Practical Schemes for Privacy and Security Enhanced RFID (Extended Abstract)

Jaap-Henk Hoepman[1,2] and Rieks Joosten[1]

[1] TNO Information and Communication Technology
jaap-henk.hoepman@tno.nl, rieks.joosten@tno.nl
[2] Radboud University Nijmegen
jhh@cs.ru.nl

Abstract. Proper privacy protection in RFID systems is important. However, many of the schemes known are impractical, either because they use hash functions instead of the more hardware efficient symmetric encryption schemes as a efficient cryptographic primitive, or because they incur a rather costly key search time penalty at the reader. Moreover, they do not allow for dynamic, fine-grained access control to the tag that cater for more complex usage scenarios.

In this paper we propose a model and corresponding privacy friendly protocols for efficient and fine-grained management of access permissions to tags. In particular we propose an efficient mutual authentication protocol between a tag and a reader that achieves a reasonable level of privacy, using only symmetric key cryptography on the tag, while not requiring a costly key-search algorithm at the reader side. Moreover, our protocol is able to recover from stolen readers.

1 Introduction

Radio Frequency Identification (RFID) is a technology that allows to wirelessly identify and collect data about a particular physical object from a relatively short distance. The data is stored on so-called tags attached to the object, and is collected using so-called readers. RFID tags can be very small, can be attached invisibly to almost anything, and can transmit potentially unique identifying information. Therefore, proper privacy protection within RFID based systems is of paramount importance [11, 17].

Yet RFID is also an enabler for the vision of an Internet-of-Things where the physical and the virtual become interconnected in one single network. This will spark all kinds of applications beyond our current imagination. Unfortunately, the current trend in RFID related policy aims to mandate a kill-switch on all RFID tagsthat will silence such a tag forever once it leaves the shop. Such a kill-switch is a very coarse, all-or-nothing approach to protecting privacy. It would be far better to develop an approach that allows the user to have fine grained and dynamic control over who can access his tags, and when. The research reported on in this paper takes a step into that direction.

P. Samarati et al. (Eds.): WISTP 2010, LNCS 6033, pp. 138–153, 2010.

1.1 State of the Art

Because of the privacy risk associated with the large scale use of RFID tags, many proposals exist to provide a certain level of privacy protection for a particular application of RFID. We give a brief overview of the state of the art, focusing on authentication and access control. For details we refer to the full paper [14], to Juels [17] (and the excellent bibliography[1] maintained by Gildas Avoine) for a much more extensive survey of proposed solutions, and [19] for a more formal analysis of the privacy properties actually achieved by some of the proposed authentication protocols.

Early proposals use relabelling of tag identifiers [24], or re-encryption techniques [18, 2, 12] that randomly encrypt the identifier from time to time, so that it can only be recovered by authorised readers, while being untraceable for others.

Another approach is to implement some form of authentication between tag and reader, and to allow only authorised tags to retrieve the tag identifier. In a public key setting this would be easy, but RFID tags are generally considered to be too resource poor to accommodate for that. Therefore, several identification and authentication protocols using hash functions or symmetric key cryptography have been proposed [29, 9]. In particular, Dimitriou [8] presents a technique for achieving forward privacy in tags. All readers should be *on line*, connected with one central database, so they can be synchronised and the response of a tag can be looked up immediately in the database.

In a symmetric key setting the reader cannot know the identifier of the tag a priori, or obtain the identifier of the tag at the start of the protocol because of privacy concerns. One can give all readers and tags the same symmetric key, but this has the obvious drawback that once the key of one tag is stolen, the whole system is corrupted. To increase security, tags can be given separate keys, but then the reader must search the right key to use for a particular tag. The core challenge is therefore to provide, possibly efficient, trade offs and solutions for key search and key management. Molnar and Wagner [20] propose a tree-based key hierarchy to achieve such a trade-off. They also introduce the concept of delegation that allows a tag owner to enable another party to access a tag over some period of time. In another approach, Avoine, Dysli, and Oechslin [3] show how a time-memory trade off can be exploited to make the search for the key to use more efficient. We note that none of these systems are practical for RFID systems where millions of tags possess unique secret keys.

Spiekermann *et al.* [25] observe that although there are many protocols and proposals for limiting access to RFID tags (either by killing them completely or by requiring the reader to authenticate), few systems have been proposed that allow effective and fine grained control over access permissions. The RFID Guardian [23] is a notable exception. The main idea is to jam all reader to tag communication, except for requests that satisfy a pre-defined privacy rule.

We base our work on (relatively) new insights regarding the amount of hardware required to implement symmetric key cryptosystems as compared to hash

[1] `www.avoine.net/rfid/`

functions [22], which shows that symmetric cryptography is the preferred choice for lower cost tags.

1.2 Our Contribution

Our contribution is to propose a model and corresponding protocols that allow effective, efficient and fine grained control over access permissions for RFID tags, that respect the privacy of the users. The model is enforced by the tags themselves. The protocols use authentication as a basic component, and we propose a novel combination of (universal) re-encryption [18, 12] with symmetric cryptography based authentication [16] to obtain a reasonable level of privacy protection without using public-key cryptography on the tag, and without the need for a time consuming key-search algorithm. Although such key-search algorithms are highly popular in the research community because of their superior privacy properties, we believe they are unreasonable for large scale applications that may involve millions of tags (and hence keys). Finally, our protocols are resistant to stolen reader attacks, using techniques from [4]. A detailed description of the properties of our authentication protocol is presented in Sect. 4.

The model is loosely based on the "Resurrecting Duckling" paradigm of Anderson and Stajano [27, 26]. Our model is general enough to capture several RFID use case scenarios, like supply chain management, ticketing and ambient home intelligence. See the full paper [14] for details. The essence of the model is that a potentially dynamic system of access permissions is defined. We generalise the concept of an RFID tag, and view such a tag as a container of several data objects on which a reader wishes to execute certain functions. This extends the notion of an RFID tag containing just a unique identifier to slightly smarter data container. We believe that labelling a physical object with a unique identifier on an RFID tag, and storing all relevant data on the object in a central database is going to prove too limitative in the future. For privacy reasons it is better to require physical proximity to read the data on the tag instead of having that data available in a database all the time. This research is related to the PEARL[2] project.

The paper is structured as follows. In Sect. 2 we present our system model. We then continue to implement this model using data structures (Sect. 3), an authentication and session key establishment protocol (Sect. 4) and subsequent protocols (Sect. 5). The security proofs appear in Sect. 6 and we present some conclusions and further research in Sect. 7. The scenarios on which the model is based, and an analysis of the mapping of the system model on the use cases are omitted due to space constraints but can be found in the full paper [14].

2 System Model

The system model describes the different entities in the system, their mutual relationships, and the operations that they can perform on each other.

[2] `www.pearl-project.org`

2.1 Notation

We use k to denote a symmetric key, possibly sub-scripted to denote its owner, and use s to denote a symmetric session key. We use PK for a public key and sk for the corresponding private key. Hash functions are denoted by $h(\cdot)$. We write $\oplus$ for the exclusive-or operation, and ; for concatenation of bit strings. $\{m\}_k$ denotes the encryption of message m with symmetric key k using some symmetric cipher, typically AES. $[m]_k$ denotes a message authentication code (MAC) for message m derived from a symmetric cipher (for instance CMAC [21, 6]) using key k. Finally, $[\{m\}]_k$ denotes the authenticated encryption of m with key k, for instance by appending the MAC of the ciphertext [5].

2.2 Tags and Readers

A *tag* t is a piece of hardware that contains data. At the very minimum, tags store a bit string that can be read and sometimes written. Usually, tags store several values that can be grouped together as tuples because of their logical use. More complex, smart card like tags, contain ISO 7816 [15] like file structures. We assume that for the anti-collision protocol random identifiers are used (or else all bets to achieve some level of privacy are off).

The system model follows the object oriented (OO) metaphor, so that tags are said to contain *objects*, each of which is a group of bit strings whose structure is defined by the *class* that it instantiates. For every class, each tag contains at most one instantiating object. Every class defines a set of *methods*, each of which specifies a kind of operation that may take place on objects that instantiate that class. Simple methods specify how to read or perhaps write values in a tuple of a certain type stored on a particular tag. More complex cases methods might invalidate a ticket on a tag, or increase an electronic purse balance. Every method is defined in precisely one class.

Every tag always contains one instance Ω of the *tag management class*, initially with default settings. The tag management class implements functions to manage tag access and ownership. This allows us to implement tag and class management operations in a similar way as methods on ordinary objects, thus simplifying the implementation. Details are provided in Sect. 5.

We assume readers are at least on-line some of the time to obtain fresh data and keys from the central back office.

2.3 Domains and Principals

We use the term *domain* to refer to a (legal) entity that is capable of bearing responsibilities. Thus, companies, organisations and governments are considered to be domains, as well as individual (adult) persons. We use the term *principal*, or *actor*, to refer to a resource (e.g. a person, or a running application within a computer) that is capable of acting under the responsibility of a domain. We assume that at any particular point in time d acts on behalf of precisely one domain D. Thus, if a principal d acts on behalf of a domain D at a given point

in time, then D is responsible for everything that d does at that time. We use $\mathcal{D}$ to denote the set of all domains.

2.4 Ownership

We use the term *owner(ship)* to refer to the responsibilities associated with controlling tags, objects, etc. Since responsibilities are born by domains, ownership can only be assigned to domains. Ownership can be transferred by the owning domain to another (accepting) domain.

Thus, the *tag owner* T for a tag t is a domain that bears the responsibility for controlling access to t, i.e. for issuing and revoking the associated permissions. Also, it controls the permissions associated with other tag related functionality, such as the creation of objects or the transferal of tag ownership. We use $\mathcal{T} \subseteq \mathcal{D}$ to denote the set of tag owners. We write $t \in T$ to indicate that tag t is owned by T.

The *class owner* is responsible for controlling access to objects that instantiate this class, i.e. for issuing and revoking permissions for executing methods defined by that class. We write $c \in C$ to mean that class c is owned by domain C (i.e. its class owner).

Note that if a class owner C owns a class c, then (initially) it also owns every object $o \in c$. Thus, object ownership is (initially) implied by class ownership. However, ownership of individual objects may be transferred to other domains later on. If that happens, the class owner is not necessarily the owner of all objects of that class.

2.5 Permissions

Every *permission*, i.e. the right to access a tag or the right to execute a method on an object, is issued by the domain that owns the tag or the object, to any domain of its choosing. One of our main contributions is the distinction we make between accessing (i.e. communicating with) tags and accessing (i.e. executing methods on) objects on a tag. A consequence of this distinction is that it requires two rather than one permission to access an object on a tag: one permission is needed for accessing the tag on which the object is stored (which is granted by the tag owner), and the other permission is required to execute the appropriate method on that object (which is granted by the object owner).

2.6 Operations on a Tag

The most basic operation the model must support is calling a method on an object of a certain class stored on a particular tag. For this, two permissions are required: first, the domain must be allowed to access the tag, and secondly the domain must be allowed to execute the method on (the class of) the object. Note that access to a method is initially granted at the class level. So access rights for a particular method initially apply to all objects of that class.

The creation of permissions is done off-tag, as is the distribution thereof. Tag ownership is controlled through the functions **TakeTagOwnership**, **TransferTagOwnership** and **RelinquishTagOwnership**. Tag access is controlled through the following functions: **GrantTagAccess** and **RevokeTagAccess**. These functions are only executable by the tag owner.

Object management is controlled through the following functions: **InstallObject**, **UpdateObject**, **UpdateClassKey** and **DeleteObject**. For more information we refer to Sect. 5 and the full paper [14]

3 Data Structures

In this section we describe the data structures stored by the tags, and the keys and permissions used by the domains to access the data on a tag. In the next section we describe the implementations of the operations that can be performed on a tag.

3.1 Keys

To implement permissions, the system uses the the following types of keys. Some keys (the domain key pairs PK_D, sk_D) are asymmetric keys, the other keys are symmetric keys.

Tag access keys k_a. Access to tags is controlled using tag access keys k_a. These keys are unique to a tag, and derived from the tag identifier t using a master access key k_A through key diversification [1] by $k_a = \{t\}_{k_A}$.

Master access keys k_A. Each domain has a master access key k_A. Readers in a domain use this master access key k_A to derive tag access keys from tag identifiers. Each tag thus stores, for each domain that is allowed to access it, a different tag access key.

Domain key pairs PK_D, sk_D. Each domain keeps a unique ElGamal public/private domain key pair PK_D, sk_D. These keys are used in the authentication protocol to preserve privacy of the tag identifier t. To thwart stolen reader attacks, readers get a new pair of keys every once in a while. These keys are stored in the array $E[]$.

Class keys k_c. For each class there exists a unique class key k_c. The class key is used to encode access permissions to the class methods. A tag stores, for each object, the corresponding class key to verify such permissions. Class owners know all the class keys of the classes they own. Changing the class key of an individual object can be utilised to transfer ownership of that particular object. Conceptually, however, this makes the object member of another class (albeit with the same structure and methods as the class it originally was a member of).

3.2 Other Data Stored on the Tag

A tag t also performs a bit of bookkeeping. Firstly, it records a time stamp now_t that approximates the current date and time (see below), initially $-\infty$. Tags

also store several objects, each of a class c together with the key k_c. Also, a tag t keeps an access set A_t that stores, for each domain D that is granted access to the tag, the following three items.

- An encrypted tag identifier id, equal to the ElGamal encryption $(t \cdot PK_D^x, g^x)$ of the tag identifier t.
- The epoch e in which the encrypted tag identifier was created (for explanation see Sect. 4).
- The diversified tag access key k_a, which equals $\{t\}_{k_A}$ for the master key k_A used by domain D.
- A boolean flag indicating whether this domain is the owner of the tag.

We interpret the access set as a dictionary indexed by domains (as a domain can have at most one such tuple in the access set), and write $A_t[D] = (id, k_a, b)$. There is at most one domain that is the owner of the tag. We write $owner_t$ for that domain (which equals $\perp$ if the tag is not owned by a domain). Initially, $A_t = \emptyset$.

Finally, the tag stores the current session key s, which initially and in between sessions equals a default value (denoted $\perp$, but which actually is a valid key), and which is set to a certain value as the result of a successful mutual authentication (in which case the authenticated reader holds the same session key). It also stores the domain of the reader that was authenticated in Γ (which equals $\perp$ in between sessions). We usually omit the subscript from now, $owner$ and A.

3.3 Permissions

To grant a domain D access to a method f on an object of class c up to time Δ, the class owner C generates a *permission token* $k_{c,f,D,\Delta} = \{f, D, \Delta\}_{k_c}$ and sends this to the domain D. This permission token expires as soon as the current time exceeds Δ. Tags use now as their estimate of the current time to verify this. They do not need to have their own clock, and this estimate does not have to be synchronised with other tags. This is updated after each successful call of a method on the tag (which includes the current time as asserted by the caller). It is also set to the current time when the first domain takes ownership of the tag. A similar method is also used by the European RFID passports [7, 13].

4 Mutual Authentication and Establishing a Session Key

A basic step underlying the protocols that implement the operations that access a tag, is to mutually authenticate a tag and a reader, and to establish a session key among them.

Below we present a protocol that is efficient for both the reader and the tag. In principle it combines elements of three different known authentication protocols to strike a balance between tag and reader efficiency, achieve a robustness against a reasonably large class of adversaries, and achieve a certain level of privacy as well. In fact it combines a standard, ISO/IEC 9798-2 [16] based

symmetric key authentication protocol, with (universal) re-encryption [18, 12] to avoid the costly key search, and a counter based approach to invalidate keys from stolen readers [4]. To further enhance privacy, users may perform a separate re-encryption of all identifiers on a tag at any time.

To be precise, the protocol achieves the following properties

mutual authentication. The reader and the tag are mutually authenticated.

soft privacy. Tags can only be traced in between two successful re-encryptions (including the re-encryption performed during an authentication). Except for the reader performing the re-encryption, no other reader or eavesdropper can link the presence of the tag after the re-encryption with an observation of this tag before the re-encryption.

resilience to tag compromise. Tags do not contain global secrets. Hence a tag compromise does not affect any other tags in the system.

resilience to reader compromise. Stolen readers (or otherwise compromised readers) will not be able to recognise or access tags, once those tags have been in contact with another valid reader after the compromise [4]. A similar property is achieved by the European biometric passports [7, 13].

reader efficiency. The reader performs a constant number of operations.

tag efficiency. The tag performs only a constant number of symmetric key cryptography operations.

The protocol we present below explicitly checks the correctness of the responses, that may contain additional information for that purpose, to positively authenticate the other party. Another option is to rely on implicit authentication through the session key that is established as well: if the authentication fails, both parties will have different values for the session key, and therefore subsequent protocol steps will fail. Note that in the description of the protocols we do not explicitly describe the behaviour of a principal if it detects such an error. Instead we use the convention that if an internal check fails, the principal continues to send the expected messages at the appropriate times, with the appropriate message format, but with random message content. This is necessary to preserve privacy, as observed by Juels *et al.* [19].

Our protocol (see Fig. 1) is an extension of the the ISO/IEC 9798-2 [16] standard, using diversified keys [1] to avoid sharing keys over many tags. To compute the diversified tag access key k'_a from the master access key k_A it stores, the reader needs to learn the tag identifier t. This cannot be sent in the clear for privacy reasons. The solution is to encrypt the tag identifier t against the public key of the reader to obtain id, and let the reader re-encrypt [18] that value with every authentication run. This way the tag does not have to perform any public key operations. Note that the re-encrypted value is only used as the new tag identifier after a successful authentication of the reader. This avoids denial-of-service attacks. Finally, the re-encryption keys stored by the readers are updated every time a reader is stolen. Every time this happens, a new *epoch* is started. Stolen readers no longer receive keys for future epochs. Tags that authenticate successfully, receive a new encrypted identity, encrypted against

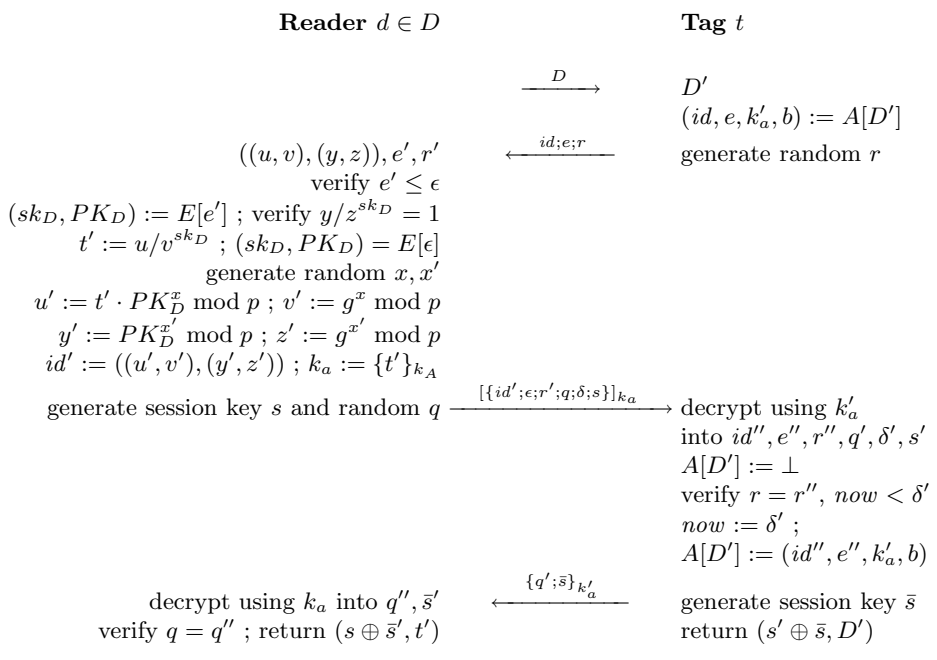

Fig. 1. Authentication and session key agreement

the most recent epoch key. This makes it impossible for compromised readers to track this tag.

At the reader side the protocol returns the tag identifier and the session key to be used. $AuthenticateR(sk_D, PK_D, k_A)$ denotes a call to such an authentication protocol run in the protocols below. At the tag side, the protocol returns the session key, as well as the authenticated domain. We write $AuthenticateT()$ for this call.

4.1 Re-encryption

The protocol uses re-encryption, or rather *universal* re-encryption [12], as follows. We use the ElGamal encryption scheme [10] over a cyclic group G of order q. To be concrete, and to achieve semantic security [28], we choose two primes p and q such that $q \| (p-1)$ (i.e., q is a divisor of $(p-1)$) and choose as G the cyclic subgroup of $\mathbb{Z}_p$ with order q, and pick a generator g for G. These are global, system wide, constants.

Each domain has, for each epoch, its own public/private key pair (PK_D, sk_D) where sk_D is a random integer between 1 and $q-1$, and $PK_D = g^{sk_D}$. The tag identifier t is encrypted, using ElGamal, as $(u, v) = (t \cdot PK_D^x, g^x)$, where x is a random value in $[0, q-1]$. To allow re-encryption by readers that do not know the corresponding private key, each tag stores with each encrypted tag identifier

a corresponding re-encryption factor $(y, z) = (PK_D^{x'}, g^{x'})$, where x' is a new random value in $[0, q-1]$. Note that this is basically an encryption of the value 1 against the same key. Because ElGamal enjoys the homomorphic property that the multiplication of the encryption of two ciphertexts equals the encryption of the multiplication of the corresponding plaintexts, we see that (uy, vz) in fact equals the encryption of tag identifier t. The encrypted identifier now becomes $id = ((u, v), (y, z))$.

Readers store the key pairs for the epochs in an array $E[]$, storing the keys for epoch e at $E[e]$. This array is filled with epoch keys up to and including the current epoch ϵ, and grows in size over time.

To re-encrypt, a reader that knows the corresponding, most recent public epoch key PK_D does the following. It generates new random values a and a' in $[0, q-1]$ and computes $(u', v') = (t \cdot PK_D^a, g^a)$ and $(y', z') = (PK_D^{a'}, g^{a'})$ and sends $id' = ((u', v'), (y', z'))$ to the tag. Readers that do not know the current epoch key can use the re-encryption factor to compute a new encrypted identifier as follows. Again two random factors a and a' in $[0, q-1]$ are generated, and then the reader computes $(u', v') = (u \cdot y^a, v \cdot z^a)$ and $(y', z') = (y^{a'}, z^{a'})$ and again sends $id' = ((u', v'), (y', z'))$ to the tag

To decrypt, one simply verifies that $y/z^{sk_D} = 1$ and computes u/v^{sk_D}, using the appropriate epoch key stored in $E[]$. To avoid the need to search for the right key, the tag sends, together with is encrypted identifier, the epoch in which it was last updated[3].

5 Protocols

Below we will describe protocols that implement the operations from Sect. 2.6. We take a rather generic approach. Instead of implementing special protocols for each of these operations, we in fact model all these operations either as calls on normal objects (**DeleteObject** and **UpdateObject**), or as special methods of the tag management object Ω (all other operations). That is, we present pseudo-code for the body of each of these operations as if they were methods of a certain object, operating on the state of the object and or operating on the state of the tag. Due to space constraints, we only describe the tag-ownership operations.

5.1 Calling a Method

To call a method f on an class c, the reader d belonging to domain D and the tag t first set up a session using the protocol in Fig. 2. If this is successful, the reader and the tag share the same session key. Both initialise their message sequence counter (m and n) to 0.

The actual method call follows the protocol in Fig. 3. This protocol can be executed several times in a row, to execute several methods within a single

[3] This is not an additional privacy concern as the tag will broadcast the same encrypted tag identifier anyway, until it is successfully updated (in which instance its epoch will be set to the most recent epoch, which contains a large number of tags).

$$\begin{array}{rc} \textbf{Reader } d \in D & \textbf{Tag } t \\ (s', t') := \mathit{AuthenticateR}(E[], k_A, \epsilon) & \leftrightarrow \quad (s, \Gamma) := \mathit{AuthenticateT}() \\ n := 0 & m := 0 \end{array}$$

Fig. 2. Setting up a session

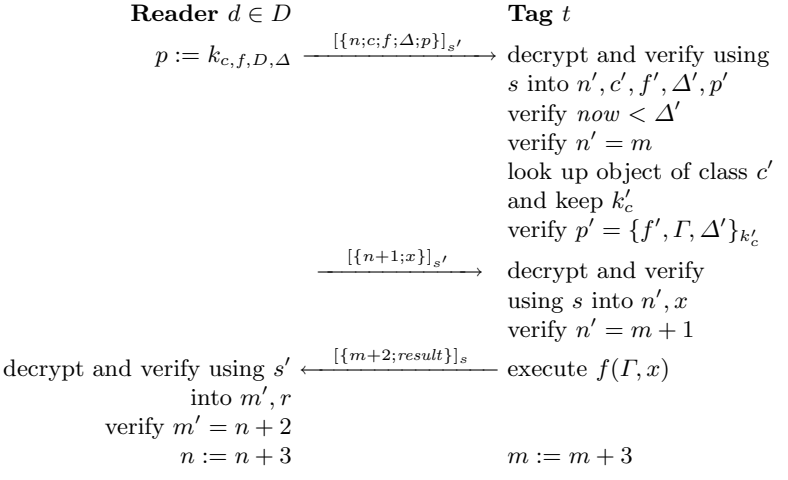

Fig. 3. Calling method $f(x)$ on class c using permission $k_{c,f,D,\Delta}$ valid until Δ

session. Each message includes the current value of the message counter, and each message is encrypted and MAC-ed with the session key. The message counters are incremented with every subsequent message within a session. The receiver verifies the included message counter to prevent replay attacks.

For each method call, the reader sends the corresponding permission token, which is verified by the tag using the class key k'_c of the class whose method is called. It also verifies whether the permission token is still valid, using its own estimate of the current time *now*, and whether the permission token is bound to the domain that was authenticated in the first phase. Then the reader sends the method call parameters, and the tag responds with the method result. If the method is supposed to return no result, a random value is sent instead. Note that the method is called with the name of the calling domain as the first parameter.

To call a method on an object for which no permission tokens are necessary (which is the case for some of the methods of the tag management object, see below), basically the same protocol is used. In this case however, the caller does not have to send a permission token, and the tag only verifies that the requested method on that object is indeed callable without permission.

Finally, to close a session, the protocol in Fig. 4 is executed.

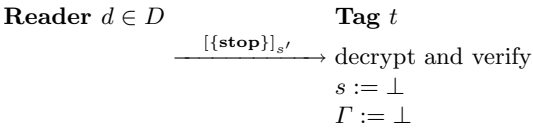

Fig. 4. Closing a session

5.2 Tag Ownership Functions

The following methods on the tag management object Ω implement transfer of ownership. To relinquish ownership of a tag, the tag owner can execute the following method. The functionality of **RelinquishTagOwnership** may be extended to include the deletion of all objects (other than the tag management object), and the resetting of information in the tag management object.

RelinquishTagOwnership(*caller*):
verify $owner = caller$;
$A := \emptyset$ (hence $owner = \perp$);
$s := \perp$.

To become the owner of an unowned tag, a domain calls the following method, where the caller of **TakeTagOwnership** from domain D has received the tag identifier t out-of-band. He then generates a random x, computes $id = (u, v) = (t \cdot PK_D^x, g^x)$ and computes $k_a = \{t\}_{k_A}$ using its own master access key k_A, before calling the method. Note that this protocol is susceptible to hijacking and eavesdropping on the new owner's access key, if the default session key $\perp$ is used (which is the case when the tag has no owner).

TakeTagOwnership($caller, D, id, k_a$):
verify $owner = \perp$;
$A[D] := (id, k_a, true)$;

To transfer ownership of tag t from tag owner T to domain T', a new entry for the new tag owner must be set in A with a new encrypted tag identifier and a new diversified access key (and in fact all other entries in the access set need to be deleted). The tag identifier does not change. This process is in fact a three party protocol that is implemented by two method calls. The first runs as follows.

TransferTagOwnership(*caller*):
verify $owner = caller$;
$A := \emptyset$ (hence $owner = \perp$) ;

Note that this function can only be executed in sessions of the authentic tag owner. After execution of this function, the session is *not* terminated (i.e. the session key is *not* reset). While in this state, the tag is shipped to the new owner T' and the values of the tag identifier id, the session key s and the message counter n are sent to T' out of band. Then, T' calls **TakeTagOwnership** (without prior authenticating and hence starting a new session!), thus becoming the new tag owner (preferably when the old owner is out of reach so it cannot eavesdrop on the new values sent to the tag).

6 Security Analysis

We first give a security analysis of the authentication protocol from Sect. 4 against the most important security properties mentioned in that section. We then analyse the security of the method invocation protocol from Sect. 5.1.

The adversary we consider has full control over the communication medium: he can block, intercept, duplicate, modify and fabricate arbitrary messages. He can, however, not create valid MACs for messages if he does not know the key, and cannot encrypt or decrypt messages for which he does not know the symmetric key. The adversary can corrupt arbitrary tags and hence know their full state including any keys they store. The adversary can also corrupt arbitrary readers. However, such readers are known to be corrupted and the system is notified of any such corruption.

Let γ be the security parameter (implicitly defined by the size of G (see 4.1) and the choice of the size of the symmetric keys).

We first prove the security of the authentication protocol.

Lemma 6.1. *Let a call* $\mathit{AuthenticateR}(sk_D, PK_D, k_A)$ *from a reader from domain* D *return* (σ, t')*. Let tag* t *call* $\mathit{AuthenticateT}()$ *which returns* (σ', D')*. Then* $\sigma = \sigma'$ *only if* $t = t'$ *and* $D = D'$*. No other entity not in domain* D *knows* σ*.*

Proof. Consider the protocol in Fig. 1. Suppose $\sigma = (s \oplus \bar{s}') = (s' \oplus \bar{s}) = \sigma'$. Then the reader accepted the message $\{q'; \bar{s}\}_{k'_a}$. Hence $k_a = \{t'\}_{k_A}$ as computed by the reader equals k'_a. As k'_a is retrieved from $A[D']$ and k_A is only known to D this proves $D = D'$ Also the tag must have accepted the message $\{id'; \epsilon; r'; q; \delta; s\}_{k_a}$ using its own key k'_a. Again for $k_a = \{t'\}_{k_A}$ we must have $k'_a = k_a$. Because only t holds $k_a = \{t\}_{k_A}$ we must have $t = t'$.

To know σ one needs to know both s and $\bar{s}$. This requires one to know k_a. Clearly t knows this. Otherwise, it requires one to know k_A (and t). This is only known to members of D. This proofs the final statement of the lemma. □

Privacy after authentication or full re-encryption is guaranteed by the following lemma.

Lemma 6.2. *Let* t *be a tag, whose tag identifier* t *for domain* D *gets re-encrypted from* id *to* id' *(either by authentication or by a full re-encryption). Let* id'' *be the encrypted tag identifier for domain* D *of an arbitrary tag* $t' \neq t$*. Then there exists no adversary (that has no access to the private keys of domain* D*) with resources polynomially bounded in* γ *that can decide whether* id' *and* id'' *or* id' *and* id *are encrypted tag identifiers of the same tag.*

Proof. In [12] it is shown that, given our use of ElGamal over our choice of group G, there does not exist an adversary with resources polynomially bounded in γ that can properly match the re-encryptions of two ciphertexts with the original input ciphertexts. The adversary linking either id or id'' with id' would trivially solve this problem too, and hence cannot exist either. □

Resilience to reader compromise is shown by the following lemma.

Lemma 6.3. *A reader from domain D reported stolen in epoch e cannot decide whether two tags that have successfully authenticated with an unstolen reader from domain D in epoch $e' > e$ corresponds with a tag observed before.*

Proof. At the start of epoch e', we have $\epsilon = e'$, and all readers in domain D that are not reported stolen receive new epoch keys $({sk_D}', PK'_D)$ that are stored in $E[\epsilon]$. If a tag authenticates with this reader, according to the protocol, it receives a new encrypted identifier encrypted with the keys $({sk_D}', PK'_D)$. Let two tags meet such a reader, obtaining encrypted tag identifiers id'_a and id'_b in exchange for their old identifiers id_a and id_b. If subsequently these tags meet a reader from domain D that was reported stolen in epoch $e < e'$, this reader does not own the key pair $({sk_D}', PK'_D)$ and hence cannot decrypt id'_a or id'_b. Therefore, by Lemma 6.2, the reader cannot link the encrypted identifiers id_a and id_b. □

Finally, we show security of the method invocation protocol.

Lemma 6.4. *A tag t only executes a method f of class c with class key k_c if a reader in domain D with*

- *$A_t[D] \neq \perp$ when it starts the session, and*
- *permission token $k_{c,f,D,\Delta} = \{f, D, \Delta\}_{k_c}$ with $\Delta > now_t$ (when the permission is verified)*

issued the command to the execute this method in the session it started. Moreover, the tag will do so at most once.

Proof. Checking the protocol, we see that a tag t executes method f on class c with class key k_c when

- it receives a message correctly encrypted and mac-ed with its session key s, containing the parameters and the expected message counter $m + 1$, and before that
- has received a message correctly encrypted and mac-ed with its session key s, containing f, c, Δ and a permission token $k_{c,f,D,\Delta} = \{f, D, \Delta\}_{k_c}$ with $\Delta > now_t$, and the expected message counter m.

The authentication protocol guarantees (see Lemma 6.1) that only if D is a member of A_t when starting a session, the reader and the tag share the same session key s. Therefore, in the current session the tag only accepts messages constructed by such a reader in domain D. This proves that it must have issued the command to the execute this method in the session it started, and also that it held the appropriate permission token. Moreover, due to the use of message counters, the current session only accepts a particular message encrypted for this session at most once. This proofs the final statement of the lemma. □

7 Concluding Remarks and Further Research

Although our model accommodates a multitude of use cases, in the course of this research we have identified several capabilities that our current implementation

lacks. Access to tags and objects is bound to specific domains, and a domain with certain permissions cannot delegate them to another domain. Although access to a *tag* can be revoked instantaneously, permission tokens to access *objects* cannot be revoked (although their validity can be constrained by using short validity periods). Domains are granted access to specific tags one by one by the respective tag owners. Permission tokens to call a method on an object are however not tag specific (unless each object of the same class is given a separate class (or rather object) key. The distinction between a permission to access a tag and a permission to call a method on an object is confusing and perhaps unfortunate.

Finally, to re-encrypt an identifier, one needs to own the corresponding access key. This severely limits the options for owners to re-encrypt their tags. On the other hand, not requiring such an access key puts tags wide open to denial-of-service attacks that feed them with bogus identifiers.

We welcome discussion and feedback on these issues.

References

[1] Anderson, R.J., Bezuidenhoudt, S.J.: On the reliability of electronic payment systems. IEEE Trans. on Softw. Eng. 22(5), 294–301 (1996)
[2] Avoine, G.: Privacy ussues in RFID banknotes protection schemes. In: 6th CARDIS, Toulouse, France, September 2004, pp. 43–48 (2004)
[3] Avoine, G., Dysli, E., Oechslin, P.: Reducing time complexity in rfid systems. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, pp. 291–306. Springer, Heidelberg (2006)
[4] Avoine, G., Lauradoux, C., Martin, T.: When compromised readers meet RFID. In: Workshop on RFID Security (RFIDsec), Leuven, Belgium, June 30–July 2, pp. 32–48 (2009)
[5] Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 531–545. Springer, Heidelberg (2000)
[6] Black, J., Rogaway, P.: CBC MACs for arbitrary-length messages: The three-key constructions. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 197–215. Springer, Heidelberg (2000)
[7] BSI. Advanced security mechanisms for machine readable travel documents – extended access control (eac). Tech. Rep. TR-03110, BSI, Bonn, Germany (2006)
[8] Dimitriou, T.: A lightweight RFID protocol to protect against traceability and cloning attacks. In: IEEE International Conference on Security and Privacy for Emerging Areas in Communication Networks, SECURECOMM 2005 (2005)
[9] Engberg, S.J., Harning, M.B., Jensen, C.D.: Zero-knowledge device authentication: Privacy & security enhanced RFID preserving business value and consumer convenience. In: 2nd Ann. Conf. on Privacy, Security and Trust, Fredericton, New Brunswick, Canada, October 13–15, pp. 89–101 (2004)
[10] Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. on Inf. Theory 31(4), 469–472 (1985)
[11] Garfinkel, S.L., Juels, A., Pappu, R.: RFID privacy: An overview of problems and proposed solutions. IEEE Security & Privacy, 34–43 (May 2005)
[12] Golle, P., Jakobsson, M., Juels, A., Syverson, P.F.: Universal re-encryption for mixnets. In: Okamoto, T. (ed.) CT-RSA 2004. LNCS, vol. 2964, pp. 163–178. Springer, Heidelberg (2004)

[13] Hoepman, J.-H., Hubbers, E., Jacobs, B., Oostdijk, M., Wichers Schreur, R.: Crossing borders: Security and privacy issues of the european e-passport. In: Yoshiura, H., Sakurai, K., Rannenberg, K., Murayama, Y., Kawamura, S.-i. (eds.) IWSEC 2006. LNCS, vol. 4266, pp. 152–167. Springer, Heidelberg (2006)
[14] Hoepman, J.-H., Joosten, R.: Practical schemes for privacy & security enhanced RFID, CoRR abs/0909.1257 (2009)
[15] ISO 7816. ISO/IEC 7816 Identification cards – Integrated circuit(s) cards with contacts. Tech. rep., ISO International Organisation for Standardisation (ISO) JTC 1/SC 17
[16] ISO 9798-2. ISO/IEC 9798 Information technology – Security techniques – Entity authentication – Part 2: Mechanisms using symmetric encipherment algorithms. Tech. rep., ISO JTC 1/SC 27
[17] Juels, A.: RFID security and privacy: A research survey. IEEE Journal on Selected Areas in Communications 24(2), 381–394 (2006)
[18] Juels, A., Pappu, R.: Squealing euros: Privacy protection in RFID-enabled banknotes. In: Wright, R.N. (ed.) FC 2003. LNCS, vol. 2742, pp. 103–121. Springer, Heidelberg (2003)
[19] Juels, A., Weis, S.: Defining strong privacy for RFID. In: 5th Ann. IEEE Int. Cont. on Pervasive Computing and Communications Workshops – Pervasive RFID/NFC Technology and Applications (PerTec), pp. 342–347 (2007)
[20] Molnar, D., Wagner, D.: Privacy and security in library RFID: issues, practices, and architectures. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) ACM Conference on Computer and Communications Security, Washington D.C., USA, October 25–29, pp. 210–219. ACM, New York (2004)
[21] NIST 800-38B. Recommendation for block cipher modes of operation: The CMAC mode for authentication. Tech. Rep. NIST Special Publication 800-38B, National Institute of Standards and Technology, U.S. Department of Commerce (May 2005)
[22] Oswald, E.: Suggested algorithms for light-weight cryptography. Tech. rep., ECRYPT (September 2006)
[23] Rieback, M.R., Gaydadjiev, G., Crispo, B., Hofman, R.F.H., Tanenbaum, A.S.: A platform for RFID security and privacy administration. In: LISA, pp. 89–102. USENIX (2006)
[24] Sarma, S.E., Weis, S.A., Engels, D.W.: RFID systems, security & privacy implications (white paper). Tech. Rep. MIT-AUTOID-WH-014, Auto-ID Center. MIT, Cambridge, MA, USA (2002)
[25] Spiekermann, S., Evdokimov, S.: Critical RFID privacy-enhancing technologies. IEEE Security & Privacy 11(2), 56–62 (2009)
[26] Stajano, F.: The resurrecting duckling - what next? In: Christianson, B., Crispo, B., Malcolm, J.A., Roe, M. (eds.) Security Protocols 2000. LNCS, vol. 2133, pp. 204–214. Springer, Heidelberg (2001)
[27] Stajano, F., Anderson, R.: The resurrecting duckling: Security issues for ad-hoc wireless networks. In: Christianson, B., Crispo, B., Roe, M. (eds.) 7th Int. Workshop on Security Procotols. LNCS, pp. 172–194 (1999)
[28] Tsiounis, Y., Yung, M.: On the security of elgamal based encryption. In: Imai, H., Zheng, Y. (eds.) PKC 1998. LNCS, vol. 1431, pp. 117–134. Springer, Heidelberg (1998)
[29] Weis, S.A., Sarma, S.E., Rivest, R.L., Engels, D.W.: Security and privacy aspects of low-cost radio frequency identification systems. In: Hutter, D., Müller, G., Stephan, W., Ullmann, M. (eds.) Security in Pervasive Computing. LNCS, vol. 2802, pp. 201–212. Springer, Heidelberg (2004)