A Privacy-friendly RFID Protocol using Reusable Anonymous Tickets

Author(s):
Asadpour, Mahdi; Torabi Dashti, Muhammad

Publication Date:
2011

Permanent Link:
https://doi.org/10.3929/ethz-a-006804648

Rights / License:
In Copyright - Non-Commercial Use Permitted
A Privacy-friendly RFID Protocol using Reusable Anonymous Tickets

Mahdi Asadpour
ETH Zurich, Switzerland
Email: mahdi.asadpour@tik.ee.ethz.ch

Mohammad Torabi Dashti
ETH Zurich, Switzerland
Email: mohammad.torabi@inf.ethz.ch

Abstract—A majority of the existing privacy-friendly RFID protocols use the output of a cryptographic hash function in place of real identity of an RFID tag to ensure anonymity and untraceability. In order to provide unique identification for the tags, these protocols assume that the hash functions are collision resistant. We show that, under this assumption on the hash functions, a substantial number of the existing protocols suffer from a traceability problem that causes distinguishing a tag from another.

We propose a scalable privacy-friendly RFID protocol and describe its design and implementation issues. Our protocol substitutes the hash functions used for identification with anonymous tickets, thus avoiding the aforementioned traceability problem. The anonymous tickets are reusable. They nevertheless identify tickets, thus avoiding the aforementioned traceability problem. We show that the protocol is scalable, and compare its storage and computational requirements to some existing protocols. We formally prove the security requirements of our protocol, and mechanically analyze some of its requirements using the model checker OFMC.

Keywords: Privacy, Anonymity, Untraceability, RFID, Formal Proof, Automated Verification.

I. INTRODUCTION

a) Context.: The Radio Frequency IDentification (RFID) is an identification mechanism which relies on radio waves for communication. In order to identify an object, an electronic tag is attached to the object. The tag is powered typically by a reader. The reader communicates with the tag, and with the help of a back-end server identifies the object. An RFID system consists of tons of tags, many tag readers and a powerful back-end server. RFID tags are inexpensive: they have recently been embedded into many products, including big bank notes, student cards, passports, and library books [3]. Such inexpensive tags however come with very limited capabilities: they are typically limited to computing hash functions and XOR operations, and they do not possess pseudo-random number generators. The tags communicate with the readers over insecure media. The communication link between the readers and the back-end server is however often assumed to be secure. A common assumption is that each tag shares a secret key with the back-end server; the key is kept in a database on the server side. The back-end database should therefore be able to handle a huge number of requests by tags. Thus, scalability is an important requirement which should be taken into account appropriately for RFID protocols.

A person who carries a device equipped with RFID tags is potentially traceable by an adversary. This would violate the person’s privacy. Therefore, designing privacy-friendly RFID protocols is a major concern before this technology becomes more widespread in everybody’s life. Privacy-friendly in this context means that (1) the identity of the tags cannot be inferred from the protocol messages (i.e. anonymity), and (2) two tags cannot be linked (i.e. untraceability); see [15]. These notions are made precise in the following sections. Moreover, since RFID tags are not tamper resistant, an adversary might be able to physically access a tag and read the content of its memory and find out all its secret data. Therefore, forward untraceability is another requirement for RFID protocols: it must be unfeasible to decipher or link past communicated messages, even if all the secret data of the tags is known by the adversary at present.

b) Contributions.: Our contributions are threefold:

1) We demonstrate a traceability attack for a number of the existing RFID protocols that use output of a hash function to identity RFID tags. In the protocols susceptible to this attack, a tag replies back to every request (from a rogue or a genuine reader) with the same hash output as long as the protocol is not entirely and successfully finished; we call these protocols group one. The hash output differentiates a tag from others: if the same hash output is seen again, it is definitely from the same tag, due to the collision resistance property of hash functions. We remark that if a small probability of hash collision would be allowed in practice, the protocols of group one would not guarantee unique identification of tags, hence resulting in, e.g., monetary losses: tag A would be charged with a purchase of tag B.

2) We introduce a privacy-friendly RFID protocol which, instead of hash functions, uses anonymous tickets for identifying RFID tags. Anonymous tickets uniquely identify a tag and are reusable (after being revoked). Therefore, the same ticket may well belong to two different tags in different times, hence alleviating the traceability problem of group one mentioned above. With respect to the scalability issue, the proposed protocol requires significantly less amount of storage compared to a second group of protocols which we call group two. In the protocols of the group two, hash functions are applied on tag identities along with random numbers.
These protocols are not prone to the traceability attack mentioned above. They however require a significant amount of storage on the back-end server; e.g. see [1].

3) We formally prove the security requirements of our proposed protocol in a formal model adopted from [4]. In order to precisely capture the aforementioned traceability attack, we extend the model of [4] with a notion of application untraceability. We also perform an automated formal analysis of the protocol using the model checker OFMC [6].

c) Structure of paper: The rest of this paper is organized as follows. Section II discusses our related work. In Section III, we introduce our protocol including message exchanges between a tag and the back-end server as well as its database design. Section IV gives a formal proof of the security requirements by introducing a new notion called application untraceability. In Section V we deal with the automated analysis of the security properties using OFMC. Section VI gives a performance analysis of the proposed protocol in terms of storage and workload on the tags and the back-end server. Section VII concludes the paper.

II. RELATED WORK

In the following, we categorize related works in two major groups. Both of the groups use output of a hash function in place of identity of a tag. (See [7] for a complete list of references.)

A. Group One

Protocols in group one (as depicted in Figure 1(a)) normally use \( Y = H(X) \) (\( H \) is usually a hash function) in order to hide \( X \), which is the real identity of a tag. \( X \) is different and unique for every tag, and the value of \( X \) remains constant as long as the entire protocol is not executed. For some of the protocols including the work of Henrici and Muller [11], \( X \) is updated to \( H(X) \) after successfully running the whole protocol.

In such a protocol a tag responds with \( H(X) \) in the first round to every request, no matter who is requesting, since there is no server authentication in place. Therefore, if an adversary having a rogue reader just performs the first part of this protocol without going on to the next step (see Figure 1(b)), she will always receive the same value \( Y \) as long as the value of \( X \) is not updated on the tag (or an adversary deliberately prevents it from updating). The main problem is that due to the collision resistance property of hash functions (as though it is infeasible to find two different tags with different \( X \)s having the same \( Y \)), the adversary by observing the same \( Y \) can infer that it is definitely the same tag.

So imagine an adversary has several readers installed in crowded locations and just requests tags for the first response and nothing more. She can collect all the data in several places and then cross-check to see for example this tag was previously in this place and so on. This way the adversary is able to trace a particular tag. If in an application updates rarely happen on RFID tags, the problem aggravates. Keeping in my mind that for most applications people always tend to carry RFID enabled devices such as credit or student cards, but occasionally use them to get a service (and thus update its \( X \)).

On the other hand, even if we argue that a collision in hash functions may happen in practice and implementation, another critical problem arises: the non-uniqueness of tag identities. Because of this collision in some monetary applications [10], the server may charge a wrong person.

B. Group Two

Protocols in this group normally use \( H(X_n) \) in which \( X_n \) is changed whenever a tag is interrogated, no matter whether the entire request-response protocol is successfully executed or not. Examples of this group include [13] and [1]. The authors of [1] claim that their work is the only protocol that achieves high level of privacy having constant time burden on the back-end database. In what follows, we focus on their protocol instead of the whole group.

In the protocol of [1] (Figure 2), every time a reader sends a random number \( r \) to a tag, the tag increments its counter \( c \) and replies back with \( H(\Psi || c) \) (\( H \) is a cryptographic hash function; \( \Psi \) is a pseudonym in place of identity; \( || \) means concatenation) and \( r' = H(0 || \Psi || c || k || r) \) (\( k \) being the secret key, this hash is used for tag authentication).

At the back-end database side, the server during database initialization has pre-computed all possible \( H(\Psi_i || c) \) for all available pseudonyms \( S \) (which this number should be more than the number of tags \( N \)) and a maximum counter value \( C \). For managing this huge chunk of data, the database is divided into three parts M-I, M-II and M-III. M-I is indexed based on \( s \) most significant bits of pseudonyms and contains pointers to mini tables in M-II (i.e. of the size \( O(2^s) \)). M-II consists of mini tables each of which storing all the \( H(\Psi_i || c) \) with the same \( s \) most significant bits (i.e. of the size \( O(SC) \)). Alongside each hash value, M-II maintains a pointer to an entry in M-III. The last part, M-III keeps information about all pseudonyms pointed by M-II (i.e. \( O(S) \)). Beside the number of pseudonyms/tags in the system \( S \), the value \( C \) also plays a big role in efficiency and storage of the back-end database.

In the reply from the server, a new pseudonym \( \Psi' \) is returned back which is encrypted using XOR \( (\oplus) \): \( H(2 || \Psi || k || r') \oplus \Psi' \). This message is accompanied with \( H(1 || \Psi || k || r') \) for...
server authentication) and \( H(3 \parallel \Psi' \parallel k \parallel r') \) (for message integrity).

Their proposed protocol suffers from the following weaknesses:

1) If the maximum value for counter \( C \) is small, e.g. 100, the protocol becomes like the first group and similar attack is applicable to it. Namely an adversary can interrogate a tag \( C \) times and force the tag to reuse its previous identities. This is practical due to the fact that nowadays tags are fast and can respond many requests per second. For example, a passive EPC tag responds to request of a reader in the order of 4 milliseconds [16], therefore with \( C = 100 \) all identities are exhausted in 0.4 seconds.

2) If \( C \) is set to a big value, it has big impact on the storage size at the back-end database. As also noted by [5], if \( C \) is 1000 (based on an example brought by the authors) the designed table size will be at least 40 terabytes which is not cheaply practical. (On the other hand, some might argue that even with \( C = 1000 \) still the above mentioned attack is practical: With the same EPC tag, all 1000 identities could be over in 4 seconds, and more importantly this could be done in multiple runs.)

3) On the other hand, a big value for \( C \) influences drastically the search time, especially on M-II and M-III parts. In M-II table there might be multiple entries so the server should linearly look up all the values. Also after successfully assigning a new pseudonym to a tag, all its previous pseudonyms should be updated in M-III, which requires roughly \( 2C \) update queries (if desynchronization resilience is supported too). Following up the above example \( C = 1000 \), it means 2000 database queries per each request which is obviously costly.

4) Similar to the first group, since the authors use the output of a hash function in place of identity of a tag, it may suffer from non-unique identification (as also discussed by the authors [1]), if the hash function is not collision-resistant.

III. OUR PROTOCOL

In this section we describe our proposal for a privacy-friendly RFID protocol, with constant time identification.

We assume that in RFID system we have RFID tags, readers and a back-end server/database. We also assume that in the current system there is a secret key shared between each tag and the server, and hence there is initially a table of tuples \((ID, k)\) \((ID\) is identity of a tag and \(k\) the shared key\) at the database. At the tag side we keep \( ID \) and \( k \) too. For the key size, in practice 128-bit keys are likely to give sufficient security [14].

Our main goal is to hide \( ID \) from an adversary in order to achieve anonymity. Alongside the tuple \((ID, k)\) we add \( T_b \) as a randomly assigned ticket. We reserve two tickets for every tag for desynchronization resilience purposes, and the realization of these tickets could be simply numbers from 1 to \( 2N + \delta \), given \( N \) the number of all tags and \( \delta \) the level of privacy we demand (the bigger the value, the stronger the privacy). These are made precise in the following.

We assume that an adversary can eavesdrop on the radio link between a tag and a reader, can modify messages, can add messages of her own, is able to block some messages from a tag to reader or vice versa, and can corrupt a tag acquiring its private data (a formal model will come later). However, distinctions by physical layer behavior of tags [16] are out of the scope of our work.

A. Protocol Steps

This section is devoted to describing the protocol steps. Here, we aim to hide the identity from every entity (including the reader), except the tag itself and the back-end server. This gives us a stronger privacy (i.e. end to end privacy). As discussed before, we shall also consider a tag impersonation/corruption attack as RFID tags are not tamper resistant. As shown in Figure 3, the protocol steps are as follows:

1) The reader interrogates a tag with a random challenge \( r_1 \).

2) The tag first increments the internal counter \( n \), and calculates \( r_2 = H(k \parallel n) \) and \( r_3 = H(k \parallel T_b \parallel r_1 \parallel r_2) \). \( H \) is a public one-way hash function, and \( \parallel \) means concatenation. The tag sends the anonymous ticket \( T_b \) (instead of \( ID \)) along with \( r_2 \) and \( r_3 \) to the reader. The term \( r_2 \) mainly acts as a random number (generated by means of feeding a counter \( n \) and the secret key in the hash function), and \( r_3 \) mainly acts as a tag authenticator for the server.

3) The reader then forwards the message as well as the random challenge \( r_1 \) to the back-end server via a secure channel.

4) Using \( T_b \) the server is able to obtain the corresponding record of this tag, i.e. \( ID \) and \( k \). (Details of database operations are explained below.) Using \( k \) and the received \( r_1 \) and \( r_2 \), the server is able to verify \( r_3 = H(k \parallel T_b \parallel r_1 \parallel r_2) \) and hence authenticate the tag. The server randomly chooses a new ticket from available/free ones, say \( T_c \), and generates a session key \( H(k \parallel T_b \parallel r_3) \) for encryption.

5) For our encryption, we use a simple XOR operation \((\oplus)\). The server encrypts the new anonymous ticket \( T_c \) with
the session key \( H(k \parallel T_b \parallel r_3) \), i.e. \( T_c \oplus H(k \parallel T_b \parallel r_3) \) (for integrity checking) to the reader. The reader subsequently forwards them to the corresponding tag. The server on the other hand updates two records associated to this tag in its database: for the one next to \( T_c \) it increments the server-side counter \( m \) and stores \( H(k) \) in place of \( k \) (for forward untraceability purposes) in corresponding fields. The value of \( m \) typically represents how many requests have been made so far by this particular tag, it nevertheless has more applications (briefly described later).

6) The tag uses \( r_3 \) and \( k \) to compute \( H(k \parallel T_b \parallel r_3) \), the so-called session key. Using this session key, it can de-cipher the received \( T_c \oplus H(k \parallel T_b \parallel r_3) \) and obtain \( T_c \).

7) After successfully acquiring the new ticket \( T_c \), the tag invokes hash operation on the key and stores the outcome as the new key (i.e. \( k \leftarrow H(k) \)) in order to achieve forward untraceability (similar to the changes at the server side).

8) Finally the tag updates its tickets to \( T_c \), beside having new value for \( k \). Note that next time this tag is going to reply with \( T_c \) in the above protocol.

The counter \( n \) in the tag side, which is never transmitted, is employed to generate a pseudo-random number by means of \( k \) in a one-way hash function, i.e. \( H(k \parallel n) \). It is separate from the value of \( m \) in the server side, although these could be correlated to find out how many unsuccessful interrogations have happened on a particular tag (details in [2]).

B. Database Operations

We concentrate here on the back-end database to show that how we provide a constant time identification. In order to handle requests from tags, we maintain a ticket table and one free ticket pool:

1) **Ticket table**: Such a table already exists in RFID systems to retrieve the shared key \( k \) given an identity \( ID \). We only adjust it for our purpose (see Figure 4):

- **ticket** is now the primary index of this table and can point us to \( ID^1 \) and \( k \). Using each of ticket we can thus retrieve the real identity and the shared key in \( O(1) \), the fastest possible time in database context.
- The table has a column “ref.” (standing for reference), which associates a ticket of one tag to its previous ticket/record and vice versa. Again the **ticket** column is the primary key, so the table gives out \( ID \) if we provide one of the tickets \( T_a \) or \( T_b \), in \( O(1) \) too.
- **num** represents how many tickets are assigned to a tag so far.

2) **Free tickets pool**: It contains those tickets not assigned to any tag (in total of size \( \delta \), which was assumed in the beginning of the section). It can be implemented using a simple table of free tickets or a link list that is efficiently permutable in order to randomize the assignment.

In the initialization phase, the server fills the **Ticket table** with two randomly assigned tickets to every tag (e.g. \( T_a \) and \( T_b \) in Figure 4) while setting the other fields correctly (e.g. \( \text{num} \) is 0 for the first ticket). Therefore, in the very first run a tag responds with \( T_a \) and the server could immediately send back \( T_b \) in the encrypted format (as explained before) with no further operation, as though we already have \( T_b \) reserved for this tag.

![Fig. 3. Protocol steps.](image-url)

1We can remove \( ID \) from both the tag and the database sides, but we choose to keep it in the case that it is needed by applications on upper layers (removing it does not influence ours).
For the later runs, steps are (in respect to steps described in Figure 4):

1) Using $T_b$ on the Ticket table, the server is able to quickly obtain the corresponding $ID$, $k$ and previous ticket $T_a$ by a single lookup query. By checking the row belonging to $T_a$ (another query) the server can ensure that $T_b$ is the recent ticket by comparing the value of $num$ fields (the bigger the value, the newer the ticket). By $k$ and $T_b$, the server validates the correctness of the received values.

2) The server randomly chooses a new ticket from the Free tickets, say $T_c$. This random selecting operation is important as it removes any link between the current ticket $T_b$ and the new ticket $T_c$. We assume that a random number generator is available at the server, as the server side is not limited.

3) $T_a$ is released and returned to the Free Tickets pool.

4) The server updates the Ticket table accordingly: $T_a$ is updated to the new anonymous ticket $T_c$ (as primary index), $num$ field is incremented by one, and the ref. columns of $T_b$ and $T_c$ are pointed to each other. More importantly, the key $k$ becomes $H(k)$ in the new row.

5) As mentioned before, the server finally generates a session key $H(k || T_b || r_3)$ and returns back $T_c$ in encrypted format alongside a $H(k || r_3 || T_c)$ for integrity checking.

C. De-synchronization Resilience

We keep two rows for every tag in the back-end database in case the response of the server (i.e. carrying the new ticket $T_c$) does not reach to the tag, due to noise, denial-of-service attacks, etc. In such a case the server expects that the next ticket from this tag will be $T_c$ but the tag will start over with $T_b$. After receiving a request with $T_b$, the server obtains the corresponding ticket $T_c$ (using ref. field in the Ticket table) and by checking the value of num field realizes that in fact the tag did not receive its new ticket. So, without going through the new ticket assigning procedure, just the same as before it returns back $T_c$ with no update on the Ticket table. In this way, the proposed protocol handles de-synchronization problems.

D. Unique Identification

Uniquely identifying a tag by a server is also an important issue that may cause serious performance and security problems: assume that RFID tags are embedded into monetary cards, therefore mistaking a person with another the system may charge a wrong person. For some protocols there are easy solutions, but still as discussed by some of the related works (for example [1] and [10]), using just the output of a hash function does not guarantee the unique identification.

In our scheme we guarantee the unique identification by putting the server in charge of ticket/identity assignment and sending an anonymous unique ticket for identification. The server leverages a well-designed database system to deal with the issue: It has a series of unique tickets in the system; it keeps the assigned tickets in Ticket table and non-assigned tickets in Free tickets pool; when the server updates identity of a tag, it releases one assigned ticket to free ones and retrieves one from the free tickets into the assigned ones. Therefore no ticket is assigned to two different tags at the same time, thus the unique identification is preserved (inductively). (As mentioned, every ticket is reassigned to different tags over time, thus untraceability is enhanced as well.)

IV. Formal Security Analysis

Privacy of tags and mutual authentication between a tag and the server are our main security requirements. In order to formally define and prove them, we borrow three notions
from [4] (following a similar sketch of [1]) as well as defining a new notion called applicational untraceability.

For the sake of brevity we do not describe every detail including the proofs here, and refer the interested reader to [2].

A. Adversarial Model

The adversary A is modeled as a polynomial-time algorithm with control over the communication channel between a tag T and a reader R. We assume A has access to the following set of oracles [4]:

- **Query (T, m1, x2, m3):** this query models A sending m1 to T; receiving the response x2; and subsequently replying with m3. In other words, it models the adversary's ability to interrogate tags.
- **Send (R, x1, m2, x3):** this query models A receiving x1 from the reader R; replying with m2; and receiving response of R, x3. In other words, it models the adversary's ability to act like a tag.
- **Execute (T, R):** this query models A executing an instance of the protocol between T and R, and eavesdropping on the channel. In other words, it models the adversary's ability to monitor the channel between a tag and a reader.
- **Block (A):** this query models A blocking any part of the protocol. In other words, it models the adversary's ability to launch a denial of service attack.
- **Reveal (T):** this query models A obtaining the content of secret parameters of T. In other words, it models the adversary's ability to physically corrupt a tag and obtain its secrets.

A can call the oracles Query, Send, Execute, and Block any polynomial number of times but Reveal oracle just once on the same tag (no point in calling Reveal multiple times on one tag).

B. Untraceability

In the following, we briefly discuss universal, forward and existential untraceabilities of our protocol as well as defining a new untraceability notion.

1) **Universal Untraceability:** In RFID systems, a tag is universally untraceable if two of its responses, separated by a successful authentication with a valid reader, cannot be correlated [4]. This is modeled by the following game between the challenger C as the RFID system and an adversary A [1]:

1) C selects two tags T1 and T2, and a valid reader R.
2) A calls oracles Query, Send, Execute, and Block on T1, T2, and R for a number of its choice.
3) A notifies C, after she is done with calling oracles.
4) C successfully carries out an instance of the protocol on T1, T2 with R.
5) C randomly selects one of the tags as T.
6) A calls oracles Query, Send, Execute, and Block on T and R.
7) A outputs her guess T', and wins the game if T' = T.

Theorem 1: In our proposed protocol, tags are universally untraceable.

2) **Existential Untraceability:** In RFID systems, a tag is existentially untraceable if two of its responses, not necessary separated by a successful authentication with a valid reader, cannot be correlated [4]. Existential untraceability is modeled by the following game between C and A (the adversary in this definition is stronger than the one in [1]):

1) C selects two tags T1, T2 and a valid reader R.
2) A calls oracles Query, Send, Execute, and Block on T1, T2, and R for a number of its choice.
3) A notifies C, after she is done with calling oracles.
4) C randomly selects one of the tags as T.
5) A calls oracles Query, Send, Execute, and Block on T and R.
6) A outputs her guess T', and wins the game if T' = T.

This definition however implies that the adversary knows mutual authentication is not carried out on a tag after she notifies the challenger, which this is not realistic. Our protocol does not provide existential untraceability (similar to the group one).

Theorem 2: In the proposed protocol, tags are not existentially untraceable.

3) **Applicational Untraceability:** A problem with the definition of both existential and universal untraceability games is that we implicitly assume the adversary knows, or is given, the side channel knowledge of whether an update happens on a tag (universal) or not (existential). This assumption does not comply with the reality in RFID applications, and we believe that the existential untraceability is much stronger than what is needed.

In order to have a more realistic model, we define another untraceability notion called "Applicational Untraceability" as follows: Tags in an RFID system are said to be applicationally untraceable if an adversary cannot track a tag based on its responses to multiple interrogations, if the tag has been able to accomplish mutual authentication with an authorized reader with a probability p. The probability p can be varied for different RFID applications and thus makes the model more adaptable. Applicational untraceability is modeled by the following game between C and A:

1) C selects two tags T1, T2 and a valid reader R.
2) A calls oracles Query, Send, Execute, and Block on T1, T2, and R for a number of its choice.
3) A notifies C, after she is done with calling oracles.
4) C carries out an instance of the protocol with probability p on T1, T2 and R for polynomial number of times of its choice.
5) C randomly selects one of the tags as T.
6) A calls oracles Query, Send, Execute, and Block on T and R.
7) A outputs her guess T', and wins the game if T' = T.

In fact, the applicational untraceability can be viewed as a general case for both universal and existential untraceabilities: if the update probability p goes toward 1 (always update) we
get universal untraceability, and if $p$ goes toward $0$ (no update) then we get existential untraceability. Using this probabilistic approach, we remove the update certainty of previous definitions. The following theorem concerns applicational untraceability in our protocol.

**Theorem 3:** In the proposed protocol, tags are applicationally untraceable.

Considering this untraceability for the other two groups (see our related work), we have:

1. For the group one, if $A$ observes the same hash value, it is definitely the same tag and thus no update happened (with $1 - p$ probability) and $A$ can guess it right with probability $1$. Or $A$ sees a different hash value and thus an update has happened (with $p$ probability) and hence $A$ cannot guess it right, altogether $(1 - p) \times 1 + p \times 0 = 1 - p$. Considering also a random guess, $A$ has a winning probability of $\max\{1-p, \frac{1}{2}\}$ in the applicational untraceability.

2. For the group two, since $A$ always observes different hash values due to the update probability $p$ of $1$, the winning probability would be $1 - p = 1 - 1 = 0$. Considering also a random guess, she has $\max\{0, \frac{1}{2}\} = \frac{1}{2}$ winning probability. Therefore, $A$ does not have any advantage over a random guess, which this deduction is consistent with what achieved by this group, e.g. in [1].

4) **Forward Untraceability:** In RFID systems, forward untraceability means that an adversary capturing secret information of the tag cannot correlate the tag with its responses before a successful mutual authentication with a valid reader [4]. Forward untraceability is modeled by a game similar to what we see in the universal untraceability, except in the last phase an adversary calls the oracle $\text{Reveal}(T)$ and guesses accordingly [1].

**Theorem 4:** In the proposed protocol, tags are forward untraceable.

**C. Mutual Authentication**

In RFID systems, mutual authentication is achieved if a reader can authenticate a tag based on the message it receives (tag authentication), and a tag can authenticate a reader based on the message it receives (reader authentication). The adversary’s attempt to authenticate herself to a reader (tag) as a valid tag (reader) is modeled by the following game [1]:

1. $C$ randomly selects a tag $T$ and a valid reader $R$.
2. $A$ calls oracles $\text{Query}$, $\text{Send}$, $\text{Execute}$, and $\text{Block}$ on $T$ and $R$ for a number of its choice.
3. $A$ notifies $C$, after she is done with calling oracles.
4. $A$ calls the oracle $\text{Send}(\text{Query})$ to impersonate a tag (reader).
5. If $A$ is authenticated as a valid tag (reader), $A$ wins the game.

The following theorem deals with this issue in the proposed protocol.

**Theorem 5:** The proposed protocol achieves mutual authentication.

**V. Automated Verification**

Automated verification of security protocols can help to discover design mistakes, which are typically hard to find due to systems complexity. If such a security design is verified successfully by an automated tool, it increases the confidence in the system [8]. Verifying privacy properties with the available automated tools however has some difficulties. To name a few, some do not check indistinguishability of traces of multiple runs of a protocol (which we require for the purpose of verifying untraceability), and some others do not support XOR operations (which is employed in many RFID systems).

Here we use OFMC tool [6] that supports XOR operations, although only it supports single traces and hence makes modeling of privacy impossible [8]. We aim to verify privacy-friendly capabilities as much as possible by means of performance approximation, as well as the mutual authentication. The goal here is to re-define some of the security requirements in a way feasible for automated verification.

**A. Privacy**

To provide privacy, the following information should not be inferred from the protocol messages: 1-identity of tags (which means anonymity), and 2-linkability of two tags (which means untraceability) [15].

In OFMC, we model message exchange of our protocol using two agents: one variable agent $T$ representing tags and a static agent $s$ representing a reader (or accurately one backend server). OFMC carries out model checking against the Dolev-Yao intruder [9] represented by $i$, which has the same power as our adversary but without a Reveal oracle (see section IV-A). The goal is to transmit the next random ticket from $s$ to $T$ in a secure and authentic way, equivalent to having a secure channel in place (i.e. $\ast \rightarrow \ast$ channel). In other words, this goal principally aims to verify inability of an intruder in acquiring the next ticket of a tag. Not knowing the first ticket of a tag (which is assumed by the system), and having the above property, we inductively reason that an intruder cannot expose different tickets of a tag.

Details of the proposed model are given in [2]. As the result, OFMC gives us a “safe” status with bounded number of sessions (three) in 8.04 seconds (on WinXP system with Core(TM)2 CPU 2.00GHz).

In order to further analyze forward untraceability with OFMC, we have to slightly change our model as this tool only supports single traces. The goal here is to show that knowing a run, the intruder is not able to disclose an anonymous identity used in previous runs. So we put a next running of the protocol and all its messages in the intruder’s knowledge set and check whether it is possible to reveal a previous identity or not. Details are mentioned in [2]. As the result, OFMC gives us a “safe” status with bounded number of sessions (three) in 13.20 seconds (on the same computer).

**B. Mutual Authentication**

In modeling the above properties, we used the notation of a secure channel representing by $s \ast \rightarrow \ast T$, which means an
authentic and confidential channel between two entities [12]. In other words, T authenticates s on the message it receives via this channel (i.e. server authentication), and s ensures that only T can open the message.

On the other side, to further check the tag authentication, we add authentic channel of T ⨿ s (i.e. tag authentication) as a supplementary goal to our OFMC model. As the result, OFMC gives a “safe” status with 3 sessions in 17.12s on the same computer. Altogether, we have therefore checked the mutual authentication property with this tool as well.

VI. PERFORMANCE ANALYSIS

This section is devoted to the performance analysis of the proposed protocol.

The tag side of an RFID system should be fast and memory efficient. In our protocol, a tag performs (totally) five hash operations, one XOR operation, and one addition (incrementing n) per transaction. A tag also needs to store 4 items (although as mentioned ID could simply be removed). From a communication perspective, the whole protocol is executed in three rounds.

At the back-end server side, in our protocol all the database operations are being run on primary keys of tables (so to speak, the fastest possible in database context). Besides, some low cost operations such as generating random number, hash operations and XOR are not costly for most of the typical servers. Also we just need at most two update queries per each request: one to update the row belonging to the new ticket and one to update its previous one (kept for desynchronization resilience purposes). We recall that in Alomair et al.’s scheme 12C updates are needed, while a considerably large value for C (let say C = 1000) is required.

From the storage point of view, considering the fact that we already have a database of identity and keys (this is a common assumption in RFID systems), we add three columns to the Ticket table and double the number of its rows. With the example of 10^9 tags in the system, our scheme adds roughly 4 × 10^9 bytes to the database which is way less than one terabyte (i.e. 0.004 terabytes). Recall that for example in Alomair et al.’s scheme this value is 40 terabytes with the same number of tags, although Avoine et al. [5] propose a way to save up almost one-third of it.

VII. CONCLUSION

We point out that some of the existing privacy-friendly RFID protocols suffer from a traceability problem, as a tag outputs the result of a collision resistance hash function as a means of anonymous identification. The distinct and constant hash output of the tag can differentiate it from other tags in the system. In other words, the collision resistance property implies that whenever or wherever the same hash value is observed, it definitely belongs to the same tag. Also we argue that if we accept a small collision in the hash outputs, they still suffer from a non-unique identification problem. Altogether, we propose to revise the use of hash functions for the purpose of anonymous identifications in RFID systems: either they harm untraceability (if we assume perfect collision resistance) or they harm unique identification (if they are not collision resistant).

In order to strengthen untraceability, we introduce anonymous tickets in place of hash functions. We show that these tickets can uniquely identify a tag and since the server reuses and reassigns them randomly to other tags, an adversary cannot be sure it is the same tag by observing the same ticket. For handling these tickets efficiently at the back-end database side, we employ one free tickets pool to store all free/non-allocated tickets, and we extend one existing ticket table by adding anonymous tickets columns to it. As mentioned, this solution requires little storage overhead and only a few queries on the primary index in the back-end database.

With respect to security analysis, we formally show that the protocol guarantees universal, applicational and forward untraceabilities which are the major requirements in the context. We further model and analyze some privacy and security requirements with the automated tool OFMC.

Finally we assess the proposed protocol from a performance perspective, the load on each tag and the back-end database. Our protocol outperforms the group one of related works in particular in untraceability, and outperforms the group two in efficiency and scalability issues.

REFERENCES


