks.

G. Location Privacy

Because we put random numbers in our communications, our messages change in each session. Despite eavesdropping, eavesdroppers cannot even tell whether the messages they overhear come from the same tag. So, they are unable to track our tags’ location.

H. Data Privacy

Our delegation message is protected by $SK_i^m$ and our reader-tag communications are secured with $RK_i^m$ or $RK_j^m$. That is to say, without these keys, attackers can by no means pass our authentication and access our tags’ data. Besides, since replay, MITM and guessing attacks cannot breach our protocol, we can say without a back-end server’s authorization illegitimate reader is unable to access our tags. Hence, tags’ data privacy is secured.

I. DoS Attacks

Our tag does not change its keys during reading, offline authentication, or offline delegation transfer. First, it acquires $RK_i^m$ or $RK_j^m$ from its back-end server’s delegation messages. Then it runs a hash function to generate a session key $SK_i^m$. Therefore, there will be no asynchronous update of keys in our protocol. Still, asynchrony may occur when $RC_i^m$ has been updated but $M_2$ is missing. $TC_j^m$ and $TC_i^m$ will stay the same. Nonetheless, such asynchrony can be fixed up in the next access. It will not affect our readers’ reading or delegation transfer.

In Table 2, we compare our protocol with other delegation protocols. It is obvious that our scheme is able to resist most network threats and is capable of delegation transfer. Compared with Fouladgar et al.’s protocol, our scheme achieves mutual authentication by securing our reader-tag communications with lightweight encryption algorithms. As mentioned above, our protocol is able to guarantee backward secrecy, which is impossible in Fouladgar et al.’s and Lee et al.’s schemes. The best part is that our protocol is the only one capable of offline delegation transfer in this comparison. Such ability makes delegation more useful and flexible in a mobile RFID environment.

IV. PERFORMANCE

This section analyzes our protocol’s performance and compares our performance with that of other related studies. Here we use $T_{RG}$ to denote the time for one lightweight encryption/decryption; $T_H$: the time for one hash function; $T_{RNG}$: the time to generate a random number. Table 3 depicts the detailed computation requirement for our reader and tag when we run the delegation scheme. If there are errors during the communications, the loads can even be lower. When errors occur, we only respond with a random number. No further en/decryption will be required. As for the target reader $i$, it only runs one lightweight encryption algorithm during delegation transfer. Therefore, we leave it out of our analysis.

As shown in Table 3, our use of random numbers in almost every action helps us prevent guessing attacks. Since our mutual authentication requires a tag to store the first legitimate delegation message, a delegated reader does not need to generate any random number for this action. It only needs one decryption. But a tag has to run a hash function to generate $SK_i^m$; decrypt $DM_i^m$; and encrypt $M_1$. When a delegated reader access a tag, the two’s computation loads are the same. The former needs two encryptions and one decryption and the latter one encryption and two decryptions. In offline delegation transfer, a tag has to encrypt $M_{10}$ for reader $i$, so its computation load is one encryption more than reader $j$’s.

Next, we compare our scheme’s computation load with that of other delegation protocols, as shown in Table 4. For an offline reader, when it searches a tag’s information in its authorization table, e.g. $Tlist_i$, the computation load should be taken into account. In Table 4, we also assume that the information of a tag to be accessed must exist in the reader’s authorization table. In the cases of multiple access, i.e. a reader being delegated authority to access multiple tags, we divide the reader’s total computation loads by the number of tags. And we begin to count a tag’s
computation load after it receives a reader’s request. Here we use $RL$ to denote how many tags a reader is allowed to access; $RS$: reader’s current timestamp; $TS$: tag’s current timestamp; $TS_{Max}$: the maximum value of a timestamp; $C$: how many times a tag has been queried (a tag’s counter in Lee et al.’s scheme); $RC$: a reader’s current query times (a reader’s counter in Yang’s scheme); $RC_{Max}$: the maximum delegated query times.

As shown in Table 4, we can find that it takes much computation for a reader to verify a tag’s messages. And a tag’s computation load varies from scheme to scheme and it is not affected by the number of delegated readers. Though Fouladgar et al.’s protocol [5] demands low computation for their reader and tag, their scheme is unable to perform reader-tag mutual authentication and therefore is vulnerable to certain network threats. The computation loads in Lee et al.’s approach [9] are highly influenced by their use of timestamps and by their key updating. Every time when Yang’s tag receives a request from a reader, it has to compute a complex hash chain to generate a session key, so as to authenticate the reader. It gets worse in their first query because the tag also needs to generate a session key, so as to authenticate the reader. It can cause quite a burden for an RFID tag since it only has limited computing ability.

In our scheme, when under the same timestamp a reader needs to run two en/decryption algorithms if the reader-tag authentication fails; three en/decryption algorithms if the authentication is successful. Our reader’s average computation load will not outweigh other schemes’ readers. And our tag computes three en/decryption algorithms in both situations. Judging from the average computation loads, our proposed scheme has better performance in every aspect.

V. CONCLUSION

In this paper, we propose a delegation transfer protocol to allow offline readers to transfer their delegated authority to another authorized reader in the mobile RFID environment. Our protocol is also designed to secure against certain network threats, such as replay attacks, MITM attacks, DoS attacks in asynchronous update, counterfeit tags, and guessing attacks. It can guarantee forward/backward secrecy, data privacy, and data privacy.

TABLE IV: AVERAGE COMPUTATION LOADS OF DELEGATION PROTOCOLS.

<table>
<thead>
<tr>
<th>Devices</th>
<th>Schemes</th>
<th>Reader</th>
<th>Tag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$(1 + RL)\times T_{LE} + T_{RING}$</td>
<td>$(1 + RL)\times T_{LE} + T_{RING}$</td>
</tr>
<tr>
<td>Fouladgar et al. [5]</td>
<td>Hash-Based</td>
<td>$T_{H} + T_{RING}$</td>
<td>$2T_{LE} + T_{RING}$</td>
</tr>
<tr>
<td></td>
<td>Encryption-Based</td>
<td>$\frac{(1 + RL)}{2} T_{LE} + T_{RING}$</td>
<td>$\frac{(1 + RL)}{2} T_{LE} + T_{RING}$</td>
</tr>
<tr>
<td></td>
<td>Lee et al. [9]</td>
<td>Same timestamp: $(1 + RL) \cdot C + 1 \cdot T_{H} + T_{RING}$</td>
<td>Same timestamp: $2(RS - TS)T_{H} + T_{RING}$</td>
</tr>
<tr>
<td></td>
<td>Yang [10][15]</td>
<td>Same timestamp: $(11 + RL) / 2 \cdot T_{H} + T_{RING}$</td>
<td>Same timestamp: $(TS_{Max} - RS + 8 + RC)T_{H} + T_{RING}$</td>
</tr>
<tr>
<td></td>
<td>Different timestamp</td>
<td>$(11 + RL) / 2 \cdot T_{H} + T_{RING}$</td>
<td>Different timestamp: $(TS_{Max} - RS + 8 + RC_{Max})T_{H} + T_{RING}$</td>
</tr>
<tr>
<td></td>
<td>Our scheme</td>
<td>Same timestamp: $(RL + 2)T_{LE} + T_{RING}$</td>
<td>Same timestamp: $3T_{LE} + T_{RING}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Different timestamp: $(7 + RL) / 2 T_{LE} + T_{RING}$</td>
<td>Different timestamp: $T_{H} + 5T_{LE} + 2T_{RING}$</td>
</tr>
</tbody>
</table>

ACKNOWLEDGMENT

This work was supported in part by National Science Council under the grants NSC101-2221-E-033-016-MY2.

REFERENCES


Jia-Ning Luo holds a PhD degree in Computer Science of National Chiao Tung University, Taiwan. He specializes in network security, operating systems, network administration and network programming. He is currently working on NFC-based protocols; RFID ownership transfer; eWallet security.

Ming-Hour Yang received his doctoral degree in Computer Science & Info. Engineering at National Central University, Taiwan. His research mainly focuses on network security and system security with particular interests on security issues in RFID and NFC security communication protocols. Topics include: mutual authentication protocols; secure ownership transfer protocols; polymorphic worms; tracing mobile attackers.