Dependability of Aggregated Objects, a pervasive integrity checking architecture

Fabien Allard
SenseYou
RENNES, France

Michel Banâtre, Fabrice Ben Hamouda-Guichoux, Paul Couderc, Jean-François Verdonck
INRIA Rennes Bretagne Atlantique
RENNES, France

***Abstract*—RFID-enabled security solutions are becoming ubiquitous; for example in access control and tracking applications. Well known solutions typically use one tag per physical object architecture to track or control, and a central database of these objects. This architecture often requires a communication infrastructure between RFID readers and the database information system. Aggregated objects is a different approach presented in this paper, where a group of physical objects use a set of RFID tags to implement a self-contained security solution. This distributed approach offers original advantages, in particular autonomous operation without an infrastructure support, and enhanced security.**

***Keywords*-Ambient computing; RFID; security.**

I. 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...

However, these solutions typically use a single RFID tag on the physical object or person that is to be controlled or protected. Unfortunately, RFID tags could face various security issues. However, RFID tags are highly exposed to various attacks which could compromise the service. In this paper, we discuss an approach using a collection of tags distributed over a set physical of objects forming a logical group. As we will see, this approach can provide enhanced security in specific context, as well as other interesting properties.

The rest of the paper is organized as follows: in the next sections, we introduce the notion of aggregated objects. The third section discusses the advantages and potential vulnerabilities. The fourth section addresses some solutions. Finally, the fifth section discusses related works and concludes.

II. Aggregated objects and basic concepts

A. 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.

B. Example of applications

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) Ubi-Check: Ubi-Check [1] 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.

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.

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.

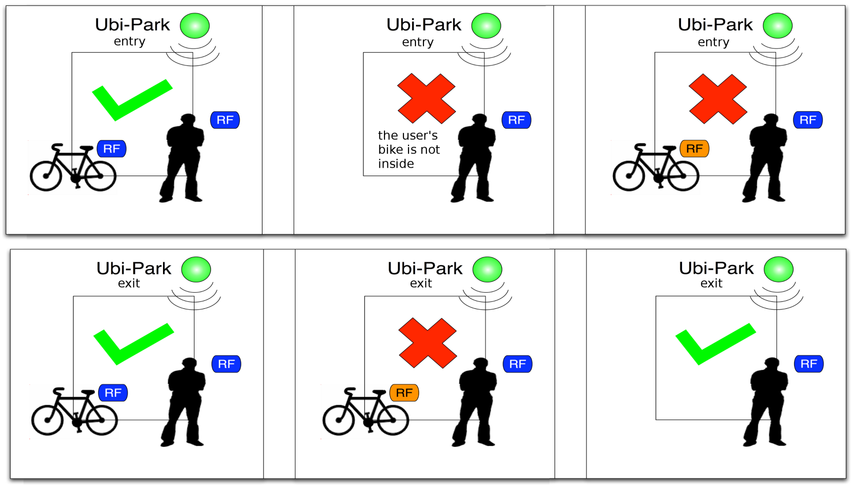

Figure 2. Ubi-Park, entrance and exit of a bike shed.

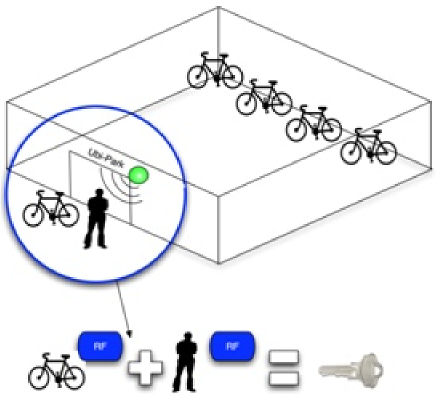

Figure 1. Ubi-Park

III. Dependability properties

This section discusses the properties of coupled objects systems with respect to dependability threats. 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. More details can be found in [2].

As any RFID system, coupled objects are exposed to various vulnerabilities. However, as we will see, the distributed nature of coupled objects help to mitigate these issues. We 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.

A. 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.

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) 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.

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 to perform aggregates checking, so aggregate-based systems limit the exposure of private information and the possibility of deriving users profile. Still, it is possible to identify the tags IDs corresponding to a specific user and start tracking his tags if the IDs are not regenerated regularly. More information about privacy threats can be found in [3]. Moreover, if aggregating data are not encrypted, it may be possible for an attacker to find all the fragments of an aggregate. 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. In Ubipark, this would enable to find somebody's bike thanks to the user badge.

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.

B. 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.

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) 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. However, aggregated objects can use a multiplicity of tags, potentially hidden in various parts of the group of objects to protect. An attacker would need to know the location of all the tags to avoid an inconsistency detection.

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 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 IV-B and IV-D).

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.

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.

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. This issue can be mitigated by regularly regenerating the IDs used to identify the fragments.

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.

C. 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.

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) 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 monitoring) to an aggregate-based system.

However, it should be noted that aggregate-based systems can use a multiplicity of tags, reducing the risk of a successful attack, because all the tags would have to be compromized in order to avoid an inconsistency to be detected. For example, in the case of UbiPark with multiple tags embedded inside the tires, under the seatpost or other parts of the bike, an attacker would have to find the location and access all the tags physically.

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.

IV. 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.

The structure of aggregated objects, using a multiplicity of tags, make them less vulnerable to Fragment creation, cloning, alteration and data retrieving faults (Section III-C3) than single tag systems : to be successful, an attacker would have to find and compromize all the tags scattered and potentially hidden inside the various objects of the group. Although more difficult in practice, these attacks are still possible and requires more complex solutions involving cryptographic means.

A. Keys and cryptosystems

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. For encryption cryptosystem, this key is used for both encryption and decryption. For dynamic authentication cryptosystem (which enables a user to prove to another user that he knows a particular secret), 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, 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. 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 extracting the key from one of the users (device theft, side channel attackss, etc.), 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 [4]. 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.

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) 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.

B. 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**

Hence we propose several solutions:

- Randomized data encryption between tags and readers using random data provided by the reader (see Section IV-B1).
- The tag contains a secret key directly link to its ID and prove it knows it to the reader without revealing it (see Section IV-B2).
- 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 IV-B3 and IV-B4).

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 IV-D.1). We will say tag has an uncloneable ID or unique ID.

Note IV-B.1. *Shared secret methods can be used if the required security level is not very high: password authentication (Section IV-F) for example.*

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

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 ([5]) and SSH protocol version 2 ([6]) 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.

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 [7], [8], [9], 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[1]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 IV-B4).

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

There are two kinds of zero-knowledge proofs: honest verifier zero-knowledge proof (like Schnorr one [10]) and general zero-knowledge proof (like Okamoto one [11]). 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.

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 IV-D.1).

3) Physical Uncloneable Function (PUF): According to [12], 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.

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

First part of [13] 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* ([14]).

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 [9], 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.

C. Memory write protection

As seen in Section III, if write or kill operations are not locked or disabled, an attacker can easily make the system unavailable. The kill feature is provided by many tags to permanently disable the them. 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 IV-F, 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 IV-B1.

However two points shall not be forgotten:

- Man-in-the-middle attacks has to be (almost) impossible (see Section IV-B). A genuine tag interrogator must not be helpfull for an attacker device to pass the authentication in order to write data into tags.
- 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 [15].

D. 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.

1) Fragment authentication: Fragment authentication enables an aggregating company (i.e. an entity allowed to deliver aggregates) 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 IV-A) 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 IV-D3 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.

Note IV-D.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 IV-B2), signature or MAC makes tag indirectly totally uncloneable.*

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.

Note IV-D.2.

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) 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. This property is obtained by almost all solutions of the Section IV. However using a MAC algorithm instead of a hash function would be significantly less resource consuming. 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 2^n different inputs require to do 2^n (or 2^{n-1} in mean) requests 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).

E. 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 system. 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.

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

Warning IV-E.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. 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 IV-D) 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 IV-D) 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.

F. Use of password

Passwords are the simplest way to do an authentication. But, as explained in Section IV-B, 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.

G. Implementation

We implemented and evaluated aggregated objects with *Higgs-3* RFID tags using a security strength of 80 bits (i.e. 2^{80} operations are needed to break cryptographic primitives) and NIST approved primitives (HMAC, DSA and AES-CFB). Discussion of this implementation is beyond the scope of this paper, but details can be found in [16].

V. Related works

Aggregated objects principle differs from many RFID systems where the concept of identification is central and related to database supported information systems. In some works, the tag memories are used to store *semantic* information, such as annotation, keywords, properties [17], [18]. Our approach is in the line of this idea: RFID are used to store in a distributed way group information over a set of physical artifacts. The concept using distributed RFID infrastructure as pervasive memory storage is due to Bohn and Mattern [19].

Maintaining group membership information in order to cooperate with "friend devices" is a basic mechanism (known as *pairing* or *association*) in personal area networks (PAN) such as Bluetooth or Zigbee. Some personal security systems based on PAN for luggages were proposed [20], which enable the owner to monitor some of his belongings, such as his briefcase, and trigger an alarm when the object is out of range. A major drawback of active monitoring is the energy power which is required, as well as potential conflicts with radio regulations that can exist in some places, namely in airplanes.

Still in the context of Bluetooth, RFID has also been used to store PAN addresses in order to improve discovery and connexions establishment time [21]. It can be seen as storing "links" between physical objects, such as in coupled objects, but without the idea of a fragmented group. Yet another variant is *FamilyNet* [22], where RFID tags are used to provide intuitive network integration of appliances. Here, there is a notion of group membership, but it resides on information servers instead of being self-contained in the set of tags as in aggregated objects. Probably the closest concept to Ubi-Check is *SmartBox* [23], where abstractions are proposed to determine common high level properties (such as completeness) of groups of physical artifacts using RFID infrastructures.

VI. Conclusion

Aggregated objects are a pervasive computing architecture for integrity checking of group of physical objects with many possible applications. In this paper, we discussed the dependability properties of aggregated objects. The essential properties of this architecture, distributed and autonomous, reduce the vulnerabilities associated with traditional RFID systems. However, some threats still exist and requires appropriate defense depending on the application and its required security level. As we have shown, some solutions exists but the computing power and memory size limitations of current RFID implementations are still challenges for the most secure approaches, and are active research topics. However, there are applications scenarios, such as UbiPark, where current implementations provide a sufficient security level and strong practicle benefits.

References

[1] M. Banâtre, F. Allard, and P. Couderc, "Ubi-check: A pervasive integrity checking system," in *NEW2AN '09 and ruSMART '09: Proceedings of the 9th International Conference on Smart Spaces and Next Generation Wired/Wireless Networking and Second Conference on Smart Spaces*, 2009, pp. 89–96.

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

[3] G. Avoine and P. Oechslin, “RFID traceability: A multilayer problem,” *Financial Cryptography and Data Security*, pp. 125–140, 2005.

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

[5] T. Dierks and E. Rescorla, “The transport layer security (tls) protocol version 1.2,” RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008, updated by RFCs 5746, 5878.

[6] T. Ylonen and C. Lonvick, “The Secure Shell (SSH) Transport Layer Protocol,” RFC 4253 (Proposed Standard), Internet Engineering Task Force, January 2006.

[7] L. Batina, J. Guajardo, T. Kerins, N. Mentens, P. Tuyls, and I. Verbauwhede, “An elliptic curve processor suitable for rfid-tags,” *Int. Assoc. for Cryptologic Research ePrint Archive*, 2006.

[8] M. Braun, E. Hess, and B. Meyer, “Using elliptic curves on rfid tags,” *IJCSNS*, vol. 8, no. 2, p. 1, 2008.

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

[10] C. Schnorr, “Efficient signature generation by smart cards,” *Journal of cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

[11] T. Okamoto, “Provably secure and practical identification schemes and corresponding signature schemes,” in *Advances in CryptologyCRYPTO92*. Springer, 1993, pp. 31–53.

[12] S. Devadas, “Physical unclonable functions and applications,” http://people.csail.mit.edu/rudolph/Teaching/Lectures/Security/Lecture-Security-PUFs-2.pdf, last access on November 11th, 2011.

[13] L. Bolotnyy and G. 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.

[14] Verayo, “Verayo PUF RFID,” http://www.verayo.com/product/pufrfid.html, last access on July 17th, 2011.

[15] R. Nithyanand, G. Tsudik, and E. Uzun, “Readers behaving badly: Reader revocation in pki-based rfid systems,” Cryptology ePrint Archive, Report 465, Tech. Rep., 2009.

[16] F. Allard, M. Banâtre, F. B. Hamouda, P. Couderc, and J. F. Verdonck, “Physical aggregated objects and dependability,” INRIA Rennes Bretagne Atlantique, Campus Universitaire de Beaulieu - RENNES, FRANCE, Tech. Rep. 7512, January 2011.

[17] M. Banâtre, M. Becus, and P. Couderc, “Ubi-board: A smart information diffusion system,” in *NEW2AN ’08 / ruSMART ’08: Proceedings of the 8th international conference, NEW2AN and 1st Russian Conference on Smart Spaces, ruSMART on Next Generation Teletraffic and Wired/Wireless Advanced Networking*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 318–329.

[18] T. D. Noia, E. D. Sciascio, F. M. Donini, M. Ruta, F. Scioscia, and E. Tinelli, “Semantic-based bluetooth-RFID interaction for advanced resource discovery in pervasive contexts,” *Int. J. Semantic Web Inf. Syst.*, vol. 4, no. 1, pp. 50–74, 2008.

[19] J. Bohn and F. Mattern, “Super-distributed RFID tag infrastructures,” in *Proceedings of the 2nd European Symposium on Ambient Intelligence (EUSAI 2004)*, ser. Lecture Notes in Computer Science (LNCS), no. 3295. Eindhoven, The Netherlands: Springer-Verlag, Nov. 2004, pp. 1–12.

[20] R. Kraemer, “The bluetooth briefcase: Intelligent luggage for increased security,” https://www-rnks.informatik.tu-cottbus.de/content/unrestricted/teachings/2004/SS/ringVL/Ringvorlesung_Kraemer_04552004.pdf, last access on November 11th, 2011.

[21] T. Salminen, S. Hosio, and J. Riekki, “Enhancing bluetooth connectivity with RFID,” in *PERCOM ’06: Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 36–41.

[22] W. Mackay and M. Beaudouin-Lafon, “Familynet: A tangible interface for managing intimate social networks,” in *Proceedings of SOUPS’05, Symposium On Usable Privacy and Security*. ACM, jul 2005.

[23] C. Floerkemeier, M. Lampe, and T. Schoch, “The smart box concept for ubiquitous computing environments,” in *Proceedings of sOc’2003 (Smart Objects Conference)*, Grenoble, May 2003, pp. 118–121.