Physical aggregated objects and dependability

Fabien Allard, Michel Banâtre, Fabrice Benhamouda, Paul Couderc, Jean-François Verdonck

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Physical aggregated objects and dependability

Fabien Allard — Michel Banâtre — Fabrice Ben Hamouda — Paul Couderc — Jean-François Verdonck

N° 7512

January 2011

Networks, Systems and Services, Distributed Computing

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7512--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Physical aggregated objects and dependability

Fabien Allard*, Michel Banâtre†, Fabrice Ben Hamouda‡, Paul Couderc§ , Jean-François Verdonck¶

Domain : Networks, Systems and Services, Distributed Computing
Équipe-Projet ACES

Rapport de recherche n° 7512 — January 2011 — 30 pages

Abstract: This documents deals with dependability issues of aggregated objects and RFID-based systems. It analyses the different categories of issues raised by these objects and proposes some measures to face them. It also provides a state of the art of actual implementations of those solutions with multiple comparisons.

Key-words: ubiquity, dependability, RFID, security, cryptography, physical objects, aggregation, coupling, signature, authentication

* Fabien Allard — SenseYou — Immeuble Gallium 80 Avenue des Buttes de Coesmes 35 700 RENNES CEDEX FRANCE - France — Email: fabien.allard@senseyou.fr

† Michel Banâtre — INRIA Rennes — Campus Universitaire de Beaulieu 35042 Rennes Cedex - France — Email: banatre@irisa.fr

‡ Fabrice Ben Hamouda — Stagiaire ACES (Elève Normalien ENS ULM) — Email: fabrice.ben.hamouda@ens.fr

§ Paul Couderc — INRIA Rennes — Campus Universitaire de Beaulieu 35042 Rennes Cedex - France — Email: paul.couderc@inria.fr

¶ Jean-François Verdonck — INRIA Rennes — Campus Universitaire de Beaulieu 35042 Rennes Cedex - France — Email: jean.francois.verdonck@inria.fr

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Objets physiques agrégés et sûreté de fonctionnement

Résumé : Ce document expose les différents aspects liés à la sûreté de fonctionnement d'applications s'appuyant sur les objets physiques aggrégés et la technologie RFID. Il expose les différentes catégories de problèmes susceptibles de compromettre le service et propose des solutions adaptées.

Mots-clés : ubiquité, sûreté de fonctionnement, RFID, sécurité, cryptographie, objets physiques, agrégation, couplage, signature, authentification, non authenticité

Introduction

Checking for integrity of a set of objects is often needed in various activities, both in the real world and in the information society. The basic principle is to verify that a set of objects, parts, components, people remains the same along some activity or process, or remains consistent against a given property (such as a part count).

While there are very few automatic solutions to improve the situation in the real world, integrity checking in the computing world is a basic and widely used mechanism: magnetic and optical storage devices, network communications are all using checksums or other error checking codes to detect information corruption, to name a few.

The emergence of Ubiquitous computing and the rapid penetration of RFID (Radio Frequency IDentification) led to development of security solutions bringing those techniques to the physical world. They can provide services such as theft detection, alarm triggering, access control... As an example, plane travellers could get warned if any of their piece of luggage is missing or got swapped with an other traveller when passing through a checking portal.

It appeared that integrity checking systems based on RFID tags could face various security issues. However, RFID tags are highly exposed to various attacks which could compromise the service.

The aim of this paper is to focus on how such security applications can be strengthened in order to resist those attacks.

1 Aggregated objects and basic concepts

1.1 Basic aggregated objects

Basic **aggregated objects** are sets of mobile and/or physically independent objects, called **fragments**.

First, fragments can be aggregated by an **aggregating system** using an **aggregating algorithm**.

Then, integrity of the resulting aggregated object can be tested at any time thanks to a **verifying system** using a **verifying algorithm**, inside a **verifying area**.

Basically, a verifying system computes the integrity information of a set of fragments brought in its verifying area and then uses it as an action trigger. For example, it could open a door when a complete set of fragments forming an aggregate is found, or trigger an alarm otherwise.

1.2 Example of uses

This section illustrates how aggregate-based systems can solve several concrete security issues.

Two examples are to be depicted: Ubi-Check and Ubi-Park. Both projects are direct application of the described basic aggregating mechanisms and improve security.

1.2.1 Ubi-Check

Ubi-Check helps travellers not forgetting one of their items, or mistakenly exchanging a similar one with someone else. During the check-in, each passenger is aggregated with all his items (cell phone, passport and suitcase for instance) using RFID stickers. After leaving the plane, passengers get their luggage integrity checked when passing through a portal. If an item is missing, an alarm can be triggered or a message displayed. More information about Ubi-Check can be found in [16].

1.2.2 Ubi-Park

Ubi-Park is a standalone system aiming at providing access control and monitoring to a bike shed (see Figure 1). It grants access to any user coupled with his bike. Users are equipped with a unique tag and their bike has to carry one aggregated object (at least one tag). The minimum equipment is an RFID portal next to the door that is able to communicate with a user's and his bike's tags.

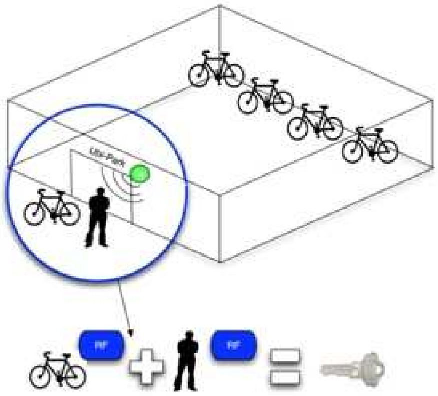

Figure 1: Ubi-Park

The key enabling to access the shed is the coupled object. People can only enter the shed with their bike, or alone if their bike is already inside it. The same way, they cannot exit with somebody else's bike as it would not be coupled with them.

1.3 Analogy with packet switching

One can compare this mechanism with packet switching. When a sender wants to send a file to a receiver through a network, it cuts the file into packets and send these independent packets through the network. The receiver receives the packets, put them together and verify the integrity of the result.

Packets correspond to fragments, the files to aggregated objects and the areas to receivers. Outside verifying areas, packets are completely independent, free to move and can even use different ways to go from the sender to the receiver. In Ubi-Check, passengers and their luggage could travel separately.

Another important special feature is that information needed to reconstruct the file is in packets themselves. This is also the case with aggregated objects: data needed to know which fragments are part of aggregated objects are stored on fragments themselves (there is no need for an external database).

1.4 Typical architecture

A typical application can be segmented in 5 main layers shown in Figure 2.

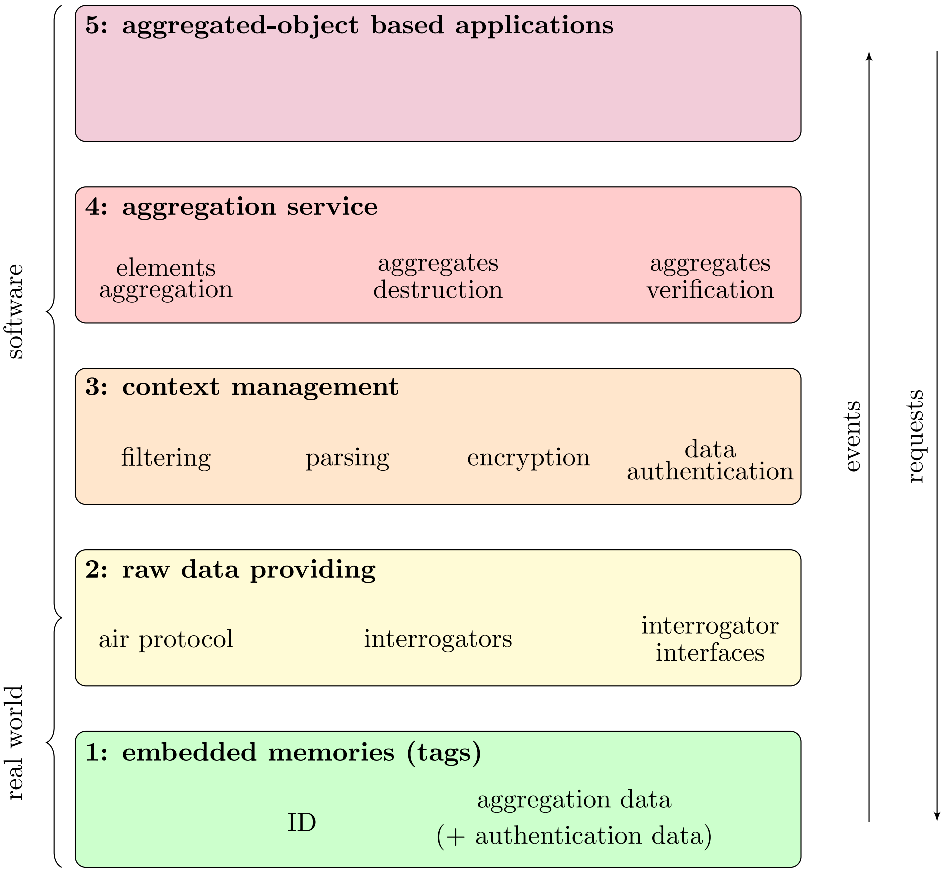

Figure 2: Global architecture layers

In the proposed architecture, reads of fresh data are propagated using an event-driven model, from the first to the last layer. Raw data that are physically written on the tag memory (layer 1) are read by the raw data providing layer (layer 2). Layer 3 is in charge of filtering, parsing, decrypting and authenticating those data. It builds a virtual context upon those data and notifies the upper layer (layer 4: aggregation service) of any of its changes. Every time a context notification is sent, the aggregation service (layer 4) searches for aggregates structures in the context and notifies aggregate-based applications (layer 5) of any structure change. Each level can also ask for lower level operations to the previous layers. As an example, a checking algorithm from layer 4 can ask for tag data authentication to layer 3 before notifying layer 5.

2 Dependability threats

This section deals with dependability threats of aggregate-based systems (like UbiCheck — section 1.2.1 — or UbiPark — section 1.2.2). Any obstacle to availability, reliability, safety, confidentiality, integrity or maintainability will be considered as a threat to the system dependability. Threats are faults, errors and failures. Faults may lead to errors, and errors to failures (see figure 3). More details can be found in [11].

Faults
Errors
Failures

Figure 3: Fault-Error-Failure chain

This study will focus on intentional attacks against RFID implementations, starting the analysis from the failures to the faults. Dependability impairments may vary according to application designs, but most of the failures, faults and errors are common to almost all aggregate-based systems. Given examples will be based on UbiCheck and UbiPark. Next section will deal with possible solutions.

2.1 Failures

Failures are deviation of the system from specified results. Some of the objectives are common to all applications, some are specific. Here is a description of the main failures.

2.1.1 Unauthorized use of a service

This failure occurs when the verifying system provides a service to an unauthorized person. In UbiPark, it would occur if the system allows a user who did not subscribed to use it and secure his bike for free. Moreover, this failure may lead to more critical failures as an attacker could get its job eased inside the shed. The main dependability attribute affected by this failure is safety.

2.1.2 Denial of service to authorized persons

This failure occurs when the verifying system denies its service to an authorized person. In UbiPark, this would happen if a user in order could not enter or exit the shed. Most of the time, it has no catastrophic consequences. The main dependability attribute affected it affects is availability.

2.1.3 Privacy leaks

Privacy leaks occur when an attacker is able to retrieve personal information about users from the system. Obviously, the main dependability attribute affected by this failure would be confidentiality.

A verifying algorithm does not need nominative user information nor database connexion to perform aggregates checking, so aggregate-based system do not expose a lot of private data. Anyway, it is really easy to identify the tags IDs corresponding to a specific user and start tracking his tags. More information about privacy threats can be found in [4]. Moreover, if aggregating data[1] are not encrypted, it may be possible for an attacker to find all the fragments of an aggregate. In Ubipark, this would enable to find somebody's bike thanks to the user badge.

2.1.4 Specific application failure (substitution, theft, vandalism...)

As applications use action triggers of verifying systems to control specific processes, a wrong behaviour of this system could cause an application specific failure. As an example, UbiCheck was designed to bring a protection against theft and accidental substitution. Thus, main failures would be theft and substitution. The main dependability attribute affected by this failure is reliability.

2.2 Errors

An error corresponds to an unexpected state of the system due to the activation of a fault. In aggregate-based applications, most of the errors can be considered as inconsistencies between reality (real aggregated objects created by authorized systems) and the state (set of aggregated objects) detected by a verifying algorithm. Most of them lead to failures. Some of them can be detected by the system, enabling exception throwing, while some cannot.

2.2.1 Illegal appearance or disappearance of a tag

In some contexts, there is no good reason for tags to appear or disappear from a defined area or read point.

If aggregated objects appear where they should not, they would compromise the integrity of the whole system and could lead to application specific failures. In UbiPark, a complete aggregate is a key to the exit. A key that would suddenly pop up inside the shed while the door is closed (as an example, it could be thrown through a grating) would be suspicious and could enable theft.

The same way, the disappearance of a tag would produce an inconsistency as one of the item sets would no longer be seen as integral even if no physical object is missing. This could lead to a denial of service (DoS).

Both situations can be detected using a reader that would monitor the whole area of the shed.

2.2.2 Tag swapping

Swapping tags from two different objects would introduce an inconsistency between objects and aggregate structure. An attacker could cause this error in UbiPark to steal a bike, leading to a substitution failure, without any RFID knowledge. There is no easy way to detect this error.

2.2.3 Forged fragment tags

Genuine tags, are tags that are meant to be used with the service and produced by an authorised aggregating authority. If genuine tag are cloned, modified or

[1] Aggregating data are produced by aggregating algorithms and carried by the tags. They store the structure which is given to the physical objects tags are attached to.

illegally built from scratch, the service could be used without authorisation, deny its service, or be compromised (specific application failure).

This error can be detected if there is a way to authenticate fragments (see section 3.2 and 3.4).

2.2.4 Presence of parasite tags

In some applications, the presence of an additional incomplete aggregated object in the control area may cause trouble. As an example, UbiPark allows one and only one bike/user couple to cross the door so the user cannot exit with his bike and another. This could lead to a denial of service: if an UbiPark tag is stuck near the door, the system would not allow anybody to enter or exit the shed.

A parasite tag can be genuine or not. Non genuine tags may be detected (see previous error). If the parasite tag is genuine, they are some situations where it can be detected. In Ubipark, a tag staying for too long at the read point of the door could be declared as parasite.

2.2.5 Unavailable communication

Unavailable communication between readers and tags would not enable to check aggregates and so would directly lead to denials of service.

This error could be detected by sticking an RFID tag near the read point in a way it should be in the same radio conditions than a user tag. A communication loss with the tag would indicate bad radio conditions.

2.2.6 Partial user localisation

Localisation of users could be a threat to their privacy. People could be directly observed or threatened. This would lead to a privacy leak failure. There is no way to detect this error.

2.2.7 Personal user data leak

If the system uses unprotected personal data, an attacker could retrieve theses data putting the user privacy in jeopardy. This would lead to a privacy failure. There is no way to detect this error. Hopefully, developed applications are not exposed to this issue as they do not involve any personal data.

2.3 Faults

Faults are inherent weaknesses of an application design that could make it behave in an unintended or unanticipated manner and might result in errors and failures. The cause could be an incorrect step, process, or data definition in a computer program. This section focuses on intentional human-made faults, another name for attacks, that could lead to the errors that were previously described.

2.3.1 RF media faults

- An attacker can prevent a tag from receiving waves from a reader by putting it inside a Faraday cage (reversible) thus making communication impossible.

- He could also destroy the tag (irreversible) or send high power HF noise.

Those faults may cause illegal appearance and disappearance errors. RF noise could also lead to communication errors with tags.

2.3.2 Physical weaknesses

- If tags can be unstuck without breaking, an attacker can physically move a tag from a fragment to another. As it is not possible to detect a tag move (no tag localisation available), it could lead a to tag swapping error.
- If it is possible to buy aggregated tags not attached to an object (for example, if it is possible to buy UbiPark tags on the Internet that can be put on a bike), there are more possible attacks. For instance, in UbiPark, an attacker can destroy tags of the bike he wants to steal and put on it the bought tags.
- As applications of pervasive computing, aggregate-based services gives free access to read points. Thus, any attacker could place parasite tags that could lead to a parasite tag error. Moreover, as tags are based on public IDs broadcasting, an attacker could get a basic localisation of a tag by detecting its presence in a read point mesh. This could help to a track a user and cause a "user located" error.
- Obviously other physical faults can be committed against specific applications. For instance, in UbiPark, an attacker could simply break the door to steal a bike. This example shows that it is often useful to add alternative protection (like video surveillance) to an aggregate-based system.

2.3.3 Data attacks

The following attacks require some specific hardware and knowledge in RFID. But, since RFID will be more and more used, anyone may have a tag interrogator installed on their mobile phones (for example) in a few years.

Using this tag interrogator, an attacker could:

- prevent access to tag data (password change, kill operation)
- alter data in a genuine aggregated tag. It would lead to a non genuine data error.
- write data in a new tag “from scratch” (without cloning). It could have the same consequences.
- clone a tag. It would lead to a non genuine tag error.
- link a user with tag identifiers by reading tag IDs and visually observing. This could help to track a user and cause a "user located" error.
- eavesdrop RF traffic or physically attack a chip (for instance proceeding a silicon die analysis or a power monitoring attack) in order to collect data. This can lead to two possible errors: the retrieval of private information and the use of non-genuine tag or data.

2.4 Summary

Figure 4 sums up the previously presented dependability threats of RFID aggregate-based system.

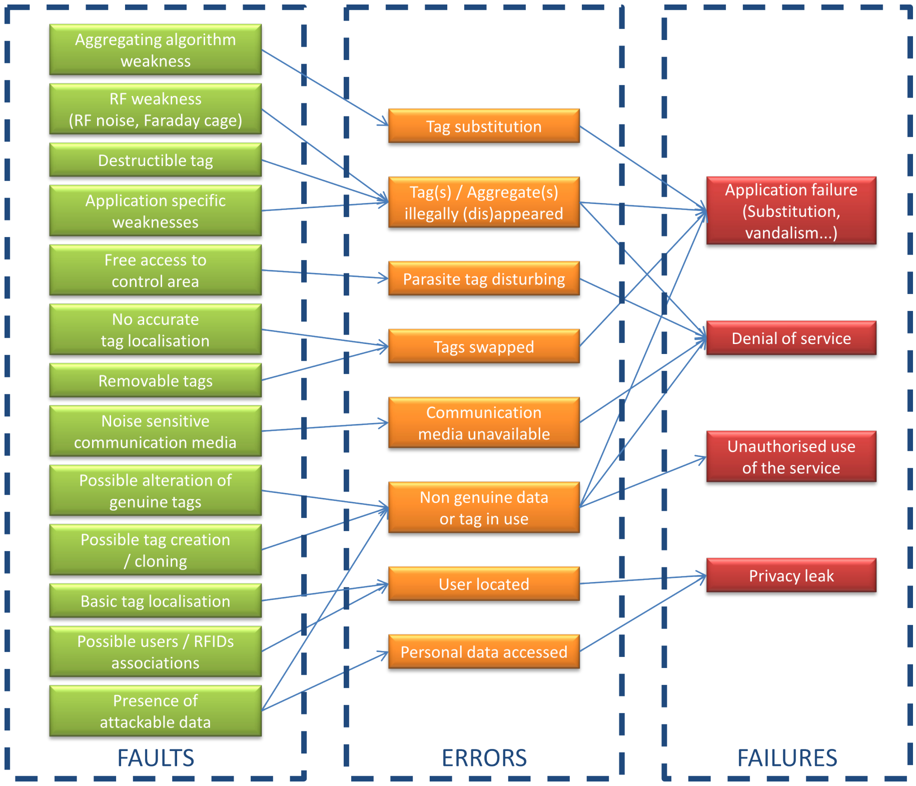
Aggregating algorithm weakness
RF weakness (RF noise, Faraday cage)
Destructible tag
Application specific weaknesses
Free access to control area
No accurate tag localisation
Removable tags
Noise sensitive communication media
Possible alteration of genuine tags
Possible tag creation / cloning
Basic tag localisation
Possible users / RFIDs associations
Presence of attackable data
FAULTS
Tag substitution
Tag(s) / Aggregate(s) illegally (dis)appeared
Parasite tag disturbing
Tags swapped
Communication media unavailable
Non genuine data or tag in use
User located
Personal data accessed
ERRORS
Application failure (Substitution, vandalism...)
Denial of service
Unauthorised use of the service
Privacy leak
FAILURES

Figure 4: Dependability threats of an RFID implementation of an aggregate-based system.

3 Solutions

Most of the previous faults and errors can be avoided using conventional countermeasures.

- Parasite tag errors could be detected, temporarily filtered and a technician could be asked to remove parasite tags or fragments.
- Tag swapping can be solved using destructible tags that would break and stop working if unstuck. However, this would not solve availability issues.
- Destructible tags faults are harder to solve: Tags should be hard to destroy but should still break if they are removed from their carrying object.

Fragment creation, cloning, alteration and data retrieving faults (section 2.3.3) require more complex solutions involving cryptographic means. The following section will focus on theses solutions. Implementations are proposed in section 4.

3.1 Keys and cryptosystems

3.1.1 Symmetric and Asymmetric cryptosystems

There are two main kind of cryptography: symmetric cryptography and asymmetric cryptography.

With a symmetric cryptosystem, a key is shared by all users[2]. For encryption cryptosystem, this key is used for both encryption and decryption. For dynamic authentication cryptosystem[3], the same key would be used by the user who wants to prove its identity and by the user who wants to verify this identity. For message authentication code (MAC) mechanism[4], the same key would be used to create and verify MACs when exchanging messages. This would lead the following issues:

- verifying a MAC (or play the role of the verifier in a dynamic authentication scheme) requires the shared key. Thus, all verifiers become able to create genuine entities.
- it is impossible to distinguish users of a symmetric authentication system (for example, given the same message, any user using the same key would issue the same MAC).
- if the key gets stolen or if one person gets corrupted (for example by distributing the key or issues pirate messages MACs), the key has to be updated for all the users and all previous encrypted or authenticated data become untrustworthy.

Therefore it is very important to ensure high protection of chips and computers which carry the shared key in their memory. Indeed if an attacker succeeds in

[2] Here, a user is a fragment or an aggregating/verifying system.

[3] A dynamic authentication cryptosystem enables a user to prove to another user that he knows a particular secret

[4] A MAC is a piece of information added to a data to authenticate it as a digital signature would . The main difference however is that anyone who can verify a MAC can also issue one because he also has knowledge of the secret key.

extracting the key from one of the users[2] (device theft, side channel attacks...), the security of the whole system collapses.

To reduce risks, keys can be changed often. However, even if it is possible to change keys of aggregates verifying systems, rewriting all tags can be sometimes really painful. A compromise would be to regularly update the encryption key and to maintain a list of trustworthy keys for decryption or authentication. This way, users using a revoked key could be ignored without disturbing other communications. The intrinsic drawback would be tags limited lifetime.

Asymmetric cryptography solves most of these problems as mentionned in [23]. On the other hand, it is more complex, needs more computing resources and requires higher data storage. With an asymmetric cryptosystem, each user generates a private key and a public key. The private key is kept secret whereas the public key is published. The private key enable its owner to decrypt or sign messages and dynamically prove to another user that he is the one related to a given public key. With the public key, any user can encrypt messages, verify signature or play the role of verifier in dynamic authentications. The private key is needed to decrypt or sign data and to play the role of prover in dynamic authentication.

With an asymmetric cryptosystem, if a user gets corrupted or gets his private key stolen, only his public key has to be revoked. If a private key shall be shared by a group of users (for example by all aggregating and verifying systems), there are fewer advantages of using an asymmetric cryptosystem. Thus, symmetric cryptosystems may be preferred for better performance and smaller memory footprint.

3.1.2 Digital certificates

A digital certificate enables to bind together a public key with the identity of a user. In particular, it contains :

- the user's description (for example an email address),
- the public key of the user's key pair (it can be used for exemple to send cipher text to the owner),
- the expiration date of the certificate,
- a signature of the previous data issued by a CA (or by another user).

Standard X.509 certificates are signed by a Certification Authority (CA) which ensures the validity of the certificate (the fact the owner of the certificate corresponds to the given description). Certificates can also be self-signed. It is the case for CA's certificates.

The CA can revoke any certificate it delivered if it becomes corrupted (owner's description does not match with real users) by publishing its corrupted public key in a revocation list.

Certificates may be hierarchical: a CA signs several certificates for users which can sign other certificates, etc. So the system is very flexible. If the behaviour of a user, his CA or the user who signed his certificate is becoming suspect, his certificate will not be trusted anymore. A big advantage of this solution is that it will not be necessary to rewrite all certificates if one user turns out to be corrupted.

Obviously, the CA shall never be compromised, otherwise all certificates would become unusable.

3.1.3 Key storage and shared key

Tag memory is often very limited. Storing a certificate (corresponding to a tag's signature for example) can be problematic. In most cases, the following solution can be used: all used public keys are stored in each aggregating/verifying system and a short identifier is assigned to each public key. Tags memory would store only their identifier and their private key (which access should be denied).

But this solution is less flexible than certificate: in particular it forces each verifying system to have a database of all public keys and their associated short identifier. If each tag shall have a different public key (or private key for symmetric cryptosystem), the memory a verifying system would need may be huge. In this case, certificates (necessarily with asymmetric cryptosystem) may be stored in the tag.

This idea is also useful when symmetric encryption is used for example: the identifier of the key used by encryption algorithm is saved (as a plain text). It makes easier changing shared key.

3.2 Uncloneable tags and authentication

Cloning a tag (and so a fragment) is one of the most critical issue of an aggregate-based system as it enables the attacker to substitute objects or to use an unauthorized service.

If tags contain only memory, cloning a tag is really easy: the attacker just needs to have a writeable tag and to copy data from the original tag to the new one. Even if manufacturers do not allow to write some memory banks (as it is the case with most of the commercial tags), it is possible to emulate a tag using appropriate hardware.

C1G2 tags enable password authentication of readers: it should prevent an attacker from directly accessing a tag's memory. However, the password can be easily eavesdropped in communications as the standard do not require tags to use a secure protocol.

Actually even tag authentication with a more complex mechanism (for example zero-knowledge proof) is insufficient as soon as the secret (used by authentication) is shared by all tags. Using a real genuine tag and a tag emulator, an attacker can make any tag (including illegal clones) look like genuine:

- if authentication is requested, the tag emulator uses the genuine tag to correctly answer
- if normal data read is performed, the tag emulator sends data of the tags to be cloned

This kind of attacks is called a **man-in-the-middle attack** (see figure 5). Hence we propose several solutions:

- Randomized data encryption between tags and readers using random data provided by the reader (see section 3.2.1).
- The tag contains a secret key directly link to its ID and prove it knows it to the reader without revealing it (see section 3.2.2).

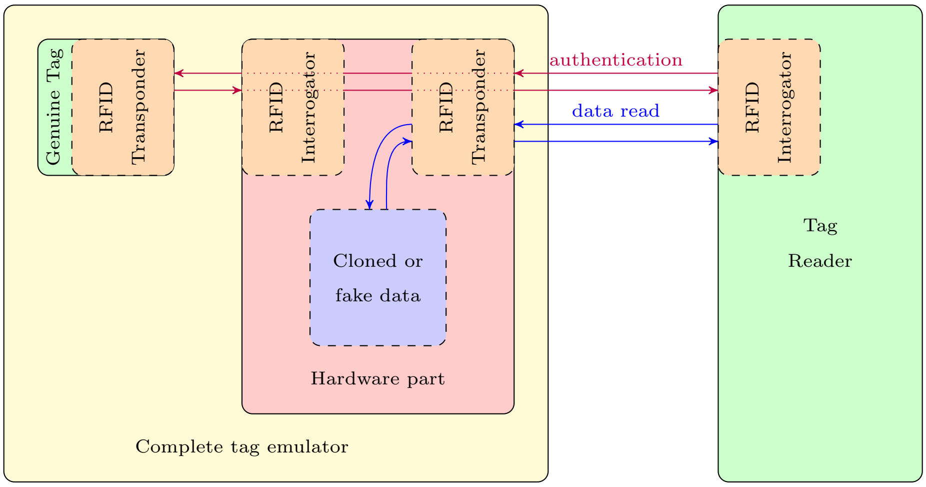

Figure 5: Tag simulator for cloning any tag with authentication support

- The tag contains a secret key directly link to its data thanks to an identity-based cryptography scheme and prove it knows it to the reader without revealing it (with a zero-knowledge proof for example).
- The tag uses a Physical Uncloneable Function (PUF) (see sections 3.2.3 and 3.2.4).

With the second method, tag is not really uncloneable, just a part of the tag is uncloneable: the ID, but this is often sufficient (see remark 3.4.1). We will say tag has an uncloneable ID or unique ID.

Remark 3.2.1. *Shared secret methods can be used if the required security level is not very high: password authentication (section 3.6) for example.*

The following section only contains advanced solutions for a very high security level.

3.2.1 Randomized encryption

If the communications between a reader and a specific tag are always the same, the latter can be easily cloned, even if all the data is encrypted and incomprehensible. The attacker would just need to eavesdrop the communications and make a device that replays the original tag's answers.

To face this, data to be sent from tags to readers can be ciphered with added random data chosen by the reader. Hence, if readers choose a nonce, each time a reader requests tag data, transmitted answers will be necessarily different. The random number must not be chosen by the tags because a tag emulator could always choose the same (the number the original tag used during the eavesdrop).

TLS ([10]) and SSH protocol version 2 ([21]) are two widely used protocols which use this idea: the server corresponds to the tag and the client corresponds to the tag interrogator. Notice these two protocols also provide tag authentication.

3.2.2 Unique ID with zero-knowledge proof

Previously presented solutions require either the sharing of a certificate or private key between tags, either the registration of all tag's public keys into all interrogators. This may not be convenient. In [5, 7, 24], there is a solution which does not need all tags to share the same secret. Each tag has its own private / public key-pair enabling zero-knowledge proof[5]of identity (or just a signing algorithm like DSA). The public key is the ID of the tag whereas the private key is stored in the tag such that only its microprocessor can read the key (more details can be found in section 3.2.4).

The tag can prove its ID is authentic by proving it knows the corresponding private key without revealing it.

This solution has many advantages over the previous one:

- there is no shared key common to all tags,
- the protocol between tag and interrogator can be a standard protocol with an additional command which enables to prove the authenticity of tags,
- authenticity verification can be performed only when high level of security is needed.

However there are also some disadvantages:

- ID cannot be chosen (otherwise there is not protection !),
- there is no authentication of the fragment's provider: any provider can create such tags contrary to previous method,
- only ID (public key) is protected.

The two last issues can be solved by adding a signature (or a Message Authentication Code) to the data (ID included) of the tag (see section 3.4.1).

3.2.3 Physical Uncloneable Function (PUF)

According to [9], a Physical Uncloneable Function (PUF) is a function:

- that is based on a physical system (common PUFs are embodied in electronic chips),
- that is easy to evaluate (using the physical system),
- which plot looks like a random function,
- that is unpredictable even for an attacker with physical access to the component.

[5] There are two kinds of zero-knowledge proofs: honest verifier zero-knowledge proof (like Schnorr one [22]) and general zero-knowledge proof (like Okamoto one [20]). With the first kind, an attacker who eavesdrops communication between a genuine tag and a genuine tag interrogator cannot learn any information about the private key (except information he can directly computes from the public key). With the second kind, an attacker who can make requests to the tag, cannot get any information about the private key. So general zero-knowledge proof shall be preferred when a high level of security is required.

PUFs can be tiny electrical circuit exploiting unavoidable IC fabrication process variations (for example path delays) to generate secrets.

First part of [6] is an example of use of PUF f for authenticating each tag. A more general idea could be to save a lot of $(c, f(c))$ pairs for all tags (where c is a random entry of the PUF) in each tag interrogator. Then a tag interrogator ask a tag to give the output of its PUF corresponding to some randomly chosen inputs c. Output of a PUF may depend a bit on external condition (like temperature), but this issue can be solved by accepting some error bits in the answer of the tag.

Unfortunately, $(c, f(c))$ pairs should be used only once, else an attacker could use recorded answers. Thus, tag readers should know all recorded challenges of each tag. This may represent a huge amount of data and would need a connection to a challenge database, meaning static or online applications. Moreover, the database could be attacked, enabling the pirate to know all used PUF challenges and emulate genuine tags without needing to physically clone a PUF.

Nowadays, the only known implementation of this PUF secured tag technology is the Vera X512H developped by *Verayo* ([25]).

3.2.4 Storing cryptographic secrets, physical attacks and PUF

Some of the previously presented cryptographic solutions require tags to store shared or private secrets. Each secret should be readable only by the tag's microprocessor for cryptographic purposes. If a very high level of security is required, it is not recommended to use memory for storing the secrets because a physical analysis of the tag's chip can enable an attacker to retrieve it.

Fortunately, PUFs can provide a solution. Indeed opening a chip with a PUF will almost always change the PUF behavior. It is difficult to use directly a PUF because output of a PUF can depend a bit on external conditions, but there are ways to solve this problem. For example, in [24], the authors present a tag authentication scheme using a signed private key issued from a PUF. It uses a helper data and a special function which takes the helper data and the response of the PUF to compute the private key. The helper data normally leak very few bits and can be stored in a normal memory. This way, the private key can be dynamically rebuilt, which avoids its storage.

3.3 Memory write protection

As seen in section 2, if write or kill operations [6] are not locked or disabled, an attacker can easily make the system unavailable. To avoid this problem while enabling authorized users to modify aggregates, a possible solution is to have **reader authentication** (not tag authentication as in the previous section). In section 3.6, some advices on ways to use simple password authentication are given. A better method (if high level of security is required) is to use a symmetric or asymmetric authentication scheme as those described in section 3.2.1.

However two points shall not be forgotten:

- Man-in-the-middle attacks has to be (almost) impossible (see section 3.2). A genuine tag interrogator must not be helpfull for an attacker device to pass the authentication in order to write data into tags.

[6] A lot of tags provide a “kill” feature which enables a tag interrogator to kill the tag, i.e. to permanently disable the tag.

- Most of the time, tags cannot embed a public key database of all authorized tag interrogators nor verify any certificate expiration's date (passive tags cannot embed a clock as they have no stable power supply). Hence reader's authentication is quite complex. More information can be found in the article [19].

3.4 Aggregating company authentication

If a high level of security is required, one of the presented solutions should be implemented to avoid cloning. However, instead of cloning tags, an attacker could try to build pirate tags from scratch or to modify genuine tags. This section will focus on methods aiming at proving the authenticity of the aggregating data carried by the tags. This way, only aggregated objects issued from an authorized provider will be taken into consideration.

3.4.1 Fragment authentication

Fragment authentication enables an aggregating company[7] to prevent unauthorized aggregating systems from creating compatible aggregated fragments or aggregated objects (related to no aggregated object but looking like a part of an aggregated object). Notice that a corrupted fragment can be used as a parasite tag.

The authentication can be dynamic or static.

Static authentication only uses public tag memory: a small amount of data is added at the end of the aggregating data which proves aggregation was done by an authorized aggregating system. If the used cryptosystem is symmetric, these extra-data are called a MAC (Message Authenticatio On the one hand, a signature mechanism enables to know which aggregating system created an aggregated object. If the latter behaves dishonestly, its public key (see section 3.1) can be revoked. On the other hand, MAC algorithms are generally significantly faster and produce a lot shorter message authentication data (regarding memory space used in the tag) than signature cryptosystems. In addition, MAC algorithms often use either cryptographic hash functions or symmetric block cipher which could be used by other parts of aggregating and verifying systems (hash functions are often used in aggregating and verifying algorithms). This could significantly speed up the system and would free up tag memory. Section 3.4.3 deals with theses perspectives.

Dynamic authentication could also be possible, but it would only attest that the tag to be authenticated knows a secret (so it should be issued from the right company). However, it would not guarantee that the aggregating data have never been modified and is very costly.

Remark 3.4.1. *The MAC/signature of cloned data remains the same as the MAC/signature of original data. So, authenticating only aggregating data does not prevent from fragment cloning. The only way to avoid it is to add uncloneable data in the input of the MAC/signing function. If tags have an unique ID (see section 3.2.2), signature or MAC makes tag indirectly totally uncloneable.*

[7] An aggregating company is a company which is allowed to deliver aggregates. Nowadays, there is only one aggregating company: SenseYou (`http://www.senseyou.fr`)

Authentication and aggregating digest size

Tag aggregation is based on data hashing. Collision resistance of the hashing function should be high enough so they will be few chances to find an object that can be swapped with an other without digitally affecting the integrity of an aggregate it is part of. Moreover, without additional security mechanisms, it is necessary to ensure that hashing functions are preimage and second preimage resistant to avoid preimage attacks.

One benefit of authentication mechanisms is that it indirectly enforces security of the aggregating system without requiring theses properties. Indeed, an attacker cannot swap a tag with a one with an other ID nor change an aggregating digest by another without corrupting the MAC/signature. So using a tag or aggregated object authentication enables to reduce the size of digest (without reducing the security level) and enable using the system without locking write operations (if an attacker changes the content of a tag, the signature will no longer be valid). With authentication, the digest size is only determined by the required probability of collisions.

Remark 3.4.2. *If a second preimage resistant hash function is used for building aggregating data, the authenticity of a fragment or aggregate can be verified only when a high security level is required*[8]*.*

3.4.2 Aggregated object authentication

Instead of authenticating each fragment, it is also possible to authenticate only complete aggregated objects. On the one hand, it may use less tag memory to store its signature because it can be spread over multiple tags, on the other hand, an attacker can create fake tags and disturb the system (it is not possible to reject unauthentic fragments are they are not signed) causing the inauthenticity of the complete aggregated object.

3.4.3 Using MAC algorithm instead of hash function

There are another complementary way to use MAC: the hash function (used by aggregating or verifying algorithms) can be replaced by a MAC algorithm. In this case, the private key must be shared by all the aggregating/verifying systems of a same service.

There are two main advantages. First, only a genuine aggregating system can create aggregated objects[9]. Then, adding or replacing a tag in a read-only aggregated object becomes a lot more difficult. Indeed, with a perfectly safe hash function (it may not exist but let suppose currently used hash functions have this intuitive property) with n bits output, finding a second preimage needs to try about 2^n different inputs. If n is big enough, this computation is very costly but can be performed on any computer without any access to a verifying system. But, if a perfectly secure MAC algorithm with n bits output is used instead of an hash function and if the key cannot be recovered, trying

[8] There may have several uses of the same aggregated object which may require different security levels.

[9] This property is obtained by almost all solutions of the section 3. However using a MAC algorithm instead of a hash function would be significantly less resource consuming.

2^n different inputs require to do 2^n (or 2^{n-1} in mean) requests[10] to a genuine verifying system. So if a verifying system does not accept more than 1 request per second (for instance), a brute force attack against an aggregated object which uses a MAC algorithm needs at least about 2^n seconds whereas such an attack against an aggregated object which uses an hash function requires only 2^n computations of the hash function (and each computation may take only a few milliseconds — furthermore these computations may be distributed on a huge number of computers).

Using a MAC enables to reduce the size of the aggregation data (without reducing the security level).

Remark 3.4.3. *It is also possible to use a signature (of the digest) instead of the digest (or the MAC) itself. Each aggregating system could have a different private key and all verifying systems could check aggregates using corresponding public keys. Even if this solution is more flexible (than MAC) for key management, it has two disadvantages. Signature size is often much bigger than MAC size and signature verification is slower than MAC computation. Unfortunately aggregating systems often need to store two digests (or signature in this case) and verifying system could require to achieve a lot of computations.*

3.5 Encryption

Encryption of the tag data avoids unauthorized readers to parse data of tags and brings so the following advantages:

- only authorized readers can create aggregated objects.
- an unauthorized reader cannot say if objects are aggregated or not (privacy feature).
- a company can prevent other companies to sell compatible aggregating or verifying systems[11].

The first point can be performed by a company authentication (see 3.4), but symmetric encryption is often a lot faster than signature (but not than MAC).

Warning 3.5.1. *Generally encryption does not provide authentication. An attacker can make a fake tag with random data (instead of encrypted data) and he can so disturb the system (the tag is seen as a part of a aggregated object by the verifying system although it is just a fake tag).*

Asymmetric or symmetric encryption algorithms can be used. However it does not seem very useful to use asymmetric algorithm because the private key (used for decryption) shall be shared by all RFID readers anyway and asymmetric encryption algorithms are often slower than symmetric ones (i.e.

[10] Request is a very generic term but it cannot be easily specified. Actually, the issue is to prevent the attacker from verifying more than one MAC in a given unit of time. MAC verification cannot be normally directly performed but they often can easily be indirectly performed (for example by putting tags with special data near the tag interrogator and by watching its reactions).

[11] It is not only a matter of technological monopoly, it is really important regarding the security. For instance, another company could interfere with one of the proposed services, or would not fully implement all security mechanisms.

they need more computing resources) and cipher text are often longer than plain text (for example, for El Gamal encryption algorithm, cipher text size is twice plain text size).

If signature (see section 3.4) is also required, signcryption can be a good alternative to symmetric encryption and asymmetric encryption. Signcryption is a cryptographic primitive which simultaneously sign and encrypt (in an asymmetric way) a plain text.

But separated symmetric encryption and signature have the following advantage: a cheap verifying system can only decrypt the tag without verifying signature whereas a state of the art one can decrypt tag data and verify signature.

If MAC (see section 3.4) is also required, authenticated encryption can be used. Authenticated encryption is a cryptographic primitive which simultaneously performs a MAC and encrypts a plain text. There is often only one private key for these two operations. Authenticated encryption is something like a symmetric signcryption.

When neither signcryption nor authenticated encryption is chosen, there is another choice to do: whether the tag is first signed (or authenticated by a MAC) then encrypted or if the tag is first encrypted and then signed (or authenticated by a MAC — signature or MAC is not encrypted). The second solution brings two advantages: it needs to encrypt a smaller amount of data and signature can be verified without decrypting data. The first solution hides the signature which may be useful. In particular, it prevents an attacker who knows the signature public keys (used by each aggregating system) but not the encryption private one from knowing which system created the aggregated object.

3.6 Use of password

Passwords are the simplest way to do an authentication. But, as explained in section 3.2, plaintext passwords can be eavesdropped. If an attacker manages to get a password, he can do the same things as a genuine reader.

So, here are some basic rules that should be applied:

- reduce the number of times a password is sent over the air,
- when encryption is supported, send the ciphertext of the password and a nonce ciphered together,
- password memory (write or read) lock should be used only when permanent lock cannot be used (when a tag shall be used multiple times),
- passwords should not be the same for all tags.

In order not to use the same password for each tag, there are (at least) two possibilities:

- store the password in a secured tag (with real authentication and encryption). Most aggregate-based applications enable using secure personal badges (sometimes from another service).
- the password of each tag is a MAC of its ID. The key of this MAC shall be different from the keys of the potential other MACs of the aggregate-based application.

The second possibility enables to do a really simple authentication to the tag and, if eavesdropping is impossible, it prevents from cloning tags. Indeed an attacker does not have access to the password and so cannot copy the tag.

In addition, password authentications should only be proceeded in restricted areas where there must be no eavesdropper.

4 State of the art

The section 3 provided only theoretical solutions without any detailed implementation (except for tag authentication and unclonability). This section focuses on a possible implementation an aggregate-based system using currently available low cost tags: Alien Higgs 3 tags.

4.1 Higgs 3 based implementation

The proposed implementation aims to provide a security strength of 80 bits (that means about 2^{80} operations are needed to break cryptographic primitives). Only NIST approved primitives are used: HMAC, DSA and AES-CFB. Recommended security strength given in this section comes from this NIST document: [18].

According to the previous sections, two main points shall be verified by tags (except if security has strictly no importance):

- tags shall be attached to physical objects..
- tags shall not be cloneable.

The first point can be provided using destructible tags: when an attacker tries to remove such a tag, it gets destroyed (see [14]). It seems possible to make destructible any RFID inlay. Another solution is to use very rugged tags and very good glue.

Concerning the second point, not all tags provide a real protection against cloning: for most tags, the protection is only based on the fact that a part of the memory can not be written by a reader; this is not sufficient, see section 3.2 for more information. The Alien Higgs 3 tags provide a TID unclonability feature. This is one reason why it has been chosen.

4.1.1 Alien Technology® Higgs 3 tags

Alien Technology® Higgs 3 tags are UHF EPC class 1 generation 2 (also called C1G2 tags — standard [13]) tags with some extra-features.

Memory banks All C1G2 tags have four memory banks:

- EPC: (RW) 96 bits for Higgs3 tags + some extra-bits for CRC.
- reserved: (private) where passwords are stored
- TID: (RO+auth) 96 factory programmed bits, of which 64 bits are called UID (unique identifier). Alien ensures each produced Higgs 3 tags has a different UID.
- User: (RW) 512 bits for Higgs3 tags.

According to EPC C1G2 standard, EPC ID shall be either as defined in ISO/IEC 15961 or as defined in the EPC Tag Data Standard ([12]). Anyway, in this implementation, only User memory is used: EPC ID is not modified (it is the one programmed by Alien).

Unique ID Higgs 3 tags have a very interesting feature: dynamic authentication. Dynamic authentication enables an Alien interrogator to recognize genuine Alien tags. For more information see: [1]. This mechanism ensures the UID is really unique, as soon as dynamic authentication is not flawed. Unfortunately, we have not had access to the specifications of dynamic authentication and we cannot see how it works exactly.

Kill and access password Higgs 3 tags, like all C1G2 tags can be killed, that means can be completely disabled such that tags do not respond anymore to a tag interrogator. Killing a tag requires knowing the kill password stored in the tag. If the kill password is 0, the tag cannot be killed (or more precisely, the kill password must first be changed).

The kill password can also be used (in some tags) to recommission a tag (see standard for more information).

The access password is another password stored in the tag. It can be used to enter the secured state. Some features of the tags are only available when the tag is in the secured state. By default, the tag is in the open state.

Lock When a tag is transitioned to the secured state (using the appropriate password) it is possible to lock its memory banks or password(s). Locking a memory bank prevents users from writing on it until they transition the tag to the secured state. Locking a password prevents users from reading and writing this password until they transition the tag to the secured state.

A tag interrogator can also request a tag in secure state to permanently write the lock status of a password or memory bank. If the lock status is “lock”, it is said the interrogator “perma-locks”the password or the memory bank, otherwise it“perma-unlocks” it.

Alien Higgs 3 tags can also perma-lock blocks of memory (instead of complete memory bank).

They also provide read locking features. Each block (64 bits) of user memory can be read locked. The read lock is effective in open state. If the tag is transitioned to secured state, memory blocks can still be read.

Important recommendation In aggregate-based application, availability is often important and so it is highly recommended to perma-lock the kill password to 0. This prevents an attacker from killing or recommissioning tags.

4.1.2 Implementation

The used aggregating format uses two 80 bits data fields to store aggregating data.

As proposed in section 3.4.3, the hash function is replaced by the HMAC-SHA-1-80 MAC algorithm (80 first bits of HMAC with SHA-1). The key length might be 160 bits or more.

Only the user memory is used. Its organization is depicted in figure 6.

The header is not encrypted.

The ciphertext part is encrypted by AES-CFB (This ensures protections against attacks which consists in XORing last part of ciphertext to create another ciphertext whose corresponding plain text has some bits inverted). The initialization vector is depicted in figure 7.

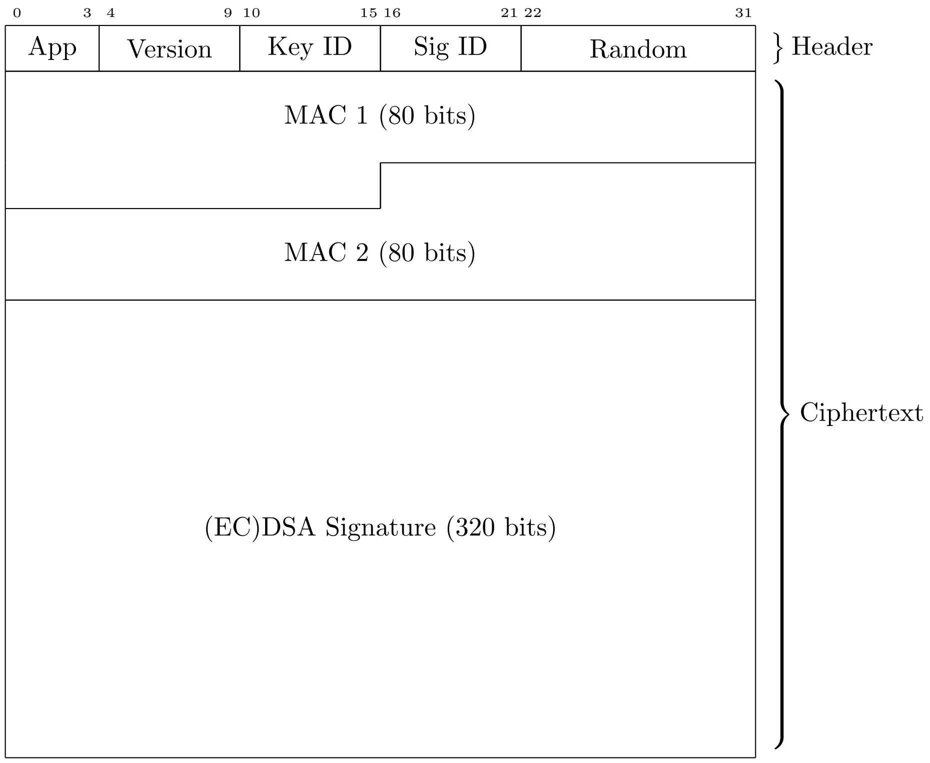

Figure 6: User data memory of the Higgs 3 implementation

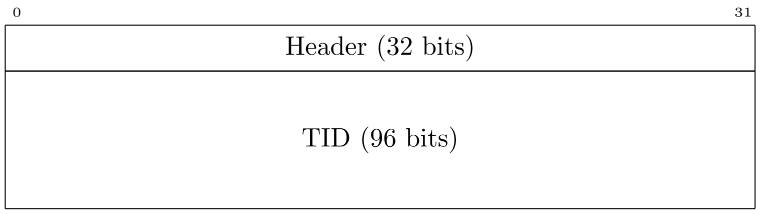

Figure 7: Initialization Vector

The random number in the header (see figure 6) should be randomly chosen each time the tag is written. It enables to avoid that the same initialization vector is used twice (but due to birthday paradox, it works approximatively only when the tag is written less than $2^5 = 32$ times).

Tag data are signed thanks to a DSA (or ECDSA[12]) signature of 320 bits. More precisely, data of figure 8 are signed.

Each aggregating controller should have its own private key to sign tags. Each verifying controller shall have all public keys corresponding to these private keys. If an aggregating controller gets corrupt, its public key is removed from verifying controllers.

There are many ways to deal with access password. It can be:

[12] ECDSA and DSA are very similar. Depending on the hardware, one may be faster than the other.

TID (96 bits)
EPC (96 bits)
User Data (192 first bits)

Figure 8: Signed data

- a MAC of the whole data of the tags (see section 3.6). In this case, an authentication can be made thanks to this tag specific password. However, it prevents from locking read operation in open-mode.
- a common password. In this case, a read lock can be performed.
- a password stored in the user badge. It supposes the user badges is a cryptographic tag. In this case, a read lock can be performed.

Notice that none of theses propositions faces the password eavesdropping problem since the password is regularly sent over the air. If data are not perma-locked but just locked, it enables an attacker to easily write random data in the tag and cause the system unavailability. In addition, risks are higher when passwords are shared: if an attacker gets the password from a tag, he can modify all the tags sharing it.

4.2 Discussion

4.2.1 Advantages and drawbacks of UHF C1G2 tags

UHF C1G2 tags are very common and inexpensive tags. Their read distance may be up to 10 meters when the read distance of most of other passive tags is often less than 1.5 meters.

Furthermore, according to [2]: current UHF tags can be used near liquids or metal.

But UHF tags features are very limited. No UHF tags have classical cryptographic primitives like RSA or AES, contrary to some HF tags. However some UHF tags have new or private cryptographic primitives (whose security level is often very low — see article [8]), like Higgs 3 tag for dynamic authentication or SecureRF Lime tag. The SecureRF Lime tags are battery-assisted C1G2 tags which provide authentication and encryption. According to their description, they seem uncloneable. They use the Algebraic Eraser™ to perform asymmetric authentication. This algorithm is described in a published article [3] but the used parameters are proprietary and are not known. There are attacks on this algorithm: [17] and [15] but it is not sure these attacks work with SecureRF parameters.

4.2.2 EPC ID

The GID-96 EPC standard proposed by GS1 do not require all the EPC to be unique but prevents from collision issues with tags from another company (each company has a different “General Manager Number” — this number is attributed by GS1).

Here is a structure that would not fully respect one of the EPC encodings but it has a GID-96 header, enables to identify the company the tag is from, and leaves 60 bits free:

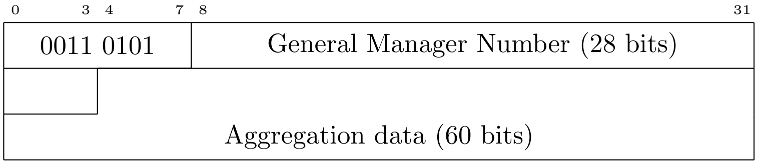

Figure 9: EPC data memory

Remark 4.2.1. *The free bits from the EPC memory bank could be used to store a part of the first MAC to detect potential aggregated object without reading the User bank. It can be advantageous for systems offering better performance at reading EPC bank data.*

4.2.3 Discussion

The Higgs 3 implementation has been chosen because of its flexibility: authentication and signature verifications are not mandatory. Some tags interrogator, in secure areas for example, can run theses tests, others might not implement them.

The proposed implementation seems quite very secured, if Alien dynamic authentication is secured (which is not proved. . .).

If there were C1G2 tags with a real unique ID feature (such as described in section 3.2.2), the Higgs 3 solution (applied to these tags) may be sufficient for almost all applications.

Moreover, most aggregate-based applications require each user to have a badge (for instance in UbiPark). This badge might be a very secured HF badge (relying on widely used cryptographic primitives) to ensure a higher level of security.

The main issue is maybe privacy: tags broadcast their unique ID as soon as they receive enough power to do it, so they are easy to track. A real authentication of the tag and a randomized encryption as described in section 3.2.1 is a partial solution. But according to [4], privacy (and non traceability) is a very difficult problem with RFID tags.

Contents

References

[1] Alien Technology®. *Alien Technology® Bolsters Higgs-3™ RFID IC Security with "Dynamic Authentication"*. Sept. 2009. URL: http://www.alientechnology.com/newsevents/2009/press091609.php.

[2] Alien Technology®. *Pharmaceutical Shifts Towards UHF RFID for Savings.* White paper, Alien. URL: http://www.alientechnology.com/docs/WP_UHF_RFIDPharmaceutical.pdf.

[3] I. Anshel et al. "Key agreement, the Algebraic Eraser™, and lightweight cryptography". In: *Algebraic methods in cryptography: AMS/DMV Joint International Meeting, June 16-19, 2005, Mainz, Germany: International Workshop on Algebraic Methods in Cryptography, November 17-18, 2005, Bochum, Germany.* Vol. 418. American Mathematical Society. 2006, p. 1.

[4] G. Avoine and P. Oechslin. "RFID traceability: A multilayer problem". In: *Financial Cryptography and Data Security* (2005), pp. 125–140.

[5] L. Batina et al. "An elliptic curve processor suitable for RFID-tags". In: *Int. Assoc. for Cryptologic Research ePrint Archive* (2006).

[6] Leonid Bolotnyy and Gabriel Robins. "Physically Unclonable Function-Based Security and Privacy in RFID Systems". In: *Pervasive Computing and Communications, 2007. PerCom '07. Fifth Annual IEEE International Conference on.* 2007, pp. 211 –220. DOI: 10.1109/PERCOM.2007.26.

[7] M. Braun, E. Hess, and B. Meyer. "Using elliptic curves on rfid tags". In: *IJCSNS* 8.2 (2008), p. 1.

[8] Nicolas T. Courtois. *The Dark Side of Security by Obscurity and Cloning MiFare Classic Rail and Building Passes Anywhere, Anytime.* Cryptology ePrint Archive, Report 2009/137. 2009. URL: http://eprint.iacr.org/.

[9] Srini Devadas. *Physical Unclonable Functions and Applications.* URL: http://people.csail.mit.edu/rudolph/Teaching/Lectures/Security/Lecture-Security-PUFs-2.pdf.

[10] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2.* RFC 5246 (Proposed Standard). Updated by RFCs 5746, 5878. Internet Engineering Task Force, Aug. 2008. URL: http://www.ietf.org/rfc/rfc5246.txt.

[11] J.C. Laprie (ed). *Depdendability Basic concepts and terminology.* Springer-Verlag Wien New York, 1991.

[12] EPCGlobal™. *EPC Tag Data Standard (TDS) v. 1.4.* June 2008. URL: http://www.epcglobalinc.org/standards/tds/tds_1_4-standard-20080611.pdf.

[13] EPCGlobal™. *UHF Class 1 Gen 2 Standard v. 1.2.* May 2008. URL: http://www.epcglobalinc.org/standards/uhfc1g2.

[14] Hanser. *Metalcraft introduces destructible option for RFID Windshield Tag.* URL: http://www.hanser.com/2009/news/metalcraft-introduces-destructible-option-for-rfid-windshield-tag.

[15] A. Kalka, M. Teicher, and B. Tsaban. "Cryptanalysis of the Algebraic Eraser and short expressions of permutations as products". In: *Arxiv preprint arXiv:0804.0629* (2008).

[16] Paul Couderc Michel Banâtre Fabien Allard. “A spatial computing approach for integrity checking of objects groups”. In: (2010).

[17] A.D. Myasnikov and A. Ushakov. “Cryptanalysis of the Anshel-Anshel-Goldfeld-Lemieux key agreement protocol”. In: *Groups-Complexity-Cryptology* 1.1 (2009), pp. 63–75.

[18] NIST. *Recommendation for Key Management.* Mar. 2007. URL: `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf`.

[19] R. Nithyanand, G. Tsudik, and E. Uzun. *Readers behaving badly: Reader revocation in PKI-based RFID systems.* Tech. rep. Cryptology ePrint Archive, Report 465, 2009.

[20] T. Okamoto. “Provably secure and practical identification schemes and corresponding signature schemes”. In: *Advances in Cryptology—CRYPTO’92.* Springer. 1993, pp. 31–53.

[21] RFC. *RFC 4253 - The Secure Shell (SSH) Transport Layer Protocol.* URL: `http://www.ietf.org/rfc/rfc4253.txt`.

[22] C.P. Schnorr. “Efficient signature generation by smart cards”. In: *Journal of cryptology* 4.3 (1991), pp. 161–174.

[23] Douglas R. Stinson. *Cryptography: Theory and Practice.* CRC-Press, 1995.

[24] P. Tuyls and L. Batina. “RFID-tags for Anti-Counterfeiting”. In: *Topics in Cryptology–CT-RSA 2006* (2006), pp. 115–131.

[25] Verayo. *Verayo PUF RFID.* URL: `http://www.verayo.com/product/pufrfid.html`.

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399