Preprint
Id: sensorpets.tex 52 2010-02-25 14:02:07Z jhh

Practical Schemes For Privacy & Security Enhanced RFID

Jaap-Henk Hoepman · **Rieks Joosten**

February 25, 2010

Abstract Proper privacy protection in RFID systems is important. However, many of the schemes known are impractical. Some use hash functions instead of the more hardware efficient symmetric encryption schemes as a cryptographic primitive. Others incur a rather large time penalty at the reader side, because the reader has to perform a key search over large tag key space. 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 investigate such scenarios, and 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 (depending on the technology used ranging from a few centimeters up to several meters). 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 [17,23].

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. Some of these applications may be useful and beneficial for individuals and society, others may be potentially very damaging (to our personal liberties, or otherwise). It would be a waste, however, to abort such future innovations by mandating the use of a kill-switch on all RFID tags[1] that 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.

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.

Jaap-Henk Hoepman
TNO Information and Communication Technology, E-mail: jaap-henk.hoepman@tno.nl, and Institute for Computing and Information Sciences, Radboud University Nijmegen, E-mail: jhh@cs.ru.nl

Rieks Joosten
TNO Information and Communication Technology E-mail: rieks.joosten@tno.nl

[1] As recommended in EC Recommendation (SEC(2009) 585/586) of 12.5.2009 on the implementation of privacy and data protection principles in applications supported by radio-frequency identification.

Early proposals use relabelling of tag identifiers [36], or re-encryption techniques [24,2,18] 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 [41,12]. In particular, Ohkubo, Suzuki, and Kinoshita [32] present a technique for achieving forward privacy in tags. This property means that if an attacker compromises a tag, i.e., learns its current state and its key, she is nonetheless unable to identify the previous outputs of the same tag. In their protocol, a tag has a unique identifier id_i, that is changed every time the tag is queried by a reader. In fact, when queried for the i-th time, the tag responds with $g(id_i)$ to the reader, and sets $id_{i+1} = h(id_i)$ immediately after that. Dimitirou [10] presents a similar protocol, but that authenticates the tag as well. In both cases, if all readers are *on line*, connected with one central database, the readers can be synchronised and the response of a tag can be looked up immediately in the database. (Note that the database can keep a shadow copy of id_i and hence can precompute the next expected value $g(h(id_i))$.) If not, or if synchronisation errors occur, a search over all possible (initial) identifiers (expanding hash chains) is necessary.

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. This issue is not properly addressed in Engberg's paper [12]. It is unclear whether in that paper tags share a single a key with a group of other tags, or that each tag has a unique and private access key it only shares with the reader. The core challenge is therefore to provide, possibly efficient, trade offs and solutions for key search and key management. Molnar and Wagner [30] (see also [11]) propose to arrange keys in a tree structure, where individual tags are associated with leaves in the tree, and where each tag contains the keys on the path from its leaf to the root. In subsequent work Molnar, Soppera, and Wagner [29] explore ways in which the sub-trees in their scheme may be associated with individual tags. 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, like for instance a fixed number of read operations. In another approach, Avoine, Dysli, and Oechslin [3,5] show how, similar to the the study of Hellman to breaking symmetric keys, 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.* [37] 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 and management over access permissions. The RFID Guardian [35] is a notable exception. The main idea is to jam all reader to tag communication, except for reader requests that satisfy a pre-defined privacy rule. This approach has its own shortcomings. For one, it is extremely hard to ensure that all reader to tag communication is effectively blocked in all cases. Moreover, tags themselves are not protected at all, leaving them vulnerable when the Guardian is out of range or malfunctioning.

We refer to Juels [23] (and the excellent bibliography[2] maintained by Gildas Avoine) for a much more extensive survey of proposed solutions, and [26] for a more formal analysis of the privacy properties actually achieved by some of the proposed authentication protocols.

1.2 On the hardware cost of cryptography

We base our work on (relatively) new insights regarding the amount of hardware required to implement symmetric key cryptosystems as compared to hash functions. Traditionally, such hash functions are perceived to be the most basic (and therefore most efficiently implementable) building blocks, and hence have been used extensively in protocol designs for RFID. This is wrong. In fact, the ECRYPT report on light weight cryptography [33] states

> Current standards and state-of-the-art low power implementation techniques favor the use of block ciphers like the AES instead of hash functions as the cryptographic building blocks for secure RFID protocols.

[2] www.avoine.net/rfid/

Currently there is an AES-128 design with only 3k4 gates, which uses 3 μA at 100 kHz and 1.5 V in 0.35μ technology [28,34,15]. Maximum throughput is 9.9 Mbps (encryption) or 8.8 Mbps (decryption). A SHA-256 design [14,15] that also has been targeted specifically for the low end results in 10k9 gates that has a maximum throughput of 22.5 Mpbs. Current consumption is 15.87 μA at 100 kHz and 3.3 V in 0.35μ technology. Other light weight symmetric cipher designs exist as well [8]. The quoted AES design is 3 times as small (and thus also 3 times cheaper), and consumes less than 10% of the power needed by the SHA-256 design, and is only about 2.5 times slower in terms of throughput. We use these observations to design efficient protocols that incorporate tag and reader authentication with session key establishment and fine grained control and management over access permissions. Advances in reducing the hardware cost of implementing public cryptography have also been made. Current implementations of (hyper)elliptic curve cryptography require 15,4k gates, executing one scalar multiplication in 243 ms when clocked at 323 kHz [13,27]. Still the gate count and the processing time are much higher than for symmetric cryptography, making symmetric cryptography the preferred choice for lower cost tags.

1.3 Our contribution

Our contribution is to propose a model and corresponding protocols that allow fine grained, effective and efficient 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 [24,18] with symmetric cryptography based authentication [22] to obtain a reasonable level of privacy protection without using public-key cryptography on the tag, and without the need for the reader to start a time consuming key-search algorithm to find the key to use for authentication. 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. 6.

The model is quite loosely based on the "Resurrecting Duckling" paradigm of Anderson and Stajano [39, 38]. Our model is general enough to capture several RFID use case scenarios, like supply chain management, ticketing and ambient home intelligence.

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. Such an object implements a particular application or service on a tag, like a customer loyalty program, or a supply chain management program (where the on-chip object stores the identity of the physical object to which the tag is attached). This extends the notion of an RFID tag containing just a unique identifier to slightly smarter data container, resembling the technology used for the new biometric passports [20]. We believe that in the end, the idea of only storing a unique identifier on the tag and storing all relevant data on the physical object attached to the tag in a corresponding data record in a centralised database is going to prove too limitative in the future. For example, 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. We refer to Sect. 2 for more examples supporting our model.

Whether the reader is allowed to execute the function depends on two constraining factors:

1. whether the owner of the *on-chip object* has given the reader a permission to execute the particular function on the particular object, and
2. whether the owner of the *tag* allows the reader to access the tag at all.

The protocols (and the observation that the real challenge in RFID privacy lies in allowing controlled use of the RFID tag even after the point-of-sale) are inspired on the work by Engberg [12]. The first constraint is enforced using specially crafted permissions. The second constraint is enforced by the mutual reader-tag authentication protocol. This research is part of the PEARL[3] (Privacy Enhanced Architecture for RFID Labels) project.

The paper is structured as follows. We first describe a few distinctive use cases of RFID and their associated requirements in terms of functionality and privacy. In Sect. 3 we present our system model. We then show how our model captures the essence of the use cases, in Sect. 4. We then continue to implement this model using data structures (Sect. 5), an authentication and session key establishment protocol (Sect. 6) and subsequent protocols (Sect. 7). We analyse their security in Sect. 8 and present some conclusions and further research in Sect. 9.

[3] www.pearl-project.org

2 Use cases

This section describes three use-cases or scenarios, each of which focusing on one or more issues that need to be addressed by our model. In the next section we lay down our model, and then analyse how well the model captures the real life situations sketched in these use cases.

2.1 Scenario 1 - Supply Chain Management

As our first scenario, we take the supply chain scenario from [12]. In this scenario, we show the need for a proper definition of tag ownership and the controlled transfer thereof.

In retail, RFID tags are attached to individual products after they have been manufactured for the purpose of enhancing the supply chain efficiency. Such tags contain an Electronic Product Code (EPC) that not only identifies the type of product as the more classical bar codes do, but also includes a number that is unique to the individual product. This allows retailers to better keep track of their inventory and counter theft attempts, and provide better customer service as well.

Often, for privacy reasons, these RFID tags are zapped, killed or removed from products at the point of sale. This is unfortunate. Many interesting after-sale services are possible if the tag allows for a more dynamic and fine grained control of access to it by its current owner. For example, it would open the way for manufacturers and retailers to store post-sale service data onto the tag such as the production run, date of sale, warranty data, repair history etc., thus providing the them with a possibility for providing more efficient en effective service to the customer.

When a product, such as a TV set, computer, refrigerator etc. needs to be repaired at some point in time, its warranty may be voided if it is not repaired by qualified service organisations. Using the tags as introduced above, the manufacturer of equipment installs a service object into the tag associated with each of its products. In this object, the manufacturer writes all sorts of service data that it does not want to disclose to parties other than qualified service organisations. Then, it provides read permissions for such service objects only to accredited service organisations.

Many objects change owner during the course of their life, consider for example second-hand cars, electronic equipment, etc. When the objects change owner, so must the attached tags. But any objects on the tag (like the service object above) must remain on the tag and must keep their data (like the repairs performed on the object).

We can accommodate for this scenario if the following requirements are met.

- A party is allowed to communicate with a specific RFID tag if and only if (a) it is the owner of the tag or (b) it is explicitly granted permission by the owner of the tag.
- A party is allowed to access a specific data (object) on the tag if (a) it owns the data (object) or (b) the data (object) owner has explicitly enabled that party to access the data (object) using a specific method. In other words, access is assigned for individual method calls on the object.
- A party that owns a tag must be capable of transferring this ownership to another party, without deleting or changing objects.

Note that a consequence of transferring ownership of a tag is that the set of parties that were allowed to communicate with the tag changes drastically, as only the owner or a party that has a permission issued by this owner, may communicate with the tag.

2.2 Scenario 2 - Smart Tickets

Event Services Company (ESC) is large company issuing an advanced, RFID based, smart event ticketing solution. With the ESC RFID tag embedded in a wristband, users can buy tickets for music events on line, flash them in their wristband, and prove purchase of the tickets at the entry gates of the event. Security is important: ticket fraud, and especially black market sales are rampant in the ticketing business. Moreover, ESC does not want free riders to use its ticketing system: only event organisers with a contract can use it.

Tom owns such a wristband. In fact, the fashion watch he got for his birthday last year contained such an ESC tag for free. He regularly buys tickets for soccer matches and music festivals on line and appreciates the fast and paper ticket less service. Recently, the gay gym he visits changed to the ESC service, allowing members to book specific slots in the gym in advance. Tom sees this as a huge advantage (the gym can be crowded at unpredictable times), but is concerned about his privacy: he would rather not get his visits to the gym and to the soccer matches get connected. Luckily, ESC was well aware of these concerns when designing the system, and instead of storing the tickets on a central server they are all stored on the ESC tag of the user.

Apart from the requirements from the first scenario, this adds the following requirements.

- Data related to the physical object is stored on the tag, and not (necessarily) on a central server.

– Permission to install an object may be restricted depending on the application (in this case: the tag issuer).

2.3 Scenario 3 - At the Hospital

In our third scenario, we consider the situation where a doctor implants a sensor tag into one of its hospitalized patients, e.g. by having him swallow a pill containing this sensor tag. In this scenario, we show the need for a proper definition of transferal of object ownership. This scenario is inspired by the hospital scenario from [39].

Consider a patient whose heart condition, respiration and the like need to be monitored, and a high-tech monitoring device exists that acts like a tag as in the previous scenario's. Because of price and the fact that they are not needed all that often, hospitals own such devices, and only a modest amount of them.

Whenever a patient's condition is to be monitored, its doctor can decide to implant such a device into the patient, e.g. by having him swallow a pill containing the device. Within the body, the sensor starts monitoring the patient's condition, filling an object that is specific for the sensor. Doing so, a sensitive amount of personal data is gathered within the object, and it is part of the doctor's job to ensure that privacy is preserved.

Since the doctor uses the sensor, he must have pretty much full control. However, he must also be able to assign read permission to e.g. nurses. This requires him to actually own the object. Note that he should not own the device itself, as this would allow him to (dis)allow other parties access to other parts of the device as well, which, if that results in a catastrophe, will put the blame with the hospital.

We can accommodate for this by adding another requirement

– A party that owns an object must be capable of transferring this ownership to another party.

3 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.

3.1 Notation

We use k to denote a symmetric key, possibly subscripted 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 [31, 7]) using key k. Finally, $[\{m\}]_k$ denotes the authenticated encryption of m with key k, for instance by appending the MAC of the ciphertext [6].

3.2 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 [21] 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).

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

3.2.1 Classes and objects on tags

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. We use $o \in c$ to denote that object o is an instantiation of class c. 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. We write $f \in c$ for a method f that is defined for class c. Every method is defined in precisely one class. Access to a specific method is controlled in one of three ways:

– the method can be called iff the tag is not owned;
– the method can be called if the user has an appropriate permission;
– the method can be called by the domain owning the class.

The OO metaphor can be applied both to the resource constrained case where a tag contains only an identifier or a tuple of values, and to the case where complex data structures are stored on a tag.

3.2.2 The tag management 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. 7.

3.3 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 on behalf of, c.q. under the responsibility of, a domain. While a principal d may act on behalf of different domains over time, and the change frequency thereof may be very high, we assume that at any particular point in time d acts on behalf of precisely one domain D. Note that in case of natural persons, who can both act as bear responsibility, the common practice where a single name is used to refer to the person both as an actor and as a domain, may cause considerable confusion. 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. Since the domains bear the responsibilities, we have no compelling need to distinguish between the various principals that may act on behalf of a given domain, and thus we assume every domain to be inhabited by exactly one principal. We use $\mathcal{D}$ to denote the set of all domains.

3.4 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* 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 T to denote a tag owner and $\mathcal{T}$ to denote the set of tag owners, so $T \in \mathcal{T}$ and $\mathcal{T} \subseteq \mathcal{D}$. 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.

3.5 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. Also, permissions are issued to domains rather than to principals, because domains can bear responsibilities associated with using such permissions, which principals cannot. In our model, a permission that has been issued to a domain can be used by any principal that acts under the responsibility of that domain. Consequently, if misuse of a permission can be traced back to the domain the permission was issued to, this domain can be held accountable. It is outside the scope of this paper whether or not a domain limits the use of permissions that it has been assigned to a subset of the actors acting on its behalf, or sanctions misuse thereof.

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). Moreover, these permissions are implemented quite differently (as described in more detail in Sect. 5.3 and 6). The first permission is checked using a mutual tag-reader authentication protocol, which verifies that the reader domain occurs in a list of permitted domains. The second permission is implemented using a permission token that encodes the permission to access a particular method on an object. Thus, manipulation of an object on a tag is controlled both by the tag owner and object owner.

3.6 Operations on a Tag

Operations are performed by actors (readers) acting on behalf of a domain. Operations can only be performed when the actor acts on behalf of a domain that has permission to do so. While other operations are certainly conceivable, we consider only the limited set of basic operations as specified in Sect. 7.

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[4]. Tag ownership is controlled through the following functions:

- **TakeTagOwnership**: Set a specific domain as the tag's owner. Can be executed by any domain as long as the tag is not owned.
- **TransferTagOwnership**: Transfer ownership of a tag from its tag owner to another domain. Can only be executed by the owner of the tag.
- **RelinquishTagOwnership**: Relinquish ownership of a tag so that the tag is no longer owned. Can only be executed by the owner of the tag.

Tag access is controlled through the following functions:

- **GrantTagAccess**: Allow a specific domain to access a tag.
- **RevokeTagAccess**: Disallow a specific domain to access a tag.

These functions are only executable by the tag owner.

Object management is controlled through the following functions:

- **InstallObject**: Create an object and set the class key. Can only be executed by the tag owner, or any domain with a permission issued by the tag owner.
- **UpdateObject**: Overwrite the contents and the code of an object. Can only be executed by the class owner, or any domain with a permission issued by the class owner.
- **UpdateClassKey**: Change the class key associated with an object. Can only be executed by the class owner. This function can (also) be used to transfer ownership of objects.
- **DeleteObject**: Destroy an object and its associated class key. Can only be executed by the class owner, the tag owner, or any domain with an appropriate permission issued by the class owner.

As said before, this paper only describes a basic set of operations that will allow us to implement the scenarios from Sect. 2. Other operations are certainly possible and can easily be added to the model and implemented in a similar fashion as the basic operations.

4 Analysis

The system model from Sect. 3 should allow us to implement a large set of common privacy friendly uses of RFID technology. To capture these use cases, we sketched three different scenarios in Sect. 2. We now briefly verify that our model indeed allows us to implement these three scenarios. The security and privacy properties are analysed after we have presented the protocols that implement the operations – they do not depend on the model, but on the actual implementation.

4.1 Mapping of Scenario 1

Product tags that comply with our model would be attached to the product when manufactured. For every product type that a manufacturer M produces, M defines an object class *Service* that contains data and access methods that is relevant to the manufacturer, including a.o. production data, production plant, serial numbers and so on. First, M takes an unowned tag and takes ownership thereof (executing the tag's function **TakeTagOwnership**). Tag owners can then execute **InstallObject**, which is what M uses to create the *Service* object on the tag. For each of the methods on this object, M creates permissions (see Sect. 5.3) that M assigns to itself so that it can access all methods itself. Note that M only needs to create such permissions once, as they will be usable on every *Service* object M creates.

To accommodate the service-organisation scenario, all that M needs to do is create a read-permission for *Service* objects for every organisation that it has accredited for servicing M's TV sets, and send this permission to the appropriate organisation in a secure manner. This way, only accredited organisations (and M) may read *Service* objects. Note that service organisations cannot yet read the *Service* objects since they do not have permission to access the tag itself. This is done later when the consumer becomes the tag owner.

Whenever a retailer R sends M an order for a number of TV sets, M prepares the delivery. For every tag in this

[4] The word 'capability' might be more appropriate than the word 'permission'.

delivery, M first writes appropriate data into the *Service* object so that it says to which retailer it will be delivered, as well as other information M might later need. Then, M transfers ownership of the tag to R (**TransferTagOwnership**, which means that M no longer is capable of accessing the *Service* object because it can no longer access the tag (see Sect. 7.2 for details). Still, M's service object remains on the tag and all permissions that it has issued to itself and the accredited service organisations remain valid. Then, M sends some data to R in a sufficiently secure manner, thus enabling R to gain ownership of the tags (See Sect. 7.2). While the shipment is in transit, only M and R can take control of it as they have the data to regain ownership. For anyone else, the tag is useless as they cannot communicate with it.

For use in its retail processes, R has already defined an object class *Retail*, and like M, R has created and distributed permissions for *Retail* objects to itself, and other domains as necessary. Thus, when R receives the data that M has sent as well as the shipment, R can take control of each tag, and create a *Retail* object on each of them, filling it with data relevant to R's retail process. Note that the tags still contain *Service* objects, but R can only access such objects if it has been issued appropriate access permissions, i.e. if R is a service organisation that M has accredited. Also note that R controls whether or not M can access its own service object, as M needs tag-access permissions which R can grant (**GrantTagAccess**) or deny (e.g. revoke using **RevokeTagAccess**).

When a customer C buys a TV set, R updates its service object and subsequently transfers ownership of the tag to C. C may subsequently grant R and M access to the tag, that would allow them to work (only!) with their own service objects (and objects for which they have been issued a permission by the corresponding class owner). Also, C can install a data object of its own on the tag provided an appropriate class has been defined and permissions created. Also, C can resell the TV set to C' and simply transferownership of the tag to C'.

If C is not interested in managing the tag, then R may safely keep tag ownership as no other domain than R (and perhaps M) would be able to use the tag, and still then only using their own data. A more difficult situation is if C had taken up tag ownership, but sells it to a party C' that is not interested in taking tag ownership. While we think there may be several solutions here, we leave this case outside the scope of this paper. Thus, throughout the lifetime of a tag, each owner M, R, or C has full control over who can use the tag and who cannot. Also, M, R, or C can install their own data the confidentiality of which is under their own control.

4.2 Mapping of scenario 2

In this scenario, ESC (Event Services Company) is the first to take ownership of the tag using **TakeTagOwnership**. Using the default (known) class key of the tag management object Ω, it creates a permission to call **UpdateClassKey** to set the class key of Ω to its own secret value. This key is used to create permissions for every event organisation that has a contract with ESC, to install objects through Ω using **InstallObject**.

ESC transfers ownerships to consumers buying ESC tags using **TransferTagOwnership**. Now users buying tickets from certain organisations first grant access to the tag for these organisations through **GrantTagAccess**. These organisations then install their own ticket object calling **InstallObject** with the relevant ticket data on the tag. They need permission from ESC (as described in the previous paragraph) to do so.

4.3 Mapping of Scenario 3

With our model, we can show how in scenario 3 ownership of objects can be transferred between parties.

We start out with a doctor D that works at a hospital H which has a patient P and a nurse N. H owns a high-tech monitoring tag (device) T, which contains at least one object being of the class Tmon which has methods implementing all sorts of monitoring functions.

When D decides to implant T into P, D becomes responsible for the use of functions of the Tmon object. While it is undesirable to transfer ownership of T to D, it is desirable to transfer ownership of the Tmon object from H to D because this allows D to control who may use which function of the Tmon object. Thus, when D borrows T from H, H transfers ownership of the Tmon object to D (issuing a permission to D to call **UpdateClassKey** on Tmon). This immediately makes all existing permissions obsolete that H has assigned to any domain for this particular Tmon object. However, such permissions remain valid for all Tmon objects that H still owns.

Now, D can issue permissions to the Tmon object, e.g. to nurse N that nurses the patient.

When P is dismissed from the hospital, T is removed from P, and ownership of the Tmon object is returned to the hospital. This immediately invalidates the permission that N has for the Tmon object. However, as long as the validity period of this permission has not expired, N can still use it to access Tmon objects on other tags (provided N has access to the tag (which is controlled by the hospital) and the Tmon object is owned by D.

5 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.

5.1 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).

5.2 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[5] 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. 6).
- 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.

5.3 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. 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 [9, 20].

[5] This is a weakness that seems to be unavoidable: the owner of the tag can in principle recover the class key; the install procedure should ensure that the key cannot be captured in transit.

6 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[6].

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 [22] based symmetric key authentication protocol, with (universal) re-encryption [24,18] 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.

owner-controlled privacy Tag owners can re-encrypt all tag identifiers for all domains at any time on the tags they own.

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 [9,20].

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.* [25,26].

Our protocol (see Fig. 1) is an extension of the the ISO/IEC 9798-2 [22] standard, using diversified keys [1] to avoid sharing keys over many tags[7]. The tag stores such a diversified tag access key k'_a that corresponds to $\{t\}_{k_A}$. To compute this diversified key 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 [24] 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 the most recent epoch key. This makes it impossible for compromised readers to track this tag.

Note that corrupt readers can update the tag identifier to an arbitrary value. If that value is not recognised as a tag identifier by a genuine reader in a next authentication run, this reader will send random data to the tag. The tag will detect this and set $A[D] := \perp$. The tag will then stop responding to requests from this domain. Without this countermeasure, the arbitrary value for the identifier would never change and the tag would be traceable forever.

[6] Actually, from a privacy perspective, we are only interested in authenticating the *reader*. Only after the reader is proven authentic, and has permission to access the tag, the tag has to identify and authenticate itself. However, since we are unable to use public key cryptography on the tag, and we do not wish to store global shared secrets on the tag, we are left with using key diversification based on the identity of the tag. Hence authenticating the reader as well as the tag simultaneously seems to be the only way forward.

[7] The first encrypted message is also protected by a MAC, because the contents of the message should not malleable while keeping the response to the challenge intact. This is not guaranteed if one only encrypts the message, e.g., in ECB mode.

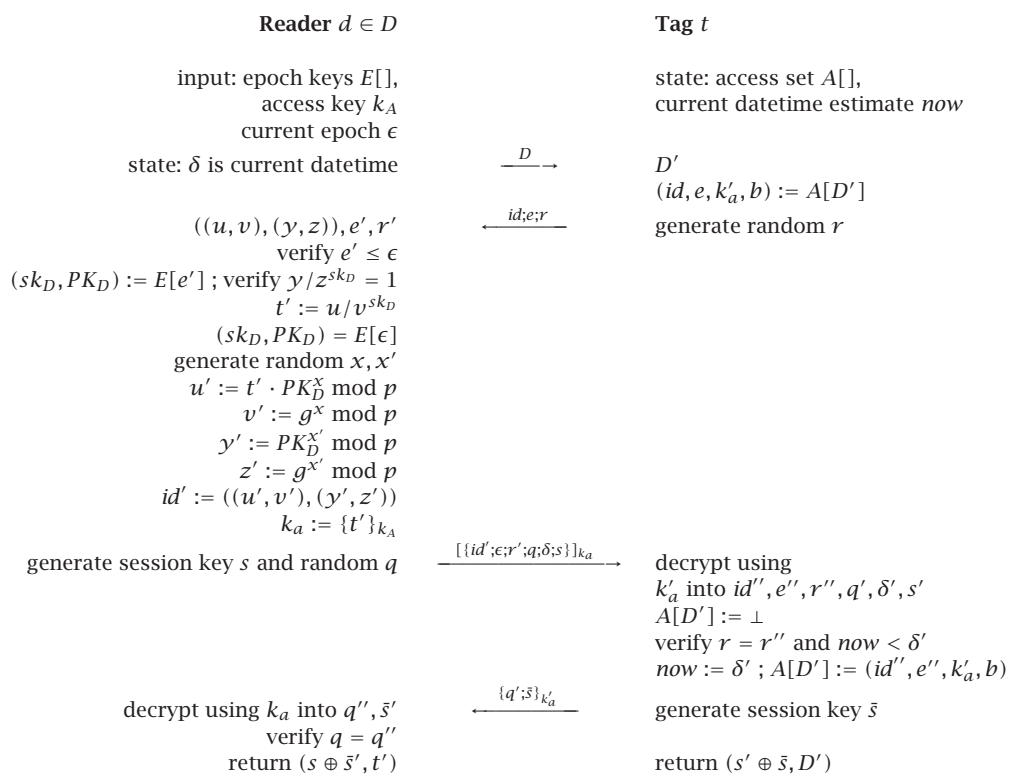

Fig. 1 Authentication and session key agreement.

The protocol can be extended using ideas from Hoepman *et al.* [19] by storing so called authentication credit on the readers, that cannot be used to generate valid authentication responses. This way, readers do not need to store master keys, and therefore need to be less trusted, or can be operated in less trusted environments.

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

6.1 Re-encryption

The protocol uses re-encryption, or rather *universal* re-encryption [18], as follows. We use the ElGamal encryption scheme [16] over a cyclic group G of order q. To be concrete, and to achieve semantic security [40], 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.

Requests to re-encrypt other encrypted tag identifiers can be issued by authorised readers to the tag management object, see Sect. 7.4. Typically, readers that are owned and operated by a tag owner will have the permission to perform such re-encryptions. This way, owners of tags have control over how easily their tags can be traced. Without universal re-encryption, only readers knowing the public key of the domain can re-encrypt. If a tag is hardly ever accessed by such a reader (consider for example a supply chain tag attached to a piece of clothing that is never accessed again after the point of sale), such a tag is principle unlimitedly traceable. By frequently re-encrypting their tags, users can make such tags much less traceable.

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[8].

[8] This impacts privacy, in particular it allows one to trace tags that are infrequently used and hence broadcast old epoch numbers. However, in the current protocol that is not a separate concern, as the same tag will broadcast the same encrypted tag identifier 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).

6.2 Alternative approaches

In the course of developing the above algorithm, we have considered various alternatives. The main drawback of the above protocol is that tags are traceable in between re-encryptions. Every malicious reader that claims to be from domain D will receive the current encryption of the identifier. This can be solved in two ways, both incurring another, more severe, drawback.

The first option is to let the tag (instead of the reader) do the re-encryption each time it is queried by a reader. Then the tag is no longer dependent on a reader to provide it with a proper re-encryption, and malicious readers no longer pose a threat. But this requires that the tag is capable of performing modular exponentiation at reasonable speed. This is out of scope for low cost tags. Moreover, if the tag can do that, then one might as well use an authentication protocol using asymmetric cryptography. Such a protocol would be much simpler than our current proposal.

The second option is to stop responding to requests from domain D after a fixed number of times, unless one such request was a full run of the authentication protocol that updated the current encryption of the identifier. This limits the time a tag can be traced, but makes the system vulnerable to denial of service attacks.

Finally, we considered another approach where the tag would randomly encrypt its tag identifier to a symmetric domain key k_D, sending

$$\{r, t\}_{k_D}$$

to the reader at the start of the authentication protocol[9]. By including the random r, the whole message is randomised, and tags become untraceable. However, k_D is stored on all tags accessible by domain D. Because tags are not tamper proof, this key is not protected and will become known after some time. From that time on, these tags become traceable and privacy is lost.

7 Protocols

Below we will describe protocols that implement the operations from Sect. 3.6. We take a rather generic approach. Instead of implementing special protocols for each of these operations, we in fact model all these

[9] This message should *not* be encrypted in ECB mode, but in CBC mode (if the nonce and the tag identifier together do not fit inside a single block). The point is that the random value r preceding the tag identifier should randomise the encryption of the *whole* message, in particular the encryption of t, to preserve the privacy of the tag.

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 pseudocode 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.

This way, the only 'protocol' that we need to describe now is how to securely call a method on an object stored on a tag. In fact, this protocol is split in three sub-protocols. The first sets up a session and a shared session key, the second securely calls the method using the session key to secure the channel and using permission tokens to verify the legitimacy of the request, and the third closes the session.

Note (cf. Sect. 6) again that we do not explicitly describe the behaviour of a principal if it detects an error.

7.1 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. This is nothing more than starting the authentication protocol from Fig. 1. 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. 4. This protocol can be executed several times in a row, to execute several methods within a single 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.

Reader $d \in D$		**Tag** t
input: keys k_A, E[]		
epoch ϵ		
$(s', t') :=$		$(s, \Gamma) :=$
$AuthenticateR(E[\,], k_A, \epsilon)$	$\leftrightarrow$	$AuthenticateT()$
$n := 0$		$m := 0$

Fig. 2 Setting up a session.

Reader $d \in D$		**Tag** t
session key s'		session key s
	$\xrightarrow{[\{\textbf{stop}\}]_{s'}}$	decrypt and verify using s
		$s := \perp$
		$\Gamma := \perp$

Fig. 3 Closing a session.

Finally, to close a session, the protocol in Fig. 3 is executed.

7.2 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.

RELINQUISHTAGOWNERSHIP*(caller)* :
verify *owner* = *caller* ;
$A := \emptyset$ (hence *owner* $= \perp$)[10] ;
$s := \perp$.

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.

To become the owner of an unowned tag, a domain calls the following method

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

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

[10] If so desired, resetting of A can be skipped. However, in that case the owner flag for Γ must be reset.

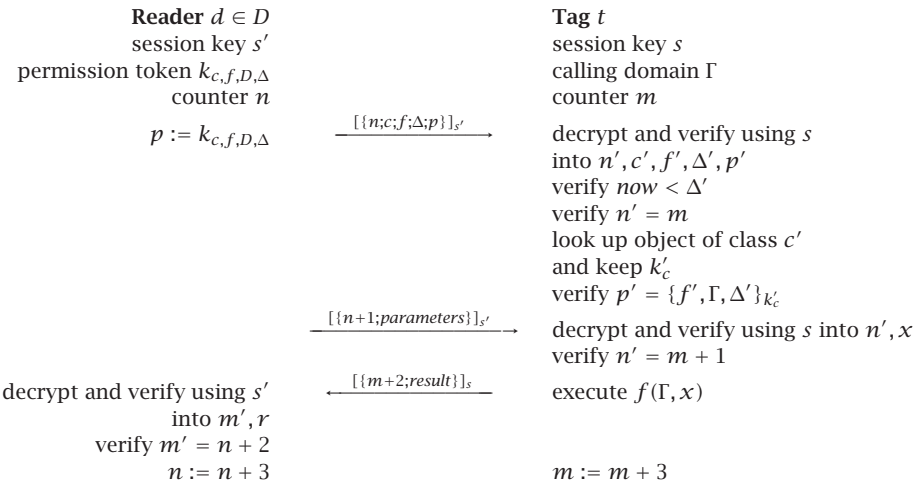

Fig. 4 Calling method f on class c using permission token $k_{c,f,D,\Delta}$ valid until Δ.

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).

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(ated) 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).

We note that the above described method might pose problems for domains that need to take ownership for many tags, as e.g. electronics manufacturers or retailers may do (see Scenario 1). They would face a practical problem of how to determine which tag would be associated to which tag identifier and which session key to use, which could easily become an adminstrative nightmare. Also, it would be more in line with Anderson's Duckling protocol [39,38] if anyone can just take ownership of an unowned tag without any other knowledge. For unowned (and unowned only) tags one could enable a method that returns the unencrypted tag identifier. To transfer the ownership of many tags using a single session key, one could extend the method **TRANSFERTAGOWNERSHIP** with an additional parameter s to set the session key on the tag to a fixed value.

7.3 Granting access to a domain

To grant a domain D access to a tag t, its access set entry $A_t[D]$ needs to be set with a new encrypted tag identifier and a new diversified access key. This process is again a three party protocol that is implemented by two method calls. None of these methods require additional permission tokens to be executed. The first method called (by the tag owner) is

GRANTTAGACCESS*(caller, D)* :
verify *owner* = *caller* ;
$A[D] := \perp$;

The tag identifier, the value of the session key as well as the value of the message counter n are sent to the domain D out of band. He subsequently calls (*not* authenticating and starting a new session!)

ACCEPTTAGACCESS*(caller, D, id, k_a)* :
verify $A[D] = \perp$;
$A[D] := (id, k_a, false)$;

computing $id = (u, v) = (t \cdot PK_D^x, g^x)$ and $k_a = \{t\}_{k_A}$ as in the case of **TAKETAGOWNERSHIP**. Note that the remarks made for **TAKETAGOWNERSHIP** with respect to the need to communicate the tag identifier, apply here equally well. Also, an improvement to these functions can be made if it would not be necessary to have

a pending session in between the calling of **GrantTagAccess** and **AcceptTagAccess** as a refusal to execute **AcceptTagAccess** would constitute a denial of service.

RevokeTagAccess(*caller*, *D*) :
verify *owner* = *D* ;
$A[D] := \perp$;

7.4 Re-encrypt identifiers

The following two functions allow a reader to re-encrypt all encrypted tag identifiers stored in A. First the reader retrieves the current encrypted tag identifiers in an array through the following method.

ReEncryptGetIds(*caller*) :
verify *owner* = *caller* ;
return a list of all encrypted tag identifiers in A ;

The reader then computes the re-encryption of each of the entries in A as described in Sect. 6.1, creating a new array R. Finally, to upload the new entries to the tag, it calls the following method.

ReEncryptPutIds(*caller*, *R*) :
verify *owner* = *caller* ;
store each entry in R in the corresponding location in A ;

Both methods can only be called by the tag owner. Alternatively, one could require that the caller owns a permission token to call the method.

7.5 Managing objects

Managing an object involves the creation, deletion or update of the object on a particular tag t. These are handled by the following methods.

To install an object, one needs to call the following method on the object manager object Ω. Depending on requirements, one may decide that further permission tokens are necessary, or instead require a specific permission token from the tag management object .

InstallObject(*caller*, *i*, *o*, *k*) :
verify *owner* = *caller* ;
verify that object with name i does not exist on the tag yet ;
create a new object o with name i with class key k ;

To update or delete an object, one needs to call one of the following methods *on the object to be updated or deleted*. Additional permission tokens from that object may be required. Only the owner of a tag can delete an object.

UpdateObject(*caller*, *i*, *o*) :
update object with name i to o ;
UpdateClassKey(*caller*, *i*, *k*) :
update the class key of object with name i to k ;
DeleteObject(*caller*, *i*) :
verify *owner* = *caller* ;
verify $i \neq \Omega$;
delete the object with name i ;

Note that by implementing object management this way, objects can only be managed by domains that

- have access to the tag because they are a member of its access set A, and
- have the correct permission token for the tag management object Ω, issued using its class key k_Ω.

Note that the tag management object itself can also be updated this way (and in particular its key), but cannot be removed or created. When tags are created, a default tag management object is present on the tag.

Also note that neither the tag owner nor the owner of the tag management object is capable of removing objects that they do not own, or do not have a delete permission for. In order to prevent tags becoming unusable because of the multitude of objects installed on it, one might consider to extend the functionality of **RelinquishTagOwnership** to include the deletion of every object (except, of course, the tag management object) on the tag.

8 Security analysis

We first give a security analysis of the authentication protocol from Sect. 6 against the most important security properties mentioned in that section. We then analyse the security of the method invocation protocol from Sect. 7.1.

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 can 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 6.1) and the choice of the size of the symmetric keys).

We first prove the security of the authentication protocol.

Lemma 1 *Let a reader from domain D call the function AuthenticateR(sk_D, PK_D, k_A) which returns (σ, t'). Let tag t call 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 $D = D'$

Similarly 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 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 [18] 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 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 that authentication.*

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 2, the reader cannot link the previous encrypted identifiers id_a and id_b. □

Finally, we show security of the method invocation protocol.

Lemma 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 maced 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 maced 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 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. □

9 Concluding remarks and further research

We have presented a model for a fine grained and dynamic management of access permissions to RFID tags, and we have presented privacy friendly protocols efficiently implementing this model. This efficiency is achieved by avoiding a costly key search algorithm at the reader

side. The price to pay is a little less privacy: tags can be traced in between successful authentications by legitimate readers. However, this is mitigated quite effectively by giving tag owners the possibility to re-encrypt tag identifiers at any point in time.

Although the 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. A domain with certain permissions cannot delegate them to another domain. Instead new permissions have to be requested from the tag owner and the class owner.
- 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).
- Another approach to limit validity of permissions is to issue one-time only permission tokens that can be used exactly once to call a particular method on an object.
- 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. This distinction arises from two factors. First, access to a tag is issued by the current owner, and is maintained on the tag to allow immediate revocation of access. Moreover, the privacy friendly authentication protocol needs to know which domains have access to the tag – hence tag related access control decisions are taken at a lower layer than object related access control decisions.
- 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.

Further research is necessary to see whether these capabilities are truly necessary in real-life applications, and, if so, how these capabilities can be added efficiently. We welcome discussion and feedback on these issues.

References

1. Anderson, R. J., and Bezuidenhoudt, S. J. On the reliability of electronic payment systems. *IEEE Trans. on Softw. Eng. 22*, 5 (May 1996), 294–301.
2. Avoine, G. Privacy ussues in RFID banknotes protection schemes. In *6th CARDIS* (Toulouse, France, Sept. 2004), pp. 43–48.
3. Avoine, G., Dysli, E., and Oechslin, P. Reducing time complexity in rfid systems. In *Selected Areas in Cryptography* (2005), B. Preneel and S. E. Tavares, Eds., vol. 3897 of *Lecture Notes in Computer Science*, Springer, pp. 291–306.
4. Avoine, G., Lauradoux, C., and Martin, T. When compromised readers meet RFID. In *Workshop on RFID Security (RFIDsec)* (Leuven, Belgium, June 30–July 2 2009), pp. 32–48.
5. Avoine, G., and Oechslin, P. A scalable and provably secure hash-based rfid protocol. In *PerCom Workshops* (2005), IEEE Computer Society, pp. 110–114.
6. Bellare, M., and Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In *ASIACRYPT* (2000), T. Okamoto, Ed., LNCS 1976, Springer, pp. 531–545.
7. Black, J., and Rogaway, P. CBC MACs for arbitrary-length messages: The three-key constructions. In *CRYPTO* (2000), M. Bellare, Ed., LNCS 1880, Springer, pp. 197–215.
8. Bogdanov, A., Knudsen, L., Leander, G., Paar, C., Poschmann, A., Robshaw, M., Seurin, Y., and Vikkelsoe, C. Present: An ultra-lightweight block cipher. (to appear).
9. BSI. Advanced security mechanisms for machine readable travel documents – extended access control (eac). Tech. Rep. TR-03110, BSI, Bonn, Germany, 2006.
10. 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).
11. Dimitriou, T. A secure and efficient RFID protocol that could make big brother (partially) obsolete. In *PerCom* (2006), IEEE Computer Society, pp. 269–275.
12. Engberg, S. J., Harning, M. B., and 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, Oct. 13–15 2004), pp. 89–101.
13. Fan, J., and Verbauwhede, I. Hyperelliptic curve processor for RFID tags. In *Workshop on RFID Security (RFIDsec)* (Leuven, Belgium, June 30–July 2 2009), pp. 129–139.
14. Feldhofer, M., and Rechberger, C. A case against currently used hash functions in rfid protocols. In *OTM Workshops (1)* (2006), R. Meersman, Z. Tari, and P. Herrero, Eds., LNCS 4277, Springer, pp. 372–381.
15. Feldhofer, M., and Wolkerstorfer, J. Strong crypto for rfid tags - a comparison of low-power hardware implementations. In *Int. Symp. on Circuits and Systems (ISCAS 2007)* (New Orleans, LA, USA, May 20–27 2007), pp. 1839–1842.
16. Gamal, T. E. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory 31*, 4 (1985), 469–472.
17. Garfinkel, S. L., Juels, A., and Pappu, R. RFID privacy: An overview of problems and proposed solutions. *IEEE Security & Privacy* (May June 2005), 34–43.
18. Golle, P., Jakobsson, M., Juels, A., and Syverson, P. F. Universal re-encryption for mixnets. In *RSA Conf.* (San Fransisco, CA, USA, Feb. 23–27 2004), LNCS 2964, pp. 163–178.
19. Hoepman, J.-H. Secret key authentication with software-only verification. In *4th Int. Conf. Fin. Crypt.* (Anguilla, British West Indies, Feb. 20–24 2000), Y. Frankel, Ed., LNCS 1962, Springer, pp. 313–326.
20. Hoepman, J.-H., Hubbers, E., Jacobs, B., Oostdijk, M., and Wichers Schreur, R. Crossing borders: Security and privacy issues of the european e-passport. In *1st IWSEC* (Ky-

oto, Japan, Oct. 23–24 2006), H. Yoshiura, K. Sakurai, K. Rannenberg, Y. Murayama, and S. Kawamura, Eds., LNCS 4266, Springer, pp. 152–167.
21. ISO 7816. ISO/IEC 7816 Identification cards – Integrated circuit(s) cards with contacts. Tech. rep., ISO JTC 1/SC 17.
22. 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.
23. Juels, A. RFID security and privacy: A research survey. *IEEE Journal on Selected Areas in Communications 24*, 2 (2006), 381–394.
24. Juels, A., and Pappu, R. Squealing euros: Privacy protection in RFID-enabled banknotes. In *7th Int. Conf. Fin. Crypt.* (Guadeloupe, French West Indies, Jan. 27–30 2003), R. N. Wright, Ed., LNCS 2742, Springer, pp. 103–121.
25. Juels, A., and Weis, S. Defining strong privacy for rfid. (technical report), Apr. 6 2006.
26. Juels, A., and 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)* (2007), pp. 342–347.
27. Lee, Y. K., Sakiyama, K., Batina, L., and Verbauwhede, I. Elliptic-curve-based security processor for rfid. *IEEE Trans. Computers 57*, 11 (2008), 1514–1527.
28. Martin Feldhofer, Johannes Wolkerstorfer, V. R. Aes implementation on a grain of sand. *EE Proceedings on Information Security 152*, 1 (Oct. 2005), 13–20.
29. Molnar, D., Soppera, A., and Wagner, D. A scalable, delegatable pseudonym protocol enabling ownership transfer of rfid tags. In *Selected Areas in Cryptography* (2005), B. Preneel and S. E. Tavares, Eds., vol. 3897 of *Lecture Notes in Computer Science*, Springer, pp. 276–290.
30. Molnar, D., and Wagner, D. Privacy and security in library rfid: issues, practices, and architectures. In *ACM Conference on Computer and Communications Security* (Washington D.C., USA, Oct. 25–29 2004), V. Atluri, B. Pfitzmann, and P. D. McDaniel, Eds., ACM, pp. 210–219.
31. 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.
32. Ohkubo, M., Suzuki, K., and Kinoshita, S. Efficient hash-chain based rfid privacy protection scheme. In *International Conference on Ubiquitous Computing (Ubicomp), Workshop Privacy: Current Status and Future Directions* (2004).
33. Oswald, E. Suggested algorithms for light-weight cryptography. Tech. rep., ECRYPT, Sept. 2006.
34. Poschmann, A., Leander, G., Schramm, K., and Paar, C. New light-weight crypto algorithms for rfid. In *Int. Symp. on Circuits and Systems (ISCAS 2007)* (New Orleans, LA, USA, May 20–27 2007), pp. 1843–1846.
35. Rieback, M. R., Gaydadjiev, G., Crispo, B., Hofman, R. F. H., and Tanenbaum, A. S. A platform for rfid security and privacy administration. In *LISA* (2006), USENIX, pp. 89–102.
36. Sarma, S. E., Weis, S. A., and Engels, D. W. Rfid systems, security & privacy implications (white paper). Tech. Rep. MIT-AUTOID-WH-014, Auto-ID Center, MIT, Cambridge, MA, USA, 2002.
37. Spiekermann, S., and Evdokimov, S. Critical rfid privacy-enhancing technologies. *IEEE Security & Privacy 11*, 2 (Mar.–Apr. 2009), 56–62.
38. Stajano, F. The resurrecting duckling - what next? In *8th Security Protocols Workshop* (Cambridge, UK, Apr. 3–5 2000), B. Christianson, B. Crispo, and M. Roe, Eds., LNCS 2133, Springer, pp. 204–214.
39. Stajano, F., and Anderson, R. The resurrecting duckling: Security issues for ad-hoc wireless networks. In *Security Procotols, 7th Int. Workshop* (1999), B. Christianson, B. Crispo, and M. Roe, Eds., LNCS, pp. 172–194.
40. Tsiounis, Y., and Yung, M. On the security of elgamal based encryption. In *Public Key Cryptography* (1998), H. Imai and Y. Zheng, Eds., LNCS 1431, Springer, pp. 117–134.
41. Weis, S. A., Sarma, S. E., Rivest, R. L., and Engels, D. W. Security and privacy aspects of low-cost radio frequency identification systems. In *1st SPC* (Boppard, Germany, Mar. 12–14 2003), D. Hutter, G. Müller, W. Stephan, and M. Ullmann, Eds., LNCS 2802, Springer, pp. 201–212.