Vulnerability analysis of RFID protocols for tag ownership transfer

Pedro Peris-Lopez\textsuperscript{a,}\textsuperscript{*}, Julio C. Hernandez-Castro\textsuperscript{b}, Juan M.E. Tapiador\textsuperscript{b}, Tieyan Li\textsuperscript{c}, Yingjiu Li\textsuperscript{d}

\textsuperscript{a}Information and Communication Theory Group, Delft University of Technology, Netherlands
\textsuperscript{b}Computer Science Department, Carlos III University of Madrid, Spain
\textsuperscript{c}Institute for Infocomm Research, A*STAR Singapore, Singapore
\textsuperscript{d}Singapore Management University (SMU), Singapore

\textbf{A B S T R A C T}

In RFIDSec’08, Song proposed an ownership transfer scheme, which consists of an ownership transfer protocol and a secret update protocol\textsuperscript{[7]}. The ownership transfer protocol is completely based on a mutual authentication protocol proposed in WiSec’08\textsuperscript{[8]}. In Rizomiliotis et al. (2009)\textsuperscript{[6]}, van Deursen and Radomirovic (2008), the first weaknesses to be identified (tag and server impersonation) were addressed and this paper completes the consideration of them all. We find that the mutual authentication protocol, and therefore the ownership transfer protocol, possesses certain weaknesses related to most of the security properties initially required in protocol design: tag information leakage, tag location tracking, and forward traceability. Moreover, the secret update protocol is not immune to de-synchronization attacks.

\textcopyright 2009 Elsevier B.V. All rights reserved.

1. Introduction

Many protocols have been proposed aiming to provide secure contact between RFID reader and tag over the open radio channel. However, due to tag limitations in terms of circuitry (computation power), storage and power consumption, designing an efficient and secure mutual authentication protocol is still a great challenge. In this paper, we analyze a new mutual authentication protocol proposed by Song and Mitchell\textsuperscript{[8]}, hereafter referred to as the SM protocol. This protocol is said to resist tag information leakage, tracking, tag and server impersonations, replay attacks, denial of service attacks and forward and backward traceability.

In addition to the mutual authentication problem, some authors are working on how to make an RFID system secure when a tag changes its owner multiple times during its life cycle\textsuperscript{[2,4,5,7]}. This issue is known as the ownership transfer problem. It arises when the server of a new owner takes over tag authorization, and so is given certain private information to interact with the tag in a secure manner. In\textsuperscript{[7]}, the usability and security of previous proposals that address the ownership transfer problem are scrutinized before a new ownership transfer scheme is proposed. The new scheme is divided into two sub-protocols, an ownership transfer protocol (wholly based on SM) and a secret update protocol. This paper examines the security of both protocols and identifies their vulnerabilities.

2. Review of RFID protocols for ownership transfer

In this section, we review the newly proposed protocols for secure ownership transfer, including the mutual authentication protocol\textsuperscript{[8]}, the ownership transfer protocol, and the secret update protocol\textsuperscript{[7]}. These protocols are viable when tags can generate random strings and support an on-board hash function or MAC.
2.1. Proposed protocols

We use the same notation as in the original paper [7,8] to describe protocols (see Table 1). First, the initialization phase of the tag and server is introduced. An initiator assigns a string $s_i$ of $l$ bits to each tag $T_i$, computes $t_i = h(s_i)$, and stores $t_i$ in the tag. The parameter $l$ should be great enough so that an exhaustive search aimed at finding the $l$-bit values of $t_i$, $s_i$, and $r$ is computationally infeasible. Additionally (although it is not mentioned in the original paper), length $l$ should be selected to ensure unequivocal identification of tagged items. A legitimate server stores a tuple of the form $(s_i, t_i)$, respectively.

2.1.1. Mutual authentication protocol

An outline of the SM mutual authentication protocol is shown below (see Fig. 1).

1. Reader → Tag: The reader generates a random bit-string $r_1 \in \{0, 1\}^l$ and sends it to $T_i$.
2. Tag → Reader: The tag $T_i$ generates a random bit-string $r_2 \in \{0, 1\}^l$, computes $M_1 = t_i \oplus r_2$ and $M_2 = f_i(r_1 \oplus r_2)$, and sends $(M_1, M_2)$ to the reader.
3. Reader → Server: The reader sends $(r_1, M_1, M_2)$ to the server.
4. Server → Reader: The server chooses $t_i$ from the values $t_{\text{new}}$ or $t_{\text{old}}$ for $1 \leq i \leq N$. It computes $M'_2 = f_i(r_1 \oplus M_1 \oplus t_i)$. If $M'_2 = M_2$, then the server has identified and authenticated the tag as $T_i$. In this case, let the current secrets be denoted as $(s_i, t_i)$. If no match is found, the server sends $\epsilon$ to the reader and stops the session. The server then computes $M_3 = s_i \oplus (r_2 \gg 1/2)$ and sends it with $D_i$ to the reader. Finally, the server updates $t_{\text{new}}$ and $t_{\text{old}}$ for the tag $T_i$ to $s_i$ and $t_i$, respectively, and sets $s_{\text{new}} = (s_i \ll 1/4) \oplus (t_i \gg 1/4) \oplus r_1 \oplus r_2$ and $t_{\text{new}} = h(s_{\text{new}})$ in its database.
5. Reader → Tag: The reader forwards $M_3$ to $T_i$, which computes $s_i = M_2 \oplus (r_2 \gg 1/2)$ and checks if $h(s_i) = t_i$. If the check succeeds, the tag has authenticated the server, and sets $t_i = h(s_i \ll 1/4) \oplus (t_i \gg 1/4) \oplus r_1 \oplus r_2)$. If the check fails, the tag keeps the current value of $t_i$ unchanged.

2.1.2. Ownership transfer protocol

In [7], Song proposed an ownership transfer protocol wholly based on the SM protocol. For convenience, we denote the current owner of a tag as $S_j$ and the new owner as $S_{j+1}$. The details of the ownership transfer protocol are shown below (see Fig. 2).

1. $S_{j+1} \rightarrow T_i$: $S_{j+1}$ generates a random string $r_1$ of $l$ bits and sends it to $T_i$.
2. $T_i \rightarrow S_{j+1}$: $T$ generates a random string $r_2$ of $l$ bits, computes $M_1 = t_i \oplus r_2$ and $M_2 = f_i(r_1 \oplus r_2)$, and then sends $(M_1, M_2)$ to $S_{j+1}$.
3. $S_{j+1} \rightarrow S_j$: After receiving $(M_1, M_2)$, $S_{j+1}$ sends $(r_1, M_1, M_2)$ to $S_j$ with a request for ownership of $T_i$ ($R_{T_i}$).
4. $S_j \rightarrow S_{j+1}$: If the received request $R_{T_i}$ is valid, $S_j$ searches its database for a pair $(t_i, s_i)$ for which the value of $t_i$ satisfies $M_2 = f_i(r_1 \oplus M_1 \oplus t_i)$. If such a pair $(t_i, s_i)$ is found, $S_j$ sets $r_2 = M_1 \oplus t_i$ and computes $M_3 = s_i \oplus (r_2 \gg 1/2)$. Otherwise, the session stops. $S_j$ updates the secrets as $s_{\text{old}} = s_i$, $t_{\text{old}} = t_i$, $t_{\text{new}} = (s_i \ll 1/4) \oplus (t_i \gg 1/4) \oplus r_1 \oplus r_2$, and $t_{\text{new}} = h(s_{\text{new}})$. $S_j$ sends $M_3$ to $S_{j+1}$, and transfers the updated secrets $(t_{\text{new}}, s_{\text{new}})$ together with some other necessary information $D_i$ about the tag to $S_{j+1}$ via a secure channel.
5. $S_{j+1} \rightarrow T_i$: When $S_{j+1}$ receives $(t_{\text{new}}, s_{\text{new}}, D_i, M_3)$ from $S_j$, it stores $(t_{\text{new}}, s_{\text{new}})$. $D_i$ in its database, and forwards $M_3$ to $T_i$. Then, it computes $s_i = M_2 \oplus (r_2 \gg 1/2)$, and checks if $h(s_i) = t_i$ holds. If $h(s_i) = t_i$ holds, $T_i$ updates its secret as $t_i = h(s_i \ll 1/4) \oplus (t_i \gg 1/4) \oplus r_1 \oplus r_2$. Otherwise, the session stops.

2.1.3. Secret update protocol

When a new owner $S_{j+1}$ takes ownership of a tag, he can choose a new value for the tag’s secrets $(t_i, s_i)$, and update the value $t$ on the tag. This updating phase is necessary to provide privacy protection for the new owner of the tag. In other words, its execution will frustrate the identification and traceability of the tag by the old owner $S_j$. The details of the secret update protocol [7] are shown below (see Fig. 3):

1. $S_{j+1} \rightarrow T_i$: $S_{j+1}$ generates random strings $r_1$ and $r_2$ of $l$ bits, and computes $t_i = h(s_i \ll 1/4) \oplus (t_i \gg 1/4) \oplus r_1 \oplus r_2$. Finally, the new owner sends $(r_1, M_1, M_2)$ to $T_i$.

| $l$ | The bit-length of a tag identifier |
| $N$ | The number of tags |
| $f_k$ | A keyed hash function |
| $r_1$ | A random string of $l$ bits |
| $s_i$ | The left/right half part of the string $x$ |
| $r_k$ | Error message |
| $\oplus$ | XOR operator |
| $T_i$ | The $i$th tag $(1 \leq i \leq N)$ |
| $\rightarrow$ | Concatenation operator |
| $D_i$ | The detailed information on tag $T_i$ |
| $s_i$ | A string of $l$ bits assigned to $T_i$ |
| $x \gg k$ | Right circular shift operator, which rotates all bits of $x$ to the right by $k$ bits, as if the left and right ends of $x$ were joined |
| $t_i$ | $T_i$’s identifier of $l$ bits, which equals $h(s_i)$ |
| $x \ll k$ | Left circular shift operator, which rotates all bits of $x$ to the left by $k$ bits, as if the left and right ends of $x$ were joined |
| $x_{\text{new}}$ | The new (refreshed) value of $x$ |

Table 1

The notation.
2. $T_i \rightarrow S_{j+1}$: $T_i$ receives $(r_1, M_1, M_2)$ from $S_{j+1}$, and computes $\tau'_i = f_s(r_1 \oplus M_1 \oplus t_i)$ and $s_i = M_2 \oplus (\tau'_i \gg 1/2)$. If $h(s_i) = t_i$, $T_i$ has authenticated $S_{j+1}$ as an authorized server. Otherwise, the session stops. Then $T_i$ updates its secret as $t_i \leftarrow \tau'_i$, generates a random string $r_3$ of $l$ bits and computes $M_3 = f_s(r_3 \oplus M_1 \oplus t_i)$. On receiving $(r_2, M_3)$ from $S_{j+1}$, $S_{j+1}$ checks whether $M_3$ is equal to $f_s(r_1 \oplus r_2)$. If the validation succeeds, it indicates that $T_i$ possesses the new secret $t'_i$. In this case, $S_{j+1}$ updates its secrets $s_{\text{new}}$ and $t_{\text{new}}$ to $s'_i$ and $t'_i$, respectively; otherwise, $S_{j+1}$ starts a new session.

3. Vulnerability analysis

We use the standard Dolev–Yao intruder model [1] in which the adversary has full control over the “network”. In this model, the adversary can eavesdrop, block, modify and inject messages in any communication between (in our case) a reader and a tag. However, as is commonly assumed, we assume that communications between servers and readers are secure, which can be guaranteed by using fully-fledged cryptographic technologies.

3.1. Server impersonation and denial-of-service attack

It is claimed that the mutual authentication protocol [7,8] is secure against the server impersonation attack because an attacker cannot compute a valid $M_3$ message without knowing $s_i$. However, an active attacker who does not know $(s_i, t_i)$ can use the eavesdropped messages in a valid session to compute a correct $M_3$, thus impersonating a legitimate server. Such an attack on a tag causes loss of synchronization (a denial-of-service attack) between the tag and the impersonated server. The reader is referred
to [6] for a complete description of the attack. Although we have described the attack in terms of the protocol [8], it applies directly to the protocol [7] due to the great similarity.

3.2. Tag impersonation attack

Song et al. argue that the mutual authentication protocol is secure against the tag impersonation attack, since it is difficult for an attacker to compute a valid response [9]. However, Deursen et al. [10] show how an adversary can easily forge a valid message \( M_3 \) by computing a XOR operation with messages/nonces previously intercepted.

3.3. Tracking or compromise of location privacy

It is claimed that the mutual authentication protocol guarantees location privacy for tag owners, since the responses \( M_1 \) and \( M_2 \) provided by the tags are anonymous due to the inclusion of fresh random numbers in these messages. However, the attacker is able to listen in on both the backward and forward channels. In other words, the messages sent by the reader can be eavesdropped too. Our analysis shows that an attacker, by simply listening in on the channel, blocking or altering (e.g., flipping a bit) messages \( M_3 \) sent from the reader to a target tag, is able to discriminate this tag within a population of \( N \) tags (this privacy concept is compatible with the Juels–Weis untraceability model [3]). Details of the attack are given below:

1. The adversary eavesdrops a valid session between the target tag and the reader. A legitimate reader generates a random string \( r_1 \in \{0,1\}^l \) and sends \( r_1 \) to the tag. After receiving \( r_1 \), the tag chooses a random string \( r_2 \in \{0,1\}^l \), and computes messages \( M_1 = t_i \oplus r_2 \) and \( M_2 = f_i(r_1 \oplus r_2) \). The tag sends \( (M_1, M_2) \) to the reader.

2. The target tag is introduced in a population of \( N \) tags — \( N \) being an arbitrary value. The adversary listens in on an authentication session between one of these legitimate tags and the reader. As in the above case, the adversary can eavesdrop messages \( M_1 \) and \( M_2 \), and frustrate the correct reception of message \( M_3' \).

\[
\begin{align*}
M_1' &= t_j \oplus r_2' \quad (1) \\
M_3' &= s_j \oplus (r_2' \gg 1/2) \quad (2)
\end{align*}
\]

where \( j \) can take one of the following values \( 1, \ldots, N \), and \( N \) is the size of the population in which the target tag was introduced. The attacker can guess if the answers provided originate from the target tag by means of the following computation:

\[
\begin{align*}
M_1 \oplus (M_3 \ll 1/2) \neq M_1' \oplus (M_3' \ll 1/2). \quad (3)
\end{align*}
\]

**Proof.** We now prove that \( M_1' \oplus (M_3' \ll 1/2) \) is a constant value which does not depend on the fresh random numbers generated in each session. The attack is completely feasible because once the tag is introduced into the population of \( N \)-tags, the attacker only has to listen in to the channel and prevent the correct reception of message \( M_3' \).

The equations \( M_1' \) and \( M_3' \) are expanded in the following way: we use \( |x|_k \) to denote the left half of the string \( x \), and \( |x|_r \) to denote the right half.

\[
\begin{align*}
M_1' &= t_j \oplus r_2' = [|t_j|_l \oplus |r_2'|_l]|(|t_j|_r \oplus |r_2'|_r) \\
M_3' &= s_j \oplus (r_2' \gg 1/2) = [|s_j|_l \oplus |r_2'|_l]|(|s_j|_r \oplus |r_2'|_r).
\end{align*}
\]

Then, \( M_3' \) is rotated to the left by 1/2 bits:

\[
(M_3' \ll 1/2) = [|s_j|_l \oplus |r_2'|_l]|(|s_j|_r \oplus |r_2'|_r).
\]

**Fig. 3.** Secret update protocol.

<table>
<thead>
<tr>
<th>Server</th>
<th>Tag ( T_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>((s, t_1)<em>{odd}, (s, t_2)</em>{even}, D_i)</td>
<td>(t_i)</td>
</tr>
<tr>
<td>(r_1 \in {0,1}^l)</td>
<td>(t_i' \leftarrow f_i(r_1 \oplus t_i'))</td>
</tr>
<tr>
<td>(s'_i \in {0,1}^l)</td>
<td>(M_2 \leftarrow s_i \oplus (t_i' \gg 1/2))</td>
</tr>
<tr>
<td>(c_i \leftarrow h(s'_i))</td>
<td>(M_2 \leftarrow s_i \oplus (t_i' \gg 1/2))</td>
</tr>
<tr>
<td>(M_1 \leftarrow f_i(r_1) \oplus t_i')</td>
<td>(r_2 \in {0,1}^l)</td>
</tr>
</tbody>
</table>

If \( M_2 = f_i(r_1 \oplus r_2) \)

and \( s_{i,odd} \leftarrow s_{i,even}, s_{i,even} \leftarrow s'_i\)

\(t_{i,odd} \leftarrow t_{i,even}, t_{i,even} \leftarrow t_i'\)

The valid reader then queries the server and gets the updating message \( M_3 = s_i \oplus (r_2 \gg 1/2) \). The adversary acquires values \( M_1 \) and \( M_2 \) and prevents updating of the tag’s internal value \( (t_i) \) by blocking or altering message \( M_3 \).

Please cite this article in press as: P. Peris-Lopez et al., Vulnerability analysis of RFID protocols for tag ownership transfer, Comput. Netw. (2010), doi:10.1016/j.comnet.2009.11.007
Finally,
\[
M'_i \oplus (M''_i \ll 1/2) = [t_i]_L \oplus [s_i]_R || [t]_L \oplus [s]_L.
\]  \hspace{1cm} (7)

We prove that \(M'_i \oplus (M''_i \ll 1/2)\) only depends on \(t_i\) and \(s_i\), which are different values in each tag (unequivocal identification). Therefore, the attacker can identify a tag from a population of tags under the assumption that the updating is precluded in those tags. This is not a rare condition; in fact this situation is considered in the definition of the mutual authentication protocol as a countermeasure for desynchronization attacks. In fact, the protocol definition does not say how many times a tag can be identified using the same identifier \(t_i\). In conclusion, the target tag can be indefinitely identified from a population of tags under the assumption of not updating.

3.4. Information leakage – revealing back-end database content

Prior to using the mutual authentication protocol, an initialization phase should be completed by an initiator. The target tag and a legitimate reader execute a valid authentication session with a legitimate reader:

1. The target tag and a legitimate reader execute a valid authentication session with a legitimate reader:
   - The target tag and a legitimate reader execute a valid authentication session.
   - The reader generates a random string \(r_1 \in \{0, 1\}^l\) and sends \(r_1\) to the adversary.
   - The adversary chooses a random string \(r_2 \in \{0, 1\}^l\) and computes messages \(M_1 = t_i \oplus r_2\) and \(M_2 = f_i(r_1 \oplus r_2)\).
   - The adversary sends messages \((M_1, M_2)\) to the reader.
   - The reader forwards values \(r_1, M_1, M_2\) to the server.
   - The server checks the values, and generates an updating message \(M_3 = s_j \oplus (r_2 \gg 1/2)\), which is sent to the reader.
   - The reader forwards a message \(M_3\) to the adversary.
   - The adversary can obtain the private information \(s_j\) by simply computing an XOR operation, \(s_j = M_3 \oplus (r_2 \gg 1/2)\).

Repeatedly executing the above attack for all the possible values of \(t_i\) (\(s_j = t_i + 1\) for \(0 < j < 2^l\) and \(t_i = 0\)), the adversary is able to acquire all the information stored in the back-end database:

\[
\begin{align*}
\text{for } l, & \quad s_1 = M_3 \oplus (r_2 \gg 1/2) \\
\text{for } l, & \quad s_2 = M_3 \oplus (r_2 \gg 1/2) \\
\text{for } l, & \quad s_3 = M_3 \oplus (r_2 \gg 1/2) \\
\text{for } l, & \quad \ldots \\
\text{for } l, & \quad s_N = M_3 \oplus (r_2 \gg 1/2) \\
\end{align*}
\]

where the maximum size of the database is determined by \(l\) variable. \((N = 2^l)\).

We prove that an exhaustive search is not necessary to acquire \(s_i\). Its acquisition means that information privacy, which should be one of the principal security objectives of the system, is not guaranteed. In fact, this attack is very harmful because after \(N\) authentication sessions, the attacker possesses the private information \([s_i, t_i]_{i=1}^{N}\) linked to each tag and stored on the back-end database. The security flaw described above arises mainly because the answer provided by the server depends only on the XOR between \(s_i\) and the random number \(r_2\) selected by the tag/attacker. Finally, the reader should note that the detailed information \(D_i\) cannot be compromised as this kind of information is transmitted over a secure channel (e.g. SSL connection between the server and reader).

3.5. Tracking of future transactions

It is claimed that the mutual authentication protocol is secure against forward traceability even when the tag is compromised and the attacker knows \(t_i\). In order to guarantee this security property, it is assumed that the attacker cannot prevent \(T_i\) from receiving the last message \(M_3\), or that he does not have access to all the values \(r_1, r_2\) and \(s_i\) that are needed to refresh \(t_i\) [7]. We will show that even with these restrictions, it is possible to compute the new identifier \(t_i\). Details are given below:

1. The target tag and a legitimate reader execute a valid session. The reader generates a random string \(r_1 \in \{0, 1\}^l\) and sends \(r_1\) to the tag. After receiving \(r_1\), the tag chooses a random string \(r_2 \in \{0, 1\}^l\), and computes messages \(M_1 = t_i \oplus r_2\) and \(M_2 = f_i(r_1 \oplus r_2)\) (2010), doi:10.1016/j.comnet.2009.11.007

Proof. The verification of Eq. (8) is equivalent to verifying that \((s_i \ll 1/4) \oplus (t_i \gg 1/4)\) is equal to \((M_1 \gg 1/4) \oplus (M_3 \ll 1/4)\). Note that \(t_i = h(s_i \ll 1/4) \oplus (t_i \gg 1/4) \oplus r_1 \oplus r_2\) in the protocol definition. We can start expanding the equations for \(M_1\) and \(M_2\). We use \([x]_{l,R-LSB,MSB}\) to symbolize the most and least significant bits of the left/right half of the string \(x\).
3.6. De-synchronization attack on the secret update protocol

In Song’s secret update protocol, the new owner/server updates a tag with message \((r_1, M_1, M_2)\) and gets confirmation \((r_2, M_3)\) from the tag if the update is successful. It is claimed that the server can be re-synchronized with the tag even if the confirmation message \(M_3\) is blocked or incorrectly received. For that, the server maintains the new and old values of \(s_t\) and \(t_1\). However, we will show that an active attacker can block the first message \((r_1, M_1, M_2)\) from reaching the tag, and then send a second message \((r_1, M_1’, M_2)\). This last message will be accepted by the tag, resulting in de-synchronization between the tag and the server. In fact, this attack is based on the same principles as the attack presented in [6] because of the similarities between the messages exchanged in the authentication protocol \(\{M_1 = t_1 \oplus r_2, M_3 = s_t \oplus (r_2) \gg 1/2\}\) and the messages passed in the secret update protocol \(\{M_1 = r_1(t_3) \oplus t_1, M_3 = s_t \oplus (t_3' \gg 1/2)\}\). We briefly present the attack below and omit its proof to avoid repetition.

1. A legitimate server starts the secret update protocol by sending \((r_1, M_1, M_2)\) to the tag, where \(r_1 \in_R \{0, 1\}\), \(M_1 = f_1(r_1) \oplus t_1'\) and \(M_2 = s_t \oplus (t_3' \gg 1/2)\). Note that \(s_t\) and \(t_1\) are the tag’s current secrets and that \(s_t’ \in_R \{0, 1\}\) and \(t_1' = h(s_t')\) are the new secrets to be updated.

2. The adversary blocks the message \((r_1, M_1, M_2)\) from reaching the target tag. Then, the adversary sends \((r_1, M_1’, M_2)\), where \(M_1’ \in_R \{0, 1\}\) and \(M_2’ = M_2 \oplus ((M_1 \oplus M_1') \gg 1/2)\).

3. The tag will accept \((r_1, M_1’, M_2)\) and update its secret \(t_1 = M_1' \oplus f_1(r_1)\) to a new value, which is absolutely unknown to the server.

4. Conclusion

In this paper, we have analyzed a series of protocols related to RFID ownership transfer, including a mutual authentication protocol, an ownership transfer protocol, and a secret update protocol. We emphasize that although many of our attacks are described with reference to the authentication protocol, they are also directly applicable to the ownership transfer protocol because the messages exchanged in these two protocols are identical. Our research shows that the mutual authentication protocol and the ownership transfer protocol do not guarantee most of the security properties required in their design (e.g., information and location privacy). Moreover, we have shown that the update protocol is vulnerable to a de-synchronization attack.

References


Julio C. Hernandez-Castro has a degree in Maths and a MSc in Coding Theory and Network Security. He has worked heavily in the past on the applications of artificial intelligence techniques (notably evolutionary computation) to Cryptography and Cryptanalysis. He got his Ph.D. in Computer Science from Carlos III University in Madrid in 2003 and was there till February 2009 when he was appointed Senior Lecturer in the School of Computing of Portsmouth University in the UK. He has published more than 30 papers in international journals and more than 50 in international conferences. He is a keen chess player.

Juan M. E. Tapiador is Research Associate in the Department of Computer Science Department at the University of York, UK. He holds a M.Sc. in Computer Science from the University of Granada (2000), where he obtained the Best Student Academic Award, and a Ph.D. in Computer Science (2004) from the same university. His research interests include cryptography, steganography and computer security.

Tieyan Li is a senior researcher at Institute for Infocomm Research (I²R, Singapore) from Oct. 2001. He obtained his Ph.D. degree in 2003 at School of Computing, National University of Singapore. Dr. Li is experienced in practical system developments such as networking, system integration and software programming. He is also active in academic security research fields with tens of journal and conference publications and several patents. Currently his areas of research are in applied cryptography and network security, as well as security issues in RFID, sensor, multimedia and tamper resistant hardware/software, etc. Dr. Li has served as the PC member and reviewer for a number of security conferences and journals.

Yingjiu Li is a tenure-track assistant professor working in the area of information security. He has published more than 60 research papers in journals and conferences including IEEE Transactions on Dependable and Secure Computing, Journal of Computer Security, IEEE Symposium on Security and Privacy, ACM Conference on Computer and Communications Security, USENIX Security Symposium, ACM Conference on Information, Computer and Communications Security, and European Symposium on Research in Computer Security. He has served in more than twenty conference program committees over the past three years including the program committees for the IEEE Symposium on Security and Privacy in 2007 and 2008. Yingjiu Li is a senior member of the ACM. The URL for his web page is http://www.mysmu.edu/faculty/yjli/.