Towards a Practical Solution to the RFID Desynchronization Problem*

Gerhard de Koning Gans and Flavio D. Garcia

Institute for Computing and Information Sciences
Radboud University Nijmegen
P.O. Box 9010, 6500 GL
Nijmegen, The Netherlands
{gkoningg,flaviog}@cs.ru.nl

Abstract. Even though RFID technology has expanded enormously, this expansion has been hindered by privacy concerns. In order to prevent an adversary from tracking RFID tags and thus breaking location privacy, tags have to update their internal state with every authentication attempt. Although this technique solves the privacy problem, it has the side effect that tags and back office might desynchronize. This desynchronization can be caused by physical conditions or by adversarial intervention. If we look at consumer product identification, RFID labels and barcodes are bound to coexist for quite some time. In this paper we exploit this coexistence to reduce the workload at the reader/backoffice and allow re-synchronization. Concretely, we propose an authentication protocol that achieves correctness, forward-privacy under mild additional assumptions and synchronization in the random oracle model.

Keywords: RFID, barcodes, location privacy, forward-privacy, random oracle model.

1 Introduction

Over the last few years, the use of RFID technology has expanded enormously. It is currently deployed in electronic passports, tags for consumer goods, public transport ticketing systems, race timing, and countless other applications.

RFID technology have recently become popular as a replacement for traditional barcodes in the consumer supply chain. Even though RFID labels have indeed advantages over barcodes, they also have some drawbacks. On the one hand, RFID labels can be read faster than barcodes and have less restrictions on the physical positioning of the label. These advantages do not necessarily imply that barcodes will be replaced by RFID labels and are no longer needed. It is still useful to have some backup identification possibility. For

* Partially supported by the research program Sentinels (www.sentinels.nl), project PEARL (7639). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs.

S.B. Ors Yalcin (Ed.): RFIDSec 2010, LNCS 6370, pp. 203–219, 2010.

instance, when an RFID label breaks down it is still possible to switch to barcode identification. Barcodes are often printed right on the product (wrapping) and therefore are currently cheaper than an RFID label. Furthermore, the barcode system is deeply entrenched in many systems and complete replacement is not going to happen in the near future [WNLY06]. Actually, barcodes and RFID systems have to be used in parallel for many more years. On the other hand, the widespread use of RFID has raised various privacy concerns. Since most RFID tags will send a unique identifier to every reader that attempts to communicate with it, an adversary could build an "RFID profile" of an individual, i.e., the collection of unique identifiers of the RFID tags that the individual usually carries. This profile could be used to track this person, or to infer behavior such as spending or traveling patterns, jeopardizing this person's privacy.

If we focus on inexpensive EPC-like tags, think of the ones attached to a product in the supermarket, we observe that RFID tags are often used in parallel with barcodes, instead of replacing them. The combination of barcode and RFID label can be found on several products nowadays. Figure 1 shows an example of such a tag.

In this paper we exploit this duality by using a combination of barcode and RFID labels in order to get the best of each technology. On the one hand, flexible reading and unique identification, on the other hand the infeasibility for an adversary to track goods at will. We present a practical solution where both RFID label and barcode are combined in order to provide location privacy.

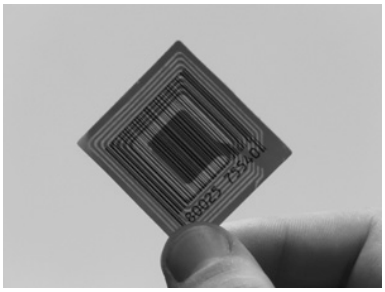

Fig. 1. Barcode and RFID label

Many privacy notions have been discussed in the literature but the notion of forward privacy is generally considered satisfactory [Vau07, BBEG09]. This privacy notion requires that an adversary who has control over the communication media should not even be able to tell whether two protocol instances involve the same tag or not. Moreover, even when all secret information in the tag is revealed to the adversary, this should not be able to link this tag with previously recorded protocol runs. In order to achieve such a strong security notion, it is necessary that the tag updates its state (using a one-way function) with every authentication attempt. This continuous updating might lead to desynchronization between the back office and the tag. This desynchronization can be both, induced by an adversary or simply due to physical conditions like the distance between tag and reader.

Related Work. A large number of protocols have been proposed in the literature that aim to achieve location privacy [JW05, Tsu06, BdMM08] and concretely forward-privacy [OSK +03, Vau07, BBEG09]. Unfortunately, many of these proposals turn out to be either impractical due to the resource-constrained nature of RFID or suffer from desynchronization. Achieving forward privacy

without using public key cryptography has shown to be a very challenging task. In fact, Vaudenay [Vau07] showed that having a forward private stateless RFID scheme implies key agreement, which is believed to be require public-key cryptography. Achieving forward-privacy with symmetric cryptography requires heavy workload on the reader side and these protocols often suffer from desynchronization. A distinguished example is due to Avoine [AO05], who proposed a scheme based on OSK [OSK +03] that achieves forward-privacy. Unfortunately this protocol suffers from desynchronization which has impact on availability. The scheme of Dimitriou [Dim05] is reminiscent of the Hash-Locking scheme of Weis [WSR +04] but it also suffers from desynchronization. For a complete survey of related work we refer the reader to [Jue06].

Our Contribution. This paper proposes a forward private RFID authentication protocol that incorporates a mechanism for re-synchronization. We exploit the coexistence of RFID and barcodes in the protocol design in order to achieve a more efficient search procedure on the reader side. The main idea of the protocol resembles that of OSK, except that we allow a limited and small number of failed authentication attempts. This reduces dramatically the search space on the reader side. Should this limit be exceeded, then the barcode allows the protocol to re-synchronize. This re-synchronization takes place within the authentication protocol itself so that it does not compromises privacy.

We propose a model for RFID privacy using provable security techniques, following the lines of [Avo05, Vau07, JW09, GvR10]. Within this model we define correctness, forward-privacy and synchronization. Finally, we show that our protocol satisfies all these security notions using the random oracle methodology.

Organization of the Paper. In Section 2 we briefly explain the desynchronization problem. Section 3 describes the system and adversarial models. Section 4 then provides definitions for security, (forward-)privacy, (strong-)correctness and desynchronization. Section 5 describes our protocol and Section 6 substantiates the security claims. Finally, Section 7 concludes the paper and discusses future work.

2 The Desynchronization Problem

Our goal is a practical RFID protocol that provides location privacy. The meaning of the adjective "practical" heavily depends on the resources and restrictions that are given. A good first attempt is the following protocol where a tag T sends the hash of its identity id concatenated with some random value r and r itself to a reader R.

$$\mathrm{T} \rightarrow \mathrm{R}: \ h(id, r), r$$

Assuming a perfect hash function and random number generator it is impossible for an eavesdropper to retrieve the identity id. This small protocol is more or less what was proposed as the Randomized Hash-Locking scheme by Weis et al.

in [WSR +04]. The reader is connected to a back-end where a database is maintained with all tag identities. The big drawback in this solution is in the search procedure. To look up a tag every identity needs to be hashed in combination with the random r. This drastically reduces the applicability of this solution to small systems with a limited number of tags.

Another well-known RFID protocol from the literature is OSK [OSK +03] where the tag identifiers are updated in every protocol run regardless whether it was a successful run or not. This is done by a hash chain where $h^i(x)$ means that x is successively hashed i times. In [CC08] it is shown that the OSK scheme is synchronizable since $D_{\mathcal{R}} = \infty$, $D_{\mathcal{T}} = 0$, $R_{\mathcal{R}} = \infty$ and $R_{\mathcal{T}} = 0$. This illustrates the fact that a resynchronizable protocol is not automatically efficient in its search procedure. For instance, a denial-of-service attack (DoS attack) might be induced by simply sending a random value to the reader.

Barcode Analogy. The protocol that is introduced in this paper can be best explained in analogy to the traditional and very successful barcode. RFID is used to automatically identify products and to process the gathered data. This can be used to track products along the supply chain in industry [Att07], the medical sector [WCO+07], libraries [MW04] and many other situations where barcodes are already employed.

A well known daily example of barcodes can be found in a shop. The cashier scans the barcodes of products that the customer wants to buy. From time to time the scanner might not be able to read a barcode. In such cases the cashier enters the serial number by hand using a keypad. This backup procedure costs more time and effort, but at the end the checkout procedure is far more efficient than it would be when every product was entered manually at default.

The number of times that the cashier has to fall back to the manual input procedure is very low. If this was not the case, the use of barcodes would be questionable. Actually, we face the same problem in privacy friendly RFID. Here, the tag and reader need to stay synchronized in some way. To the best of our knowledge, all attempts to design a protocol that keeps up with these discrepancies try to achieve this without any human intervention. Many proposals try to prevent desynchronization purely by means of the wireless link. This becomes a very hard task when, at the same time, an adversary is allowed to exhaustively query a tag. In practice desynchronization is a problem that should be handled, merely because it may also occur due to physical problems in the reading process. Now, recall the same shop as mentioned before but let the products be equipped with RFID tags. When a tag is no longer synchronized with a genuine reader and the system fails to identify a tag, we fall back to the use of a *second channel* which provides the reader with the needed identity. This identity can then be read from a barcode or serial number which is physically printed on the RFID tag. A protocol run in which a *second channel* is used to synchronize the tag and reader state is called a *synchronization run*. Since a *synchronization run*

involves additional actions apart from running the protocol it can be treated as a special instance of the protocol. In general, these special instances occur scarcely in practical settings. In this paper we further elaborate on a system like presented above.

3 System Model

Consider a scheme where readers have a secure communication channel with the back office. We assume that readers are single threaded, i.e., can only have one active protocol instance with a tag at a time. After running a protocol with a tag, the reader has an output that is typically the identity of the tag. New readers and tags can be added to the system at will. The formal definition follows.

Definition 1 (RFID scheme). *An* RFID scheme Π *consists of:*

- *a probabilistic polynomial-time algorithm* SetupSystem *that takes as input the security parameter* 1^η *and outputs the public key pair* (sk, pk) *of the system.*
- *a probabilistic polynomial-time algorithm* SetupReader *that takes as input the secret key of the system sk and outputs the initial state of the reader s and the reader's secret k.*
- *a probabilistic polynomial-time algorithm* SetupTag *that takes as input the secret key of the system sk and outputs the initial state of the tag s and the tag's secret k.*
- *a polynomial-time interactive protocol between a reader and a tag, where the reader returns* Output. Output *is typically the identity of the tag.*

An adversary is a probabilistic polynomial-time algorithm that interacts with the system by means of different oracles. The environment keeps track of the state of each element in the system and answers the oracle queries according to the protocol. Besides adding new tags and readers to the system and being able to communicate with them, an adversary can also corrupt tags. This models techniques like differential power analysis and chip slicing. By corrupting a tag an adversary retrieves its internal state.

Definition 2 (Adversary). *An* adversary *is a probabilistic polynomial-time algorithm that takes as input the system public key pk and has access to the following oracles:*

- CreateReader($\mathcal{R}$) *creates a new reader by calling* SetupReader(sk) *and updates the state of the back-office. This new reader is referenced as* $\mathcal{R}$.
- CreateTag($\mathcal{T}$) *creates a new tag* $\mathcal{T}$ *by calling* SetupTag(sk) *and updates the state of the back-office. This new tag is referenced as* $\mathcal{T}$.
- CorruptTag($\mathcal{T}$) *returns the internal state s of the tag* $\mathcal{T}$.
- Launch($\mathcal{R}$) *attempts to initiate a new protocol instance at reader* $\mathcal{R}$. *If* $\mathcal{R}$ *has already an active protocol instance then* Launch *fails and returns zero. Otherwise it starts a new protocol instance and returns one.*

- Send(m, A) *sends a message* m *to the entity* A *and returns its response* m'*. The entity* A *can either be a reader* $\mathcal{R}$ *or a tag* $\mathcal{T}$*.*
- Result$(\mathcal{R})$ *outputs whether or not the output of the last finished protocol instance at reader* $\mathcal{R}$ *is not* $\perp$*, i.e.,* Output $\neq \perp$*.*

Definition 3. *We denote by* $\mathcal{O}$ *the set of oracles* {CreateReader, CreateTag, CorruptTag, Launch, Send, Result}*.*

4 Security Definitions

This section elaborates on the security and privacy definitions from the literature, much of it is standard.

The main goal of an RFID system is security, which means that readers are able to authenticate legitimate tags. Throughout this paper we focus on privacy. For the sake of self containment, we include here the following security definition which is an adapted version of the security definition proposed in [Vau07].

Definition 4 (Security). *An RFID scheme is* secure *if for all adversaries* $\mathcal{A}$ *and for all readers* $\mathcal{R}$*, the probability that* $\mathcal{R}$ *outputs the identity of a legitimate tag while the last finished protocol instance at reader* $\mathcal{R}$ *and this tag did not have any matching conversation, is a negligible function of* η*. Matching conversation here means that* $\mathcal{R}$ *and the tag (successfully) executed the authentication protocol.*

Next we define privacy composing the definitions of Juels and Weis [JW09] and Vaudenay [Vau07] since each of them has its advantages: the former is indistinguishability based, which makes it more practical; the latter has the drawback of being simulation based but is stronger and allows for a variety of adversaries with custom capabilities. Privacy is defined in an IND-CCA like fashion where the adversary tries to win the privacy game. In this game, the environment creates system parameters by calling SetupSystem. Then it gives the public key of the system pk to the adversary $\mathcal{A}_0$. This adversary has access to the set of oracles $\mathcal{O}$. Eventually, $\mathcal{A}_0$ must output two uncorrupted challenge tags $\mathcal{T}_0^\star$ and $\mathcal{T}_1^\star$. Then, the environment chooses a random bit b and gives the adversary $\mathcal{A}_1$ access to $\mathcal{T}_b^\star$. At this point, the original references to $\mathcal{T}_0^\star$ and $\mathcal{T}_1^\star$ are no longer valid. Again, the adversary has access to all oracles $\mathcal{O}$. Finally, the adversary outputs a guess bit b'. The adversary wins the game if $b = b'$. The formal definition follows.

Definition 5 (Privacy game).

$\boldsymbol{Priv\text{-}Game}_{\Pi,\mathcal{A}}(\eta)$:
$(sk, pk) \leftarrow \text{SetupSystem}(1^\eta)$
$\mathcal{T}_0^\star, \mathcal{T}_1^\star \leftarrow \mathcal{A}_0^{\mathcal{O}}(pk)$
$b \leftarrow \{0, 1\}$
$b' \leftarrow \mathcal{A}_1^{\mathcal{O}}(\mathcal{T}_b^\star)$
winif $b = b'$.

The challenge tags $\mathcal{T}_0^\star$ and $\mathcal{T}_1^\star$ must be uncorrupted, which means that no CorruptTag($\mathcal{T}_{\{0,1\}}^\star$) *query has been made. Adversaries implicitly pass state.*

In general, it is hard to define a realistic adversarial model as different applications have different requirements. Following the lines of Vaudenay [Vau07], we consider different classes of adversaries depending on their capabilities. The notions of forward, weak and narrow adversaries are due to Vaudenay. The notion of thin adversary is introduced in this paper to handle protocols that use a *second channel*. Intuitively, a *forward* adversary is an adversary that observes communication between tags and readers and later on acquires one of these tags and tries to link it with some of the past sessions, compromising its privacy. If the adversary succeeds to do so, with non-negligible probability, we say that is a *winning* adversary. A *weak* adversary is an adversary that is unable to corrupt tags. In real life scenarios it is often realistic to assume that an adversary can see the outcome of an authentication attempt. For instance, this is the case of transport ticketing systems where an adversary could observe whether the gate of the metro opens or not, for a specific tag. An adversary that is unable to do so is called *narrow*. In line with the *narrow* adversary we introduce the *thin* adversary. A *thin* adversary cannot see additional information that is provided to the reader. Think for example of additional identifying information to make the search procedure more efficient.

Definition 6 (Types of adversaries). *A* forward *adversary is an adversary that has access to all oracles $\mathcal{O}$. A* weak *adversary cannot perform any* CorruptTag *query at all. A* narrow *adversary does never query the* Result *oracle. Finally, we introduce the notion of* thin *adversary which, like the* narrow *adversary, does never query the* Result *oracle. Furthermore, a* thin *adversary cannot see* synchronization runs *and thus cannot see protocol runs where information is used that is obtained by the* second channel.

Remark 1. Note that this notion of forward adversary is stronger than the one proposed by Vaudenay and closer to the notion of Juels and Weis.

Definition 7 (Privacy). *Let C be a class of adversaries in $\{$forward, weak, narrow, thin$\}$. An RFID scheme is said to be C-*private *if for all probabilistic polynomial-time adversaries $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1) \in C$*

$$\mathbb{P}[\boldsymbol{Priv\text{-}Game}_{\Pi,\mathcal{A}}(\eta)] - \frac{1}{2}$$

is a negligible function of η.

In our definition of *desynchronization* we follow [CC08]. Consider a valid tag which is referenced by *id*. Let its corresponding key k be denoted k_{id}. Every tag is initialized by SetupTag using the initial key k_{id}^0. Then, k_{id}^i denotes the tag key after i updates. Since both reader and tag keep track of their own instance of k_{id}, we write rk_{id} for the reader instance and tk_{id} for the tag instance of k_{id}. Usually, $rk_{id} = tk_{id} = k_{id}^*$, but when the tag and reader are no longer synchronized we

have $tk_{id} = k_{id}^i$ and $rk_{id} = k_{id}^j$ where $i \neq j$. In order to allow reasoning about desynchronization, first *correctness* is defined, then the definition of a *strong correctness game* follows. In its turn this game is used to define *strong correctness*. Finally, it is defined when an RFID scheme can be subject to *desynchronization*.

Definition 8 (Correctness). *An RFID system is said to be correct when the reader outputs $\perp$ after an authentication protocol π with a non-legitimate tag and outputs the tag id after an authentication protocol π with a legitimate tag.*

The **Strong Correctness Game** is comparable to the **Privacy-Game** and its setup is also indistinguishability based. Again, the challenger generates system parameters by calling SetupSystem. Then, the public key pk is given to an adversary $\mathcal{A}$ which has access to the set of oracles $\mathcal{O}$. At some point $\mathcal{A}$ outputs an uncorrupted challenge tag $\mathcal{T}^\star$. Then, the environment runs the authentication protocol with $\mathcal{T}^\star$. This yields an output $\perp$ when the tag was not recognized as legitimate or an identifier *id* when a legitimate tag was found. Finally, the adversary wins if the reader outputs $\perp$ and cannot identify $\mathcal{T}^\star$.

Definition 9 (Strong Correctness Game).

$\boldsymbol{Strong\text{-}Corr\text{-}Game}_{\Pi,\mathcal{A}}(\eta)$:
$(sk, pk) \leftarrow \text{SetupSystem}(1^\eta)$
$\mathcal{T}^\star \leftarrow \mathcal{A}^{\mathcal{O}}(pk)$
$\text{Execute}(\mathcal{R}^\star, \mathcal{T}^\star)$
$b \leftarrow \text{Result}(\mathcal{R}^\star)$
winif $b = 0$.

where $\text{Execute}(\mathcal{R}, \mathcal{T})$ *runs the authentication protocol between the reader $\mathcal{R}$ and the tag $\mathcal{T}$. The challenge tag $\mathcal{T}^\star$ must be uncorrupted, which means that no* $\text{CorruptTag}(\mathcal{T}^\star)$ *query has been made.*

Definition 10 (Strong Correctness). *Let C be a class of adversaries in $\{forward, weak, narrow, thin\}$. An RFID system is said to be C-*strong correct *if for all probabilistic polynomial-time adversaries $\mathcal{A} \in C$*

$$\mathbb{P}[\boldsymbol{Strong\text{-}Corr\text{-}Game}_{\Pi,\mathcal{A}}(\eta)] - \frac{1}{2}$$

is a negligible function of η.

Definition 11 (Key shifts). *A* key shift *in an RFID scheme is the increment of $|i-j|$ by 1 for an arbitrary tag $\mathcal{T}$ with tk_{id}^i and reader $\mathcal{R}$ with rk_{id}^j. The value $|i-j| \in \mathbb{N}$ is called* number of key shifts.

Remark 2. Note that our definition of *key shift* corresponds with the definition of *desynchronization* in [CC08]. We prefer to define *desynchronization* as the case where synchronization between a tag and reader is no longer possible.

The desynchronization value is a pair $(D_\mathcal{R}, D_\mathcal{T})$ where $D_\mathcal{R}$ is the maximum number of key shifts $j-i$ with $rk_{id}^i \neq tk_{id}^j$ and $i < j$, while $D_\mathcal{T}$ is the maximum

number of key shifts $i - j$ with $rk_{id}^{i} \neq tk_{id}^{j}$ and $i > j$. Correspondingly, the resynchronization value is a pair $(R_{\mathcal{R}}, R_{\mathcal{T}})$ where $R_{\mathcal{R}}$ and $R_{\mathcal{T}}$ are the maximum number of possible key shifts after which the RFID system still is *strong correct*. An RFID scheme is said to be synchronizable when both $D_{\mathcal{R}} \leq R_{\mathcal{R}}$ and $D_{\mathcal{T}} \leq R_{\mathcal{T}}$.

Definition 12 (Desynchronization). *An RFID scheme is subject to* desynchronization *when* $D_{\mathcal{R}} > R_{\mathcal{R}}$ *or* $D_{\mathcal{T}} > R_{\mathcal{T}}$.

5 Protocol Description

This section introduces a protocol that exploits the use of a *second channel* to achieve *thin-forward* privacy. The protocol should not be subject to desynchronization. Even when a tag is queried an unbounded number of times, this should not result in a denial-of-service (DoS) or in identification failure. First, we briefly elaborate on the notion of *second channel* that we use, then we define the tag and reader state in this protocol, and finally we discuss the protocol itself.

Second Channel. The protocol uses a *second channel* which is a channel between the tag and reader that allows a tag to send its tag identity to the reader. This channel uses other physical means than the wireless link and is therefore out of the scope of a *narrow* adversary. Like *narrow* adversaries cannot perform the Result query [Vau07], i.e. cannot learn outgoing messages on channels other than the wireless link, they cannot learn incoming messages that are sent on channels other than the wireless link. An example of an outgoing message on a *second channel* is for instance a door that opens when a tag is successfully authenticated. An example of an incoming message is for instance a barcode scanner or keypad connected to an RFID reader that communicates the tag identity to the reader. Of course, this identity still needs to be verified by the reader using the wireless link. The *second channel* speeds up the search process at the reader side when the tag and reader keys are relatively shifted. It does not replace the wireless link.

Tag and Reader State. In order to keep track of all the state changes and achieve an RFID system that cannot be desynchronized, the state is managed as follows. First we introduce some notation.

Notation	Meaning
id	The tag identifier
k	The session key; this key is updated in every protocol run
$\tilde{k}$	The synchronization key; for tag-reader synchronization
$h^i(x)$	i times successively hashing of x

Every tag is identified by an identifier id, but this identifier is not part of the tag state. However, a reader needs to relate this tag state somehow to the identifier of the tag. The tag state consists of a session key k and a synchronization key $\tilde{k}$.

This pair of keys $(k, \tilde{k})$ uniquely identifies a tag and thus can be related to id. The session key is updated in every protocol run, while the synchronization key is only updated after an authenticated message from the reader. A tag always starts to execute an internal key update before it sends any message to the outer world. The purpose of $\tilde{k}$ is to allow synchronization between the tag and reader. Finally, it should be possible to extract the identity id from the tag using a *second channel*. For example, the identity id can be printed on the tag as a barcode, which allows a barcode scanner to send id over the *second channel*. The reader state contains, apart from k and $\tilde{k}$, also the tag identifier id. To distinguish the keys in the reader state from the keys in the tag state we write $rk_{id}, r\tilde{k}_{id}$ and $tk_{id}, t\tilde{k}_{id}$, respectively. There are two ways in which the reader identifies a tag.

- The reader pre-computes $h(h^i(rk_{id}), n_r)$ for all $i < N$, all tag ids, and some nonce n_r. Now, identification is a look-up in its pre-computed table (See Tables 1 and 2).
- The reader obtains the identity id by use of a *second channel*. Now, id allows the reader to look up the synchronization key $r\tilde{k}_{id}$, which in its turn is used to induce synchronization of the tag and reader state.

The first way solely uses the wireless link whereas the latter way also uses the *second channel*. The synchronization is needed when the tag's session key is beyond the scope N of the reader. It allows a reader to quickly frame which tag it is targeting.

The protocol design is such that after a synchronization attempt of the reader a tag could either update its synchronization key or not. Depending on the situation there are two tag states possible. Therefore, the reader keeps track of two states for each tag simultaneously. The next protocol run in which this particular tag participates resolves then which of the two states is valid. In Table 1 and 2 the two states are captured by the record status st, which can either be 'old' (**O**) or 'new' (**N**). This makes the reader state consist of at most two tuples $(id, st, k, \tilde{k})$ per tag.

Table 1. Reader Database

id	Status st	Key k	Sync Key $\tilde{k}$	Identifier 1	...	Identifier i
id_1	**O**	k_1	$\tilde{k}_1$	$h(h^1(k_1), n_r)$	...	$h(h^i(k_1), n_r)$
id_1	**N**	k'_1	$\tilde{k}'_1$	$h(h^1(k'_1), n_r)$	...	$h(h^i(k'_1), n_r)$
id_2	**O**	k_2	$\tilde{k}_2$	$h(h^1(k_2), n_r)$	...	$h(h^i(k_2), n_r)$
⋮	⋮	⋮	⋮	⋮	⋱	⋮
id_n	**N**	k_n	$\tilde{k}_n$	$h(h^1(k_n), n_r)$	...	$h(h^i(k_n), n_r)$

Table 2. Look-up

Identifier	id	st
$h(h^1(k_1), n_r)$	id_1	**O**
$h(h^1(k'_1), n_r)$	id_1	**N**
$h(h^2(k_1), n_r)$	id_1	**O**
$h(h^2(k'_1), n_r)$	id_1	**N**
⋮	⋮	⋮
$h(h^i(k_1), n_r)$	id_1	**N**
$h(h^1(k_2), n_r)$	id_2	**O**
$h(h^2(k_2), n_r)$	id_2	**O**
⋮	⋮	⋮
$h(h^i(k_2), n_r)$	id_2	**O**

Precomputation and State Resolution. Two important questions need to be answered. First, how can the reader construct a precomputed table for look-up while a random nonce n_r is used in the protocol of Figure 2. Second, how can the number of possible tag states be limited in such a way that state resolution is always possible.

The reader state is stored as shown in Table 1. For every tag the reader precomputes the identifiers $h(h^i(rk_{id}), n_r)$ for all $i < N$. In practice, $N = 3$ might already be a good choice to withstand desynchronizations that occur due to bad physical circumstances. Since the reader cannot know id in advance, all nonces n_r in the precomputed table need to be the same. During idle time the reader can precalculate several tables as shown in Table 1 for different values n_r. A different representation of Table 1 is given by Table 2.

When a *synchronization run* is needed, first the identifier id is obtained by using the *second channel*. Then, the reader executes a *synchronization run*, immediately followed by a normal run. This second run makes clear whether the key update on the tag side was successful or not. If it was successful the reader is able to lookup the tag identifier in the database. However, in case of a *failure run* it is unclear whether the update was successful but the second run failed, or if the update already failed in the first place. For both scenarios the reader keeps a record corresponding to id, namely **O** and **N**. In order to prevent desynchronization on this level, this specific tag can be labeled as 'suspicious' to indicate that something went wrong in the synchronization run. The tag needs then to be synchronized in a safe environment. Every other attempt of a reader to synchronize would potentially leak location information to an adversary and should therefore not be executed.

Success, Failure and Synchronization Run. This section discusses the *success*, *failure* and *synchronization run*. The authentication protocol is depicted in Figure 2. The *success run* is a protocol run in which a reader is able to successfully identify a tag and updates the identifiers in the database accordingly. This update might just concern the next identifier k, or update the synchronization key $\tilde{k}$ as well. Whenever a reader fails to identify a tag, the corresponding protocol run is called a *failure run*. After a failure run, the reader needs to be provided with the tag identifier id using a *second channel*. Now, id can be used to select the tag in the database and find the corresponding synchronization key $\tilde{k}$ which can be used to execute a *synchronization run* and update both k and $\tilde{k}$. The idea behind the different run types is that they look the same to an adversary.

A *success run* starts with a challenge nonce n_r. Under all circumstances, the tag computes the successive tag key $k \leftarrow h(k)$ before it sends any message. This key updating is done regardless of the number of requests that are made. After this phase, the identifier m_1 is sent, which directly depends on k as $m_1 \leftarrow h(k, n_r)$. Due to this dependence on k, the successive tag identifiers might run beyond the identifiable scope N of the reader. Since a reader cannot continue to search for an identifier forever, N determines the maximum number of key updates considered in a look-up attempt. In the success run we consider a lookup

successful when it is of the form $\exists(id, k) \in T, i \leq N : h(h^i(k), n_r) = m_1$. The corresponding identity id, key k and resynchronization key $\tilde{k}$ of the tag are resolved, which completes the identification of the tag. For similarity reasons, the reader finishes by sending a random m_2 message.

The *failure run* starts like every run with a challenge n_r. In its turn, the tag first computes the next tag key $k \leftarrow h(k)$ before any message is sent. In contrast to a *success run*, the reader is unable to resolve the tag's identity from message m_1. Since m_2 can be a random message, the reader is still able to finish the protocol, as it is designed to show equal behavior in every run. However, identification was unsuccessful and thus the reader has to obtain the tag identifier id by using a second channel, e.g. the id could also be available as a barcode on the RFID label. Of course, the adversary can obtain id as well, but the tracking effort per tag is relatively large compared to the tracking of RFID labels with fixed identifier. Hence, this protocol reduces the problem of tracking RFID labels to the problem of tracking barcodes.

Finally, the *synchronization run* is used once the identifier id is obtained by the reader. The identifier id can be provided over the second channel and allows the lookup of k and $\tilde{k}$, which are used later on in this run. Again, the reader starts the protocol by sending a nonce n_r. The tag computes $m_1 \leftarrow h(k, n_r)$ and updates the tag key $k \leftarrow h(k)$, then it sends m_1 which is used as unpredictable input for the last message m_2. By m_2 the reader proves knowledge of the synchronization key $\tilde{k}$ to the tag. This time, m_2 is constructed from m_1 and $\tilde{k}$ as $m_2 \leftarrow h(m_1, \tilde{k})$. The tag knows $\tilde{k}$ and can therefore check the validity of m_2. If it is indeed a valid message, the tag updates the tag key $k \leftarrow h(\tilde{k}, m_1)$ and the synchronization key $\tilde{k} \leftarrow h(\tilde{k})$.

Reader		**Tag**
state: $T = [id, k, \tilde{k}]$		state: $k, \tilde{k}$
$n_r \leftarrow \{0,1\}^l$	$\xrightarrow{n_r}$	
		$m_1 \leftarrow h(k, n_r)$ $k \leftarrow h(k)$
	$\xleftarrow{m_1}$	
$\exists(id, k) \in T, i \leq N :$ $h(h^i(k), n_r) = m_1$ **then** $m_2 \leftarrow \{0,1\}^l$ $k \leftarrow h^i(k)$ **else** $m_2 \leftarrow h(m_1, \tilde{k})$	$\xrightarrow{m_2}$	
		if $h(m_1, \tilde{k}) = m_2$ **then** $k \leftarrow h(\tilde{k}, m_1)$ $\tilde{k} \leftarrow h(\tilde{k})$

Fig. 2. The Protocol

6 Security Analysis

This section analyzes the security of the proposed protocol in the random oracle model. In the *resynchronization run* the last message m_2 of the protocol leaks location information. For this reason, and in general because forward privacy cannot be achieved for any type of synchronized symmetric protocol construction [NSMSN09], we use the slightly more restricted *thin* adversary. First, we show that our protocol is *thin-forward* private. Then we show that the protocol is not subject to desynchronization.

Theorem 1. *The protocol depicted in Figure 2 is thin-forward private in the random oracle model.*

The proof closely follows the *narrow-forward privacy* proof of modified OSK in [GvR10]. In short, it introduces a simulator $\mathcal{S}$ which keeps track of all oracle calls $\mathcal{H}$ and stores them as an entry of the form $\langle \text{IN}, \text{OUT} \rangle$ in a table $T_{\mathcal{H}}$. Then, $T_{\mathcal{H}}$ is adapted such that the protocol messages and thus the resulting view of a particular adversary $\mathcal{A}_1$ remain the same while the keys, and thus the tag identities, are swapped. This leads to a contradiction.

Proof (Sketch). Suppose there exists an adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ that wins the **Priv-Game**$_{\Pi}$ given in Definition 5 with non-negligible probability. Then, imagine a simulator $\mathcal{S}$ that first initializes the system and then runs the adversary $\mathcal{A}_0$. Every oracle call of $\mathcal{A}_0$ to the oracle $\mathcal{H}$ is simulated as usual by a table $T_{\mathcal{H}}$ which contains all previous queries with their corresponding answers. At some point $\mathcal{A}_0$ finishes and chooses two tags $\mathcal{T}_0^*$ and $\mathcal{T}_1^*$. Let $(k_0, \tilde{k}_0)$ be the key pair of $\mathcal{T}_0^*$ and $(k_1, \tilde{k}_1)$ be the key pair of $\mathcal{T}_1^*$ after they are returned by $\mathcal{A}_0$. As in the game, $\mathcal{S}$ will draw a random bit b. Next, $\mathcal{S}$ runs $\mathcal{A}_1^{\mathcal{O}}(T_b^*)$ which at some point outputs a guess bit b'. By hypothesis we get that $b' = b$ with probability significantly higher than $\frac{1}{2}$. By $\dagger$ we identify the predecessor value of a key, so the predecessor of k_0 is $k_0^{\dagger}$. Now $\mathcal{S}$ swaps all occurrences of k_0 with k_1 in all entries of $T_{\mathcal{H}}$. Note that either the entry $\langle h(\tilde{k}_0^{\dagger}, \_), k_0 \rangle$ or the entry $\langle h(k_0^{\dagger}), k_0 \rangle$ is present in $T_{\mathcal{H}}$. The first one occurs when the last update of k was in a *synchronization run*. The latter one occurs when the last update of k was in a non-synchronization protocol run. The replacement of k_0 by k_1 and vice versa does not affect the protocol messages since k_0 and k_1 are not involved in any protocol messages after the oracle call entries defined above. Furthermore, $m_2 \leftarrow h(m_1, \tilde{k})$ is the only message that involves $\tilde{k}$ and only occurs in a *synchronization run*. Since $\mathcal{A}_1$ is thin, it is clear that $\tilde{k}$ does not have any influence on the view of the adversary.

Now, $\mathcal{S}$ runs adversary $\mathcal{A}_1^{\mathcal{O}}(T_{1-b}^*)$ with the adjusted $T_{\mathcal{H}}$. Again by hypothesis, we get that $\mathcal{A}_1$ outputs $b' = 1 - b$ with probability significantly higher than $\frac{1}{2}$. Since $\mathcal{A}_1$ is thin, its view is exactly the same as in the previous run, which leads to a contradiction.

Theorem 2. *The protocol depicted in Figure 2 is not subject to desynchronization in the random oracle model.*

Proof (Sketch). In order to show that desynchronization is impossible we have to show that both $D_{\mathcal{R}} \leq R_{\mathcal{R}}$ and $D_{\mathcal{T}} \leq R_{\mathcal{T}}$ hold. The tag state is the tuple $(k, \tilde{k})$. First, k is always updated, $\tilde{k}$ is only updated after a *synchronization run*. Therefore, we focus $\tilde{k}$ to induce key shifts since only then a desynchronization is possible. From the protocol definition we deduce that $D_{\mathcal{R}} = R_{\mathcal{R}} = 1$ since a reader only starts a synchronization run when it was able to look up $\tilde{k}$ in one of the two possible tag states. Furthermore, we know that $D_{\mathcal{T}} = 0$ from which follows that $D_{\mathcal{T}} \leq R_{\mathcal{T}}$ since $R_{\mathcal{T}}$ has to be positive. Suppose that either $D_{\mathcal{R}} > R_{\mathcal{R}}$ or $D_{\mathcal{T}} \leq R_{\mathcal{T}}$ is true, then there exists an adversary $\mathcal{A}$ that wins the **Strong-Corr-Game**$_{\Pi}$ given in Definition 9 with non-negligible probability. This means that $\mathcal{A}$ outputs a tag $\mathcal{T}^{\star}$ with key $t\tilde{k}^i$ while the reader has no matching key $r\tilde{k}^{j-1}$ or $r\tilde{k}^j$, since $i \neq j-1$ and $i \neq j$ has to be true. There are two ways for the adversary to achieve this:

$i > j$: The tag key is updated $(i-j)$-times more than the reader key. The only way to induce a key update on the tag side is to construct the message $m_2 = h(m_1, \tilde{k})$. Because of the one-wayness of h and since the adversary cannot call CorruptTag, the key $\tilde{k}$ is not known and it is impossible to construct m_2 for the adversary. Only the reader $\mathcal{R}$ is able to construct $m_2 = h(m_1, \tilde{k})$, but inherent to this generation of m_2 is the storage of the new reader keys $(rk^{j+1}, r\tilde{k}^{j+1})$ while at the same time the old keys $(rk^j, r\tilde{k}^j)$ are maintained. The last option would be a replay of m_2, but this is rendered impossible by the use of n_r in m_1, and thus in m_2, which introduces freshness in every protocol run. To conclude, it is not possible to obtain $i > j$.
$i < j-1$: The reader key is updated $(j-i)$-times more than the tag key. By hypothesis we know that $i < j-1$ since $i \neq j$, $i \neq j-1$ and $i \not> j$ as concluded in the previous case. Let $i = j$, the only way to update $r\tilde{k}^j$ to $r\tilde{k}^{j+1}$ comes with the generation of $m_2 = h(m_1, \tilde{k})$. If m_2 is received by the tag it will update its key from tk^i to tk^{i+1} and consequently $i = j$ again. Obviously, to prevent incrementation of i is to block or replace m_2 since then the tag does not update its key and as a result $i = j-1$. Next, the adversary needs to go one step further since the reader is still able to identify the tag $(t\tilde{k}^i = r\tilde{k}^{j-1})$. To induce another reader key update, the reader has to be provided with the tag identifier id by using the *second channel*. When the last synchronization attempt turned out to be unsuccessful, which is stored in the reader state belonging to id, the reader just sends random data for m_2. In this situation resynchronization has to be done in a safe environment. The tag state either contains $t\tilde{k}^i$ when in the last synchronization attempt m_2 was blocked or the tag state contains $t\tilde{k}^{i+1}$ when the last synchronization run was successful. In the latter case the reader is able to identify the tag since it knows $r\tilde{k}^j$ which equals $t\tilde{k}^{i+1}$, respectively. To conclude, the adversary needs to induce a synchronization run, which can be done by first querying the tag more than N times. Then, before the reader starts a *synchronization run* it retrieves id. By looking up the correct entry using id the reader has enough information to decide on the execution of another *synchronization run*. If the last attempt was unsuccessful this indicates that something suspicious is

going on and resynchronization should be done in a safe environment. If the last attempt was successful the reader is sure that $i = j$. So, $\max(|i-j|) = 1$ where $i < j$, which is not enough to satisfy $i < j - 1$.

Finally, from the two possible strategies to win the **Strong-Corr-Game**$_{\Pi}$ we conclude that both $i > j$ and $i < j-1$ cannot be satisfied, therefore contradicting the assumption that such an adversary $\mathcal{A}$ exists.

7 Conclusion

This paper presents a new approach to tackle the desynchronization problem. This desynchronization problem is actually an unwanted side effect of a solution to another problem: location privacy for RFID tags. Many solutions tend to solve this problem by introducing a stateful protocol. A main challenge of these protocols is to keep the tag and reader state synchronized while at the same time no information can be leaked that enables an adversary to track a specific tag. To the best of our knowledge there have been no attempts to seek the solution beyond the bounds of the wireless link. In line with the abilities of a *narrow* adversary, introduced by Vaudenay in [Vau07], in which an adversary is unable to see the result of a protocol run like a gate that opens, this paper proposes to use this information flow also in the opposite direction. This means that additional information is made available to the reader which it can use to identify and resynchronize with the tag. A *narrow* adversary does not have access to this information since it is not send on the wireless link but some other communication channel which is introduced in this paper as the *second channel*. This paper adds some mild restrictions to the *narrow* adversary and introduces this as the *thin* adversary which is needed to prove forward-privacy under mild additional assumptions. Suppose that barcode scanners are used as *second channel* and RFID tags are additionally equipped with barcodes. Additionally, assume a protocol P that uses the *second channel* such that it provides *thin-forward* privacy and is not subject to desynchronization. Then, tracking tags in this system has become as hard as tracking barcodes.

The *second channel* can be used in new protocol designs and relaxes the workload of the reader and/or database. It allows to solve the desynchronization problem in an elegant way and eliminates the need for restrictions on the number of key updates that can be induced by an adversary between two synchronizations. In order to show that such a protocol can be constructed we proposed a protocol that only uses hash functions. We have shown that it it provides *thin-forward* privacy in the random oracle model. Furthermore, we followed the desynchronization definition of [CC08] to show that the protocol is not subject to desynchronization.

We are currently working on the formalization of the security claims in Proverif, following the direction of [BCdH10]. This task turned out to be non-trivial due to the fact that our protocol is state-full.

Acknowledgments

We like to thank the anonymous reviewers of this paper for their valuable comments.

References

[AO05] Avoine, G., Oechslin, P.: A scalable and provably secure hash based RFID protocol. In: International Workshop on Pervasive Computing and Communication Security, PerSec 2005, pp. 110–114 (2005)

[Att07] Attaran, M.: RFID: an enabler of supply chain operations. Supply Chain Management: An International Journal 12(4), 249–257 (2007)

[Avo05] Avoine, G.: Adversary Model for Radio Frequency Identification. Technical Report LASEC-REPORT-2005-001, Swiss Federal Institute of Technology (EPFL), Security and Cryptography Laboratory (LASEC), Lausanne, Switzerland (September 2005)

[BBEG09] Berbain, C., Billet, O., Etrog, J., Gilbert, H.: An efficient forward private RFID protocol. In: Proceedings of the 16th ACM conference on Computer and communications security,CCS 2009, pp. 43–53. ACM Press, New York (2009)

[BCdH10] Brusó, M., Chatzikokolakis, K., den Hartog, J.: Formal verification of privacy for RFID systems. In: Proceedings of the 23nd IEEE Computer Security Foundations Symposium (2010)

[BdMM08] Burmester, M., de Medeiros, B., Motta, R.: Anonymous RFID authentication supporting constant-cost key-lookup against active adversaries. Journal of Applied Cryptography 1(2), 79–90 (2008)

[CC08] Canard, S., Coisel, I.: Data synchronization in privacy-preserving RFID authentication schemes. In: Conference on RFID Security (2008)

[Dim05] Dimitriou, T.: A lightweight RFID protocol to protect against traceability and cloning attacks. In: Security and Privacy for Emerging Areas in Communications Networks, SecureComm 2005, pp. 59–66 (2005)

[GvR10] Garcia, F., van Rossum, P.: Modeling privacy for off-line RFID systems. In: Gollmann, D., Lanet, J.-L., Iguchi-Cartigny, J. (eds.) CARDIS 2010. LNCS, vol. 6035, pp. 194–208. Springer, Heidelberg (2010)

[Jue06] Juels, A.: RFID security and privacy: A research survey. IEEE Journal on Selected Areas in Communications 24(2), 381–394 (2006)

[JW05] Juels, A., Weis, S.: Authenticating Pervasive Devices with Human Protocols. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 293–308. Springer, Heidelberg (2005)

[JW09] Juels, A., Weis, S.A.: Defining strong privacy for RFID. ACM Transactions on Information and System Security (TISSEC) 13(1), 1–23 (2009)

[MW04] Molnar, D., Wagner, D.: Privacy and security in library RFID: Issues, practices, and architectures. In: Proceedings of the 11th ACM conference on Computer and Communications Security, pp. 210–219. ACM, New York (2004)

[NSMSN09] Ng, C.Y., Susilo, W., Mu, Y., Safavi-Naini, R.: New Privacy Results on Synchronized RFID Authentication Protocols against Tag Tracing. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, p. 321. Springer, Heidelberg (2009)

[OSK +03] Ohkubo, M., Suzuki, K., Kinoshita, S., et al.: Cryptographic approach to privacy-friendly tags. In: RFID Privacy Workshop, Citeseer, vol. 82 (2003)

[Tsu06] Tsudik, G.: YA-TRAP: Yet Another Trivial RFID Authentication Protocol. In: International Conference on Pervasive Computing and Communications, PerCom 2006, Pisa, Italy. IEEE Computer Society Press, Los Alamitos (March 2006)

[Vau07] Vaudenay, S.: On privacy models for RFID. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 68–87. Springer, Heidelberg (2007)

[WCO+07] Wanga, S.W., Chenb, W.H., Onga, C.S., Liuc, L., Chuangb, Y.W.: RFID applications in hospitals: a case study on a demonstration RFID project in a Taiwan hospital. Hospitals 8, 33 (2007)

[WNLY06] Wu, N.C., Nystrom, M.A., Lin, T.R., Yu, H.C.: Challenges to global RFID adoption. Technovation 26(12), 1317–1323 (2006)

[WSR +04] Weis, S.A., Sarma, S.E., Rivest, R.L., Engels, D.W., et al.: Security and privacy aspects of low-cost radio frequency identification systems. LNCS, pp. 201–212 (2004)