Strengthening the Tree-Based Hash Protocols against Compromise of some Tags

Julien Bringer[1], Hervé Chabanne[1] and Thomas Icart[1,2]

[1]Sagem Sécurité
[2]Université du Luxembourg

Abstract. In 2004, Molnar and Wagner introduced in [6] a very appealing scheme dedicated to the identification of RFID tags. Their protocol relies on a binary tree of secrets which are shared – for all nodes except the leaves – amongst the tags. Hence the compromise of one tag also has implications on the other tags with whom it shares keys. We introduce a modification of the initial scheme to allow us to strengthen RFID tags by implementing secrets with Physical Obfuscated Keys (POK). This doing, we augment tags and tree resistance against physical threats.

Keywords. RFID tags, Tree-Based Hash Protocol, POK.

1 Introduction

Radio Frequency Identification (RFID) tags are made of a small chip containing an unique identification number. They communicate over the air with the system via a reader. One of their main application is to track objects on which they are attached.

RFID systems have to deal with the scarcity of tags resources as well with the privacy needed for the tags identification. In [6,7], a protocol which seems well suited to handle these two constraints is introduced. Indeed, the identification protocol of Molnar *et al.* requires only limited cryptographic functionality and has some nice properties such as the delegation of some identifications from a Trusted Center to readers. This protocol relies on a binary tree of secrets. The secret corresponding to a leaf is uniquely associated to one tag, but all other secrets in the tree are shared between different tags. Thus, as it is studied in [9,3,8], the compromise of the keying material of some tag leads to know the shared keys with some other tags. In case of compromise of many tags, this could allow to track some non-compromised tags. This can be considered as a main threat to the privacy of the system. More generally, secrets closer to

the root are shared between more tags than those placed near the leaves. Even in a recent publication [1], where an improvement of the protocol of Molnar *et al.* is proposed to reduce this problem, the compromise of tags still leaks information about the keying material of the whole keying material of the system.

To thwart this, we want to augment the resistance of tags against physical threats. Physical Obfuscated Keys (POK) have been introduced by Gassend [4] as a mean to securely store a secret inside a chip. They are strongly related to Physical Unclonable Functions (PUF). Indeed, PUFs and POKs were introduced as a proposition to implement keys in a more secure manner. They are built such that their observation by an adversary will corrupt the chip and then destroy them. Note that the use of PUF inside RFID tags has already been considered in [10,2].

The main achievement of this paper is to describe how to replace each secret by two POKs during the Tree-Based Hash protocol. Each one taken separately does not reveal anything on the secret. And they are turned on alternately. Cryptographic computations are carried out with two steps. During a step, only one POK is activated. An adversary can access – by sacrificing the chip – to one POK. By construction the underlying key is thus safe from this compromise of one POK.

Our paper is as follows. In Sect. 2, we recall the principles of the Tree-Based Hash Protocols [7] and those of the POK. In Sect. 3, we describe our modification of the protocol. Section 4 examines the security of our proposition. Section 5 concludes.

2 Notations and definitions

2.1 The Hash-Tree protocol [7] in a nutshell

We here only describe the general principles of the Hash-Tree protocol and invite the readers to go through [7] to get full details.

During system initialization, the Trusted Center generates a tree of secrets (keys), for instance a binary one. Each leaf is associated to a tag. A tag knows all keys $K_1, \ldots, K_s$ along the path from the root to its leaf. Let F denotes an appropriate public pseudo-random function. When a tag is interrogated by a reader which sends to it a random value r, it responds by generating a new pseudonym each time –$F_{K_1}(r, r')$, $F_{K_2}(r, r')$, ..., $F_{K_s}(r, r')$– where r' is another random value generated and transmitted

by the tag. The Trusted Center can easily check to which key corresponds the received pseudonym in its tree of secrets by verifying for a given (r, r'):

1. to which node corresponds $F_{K_1}(r, r')$,
2. between the 2 children of this node, which whom is associated with $F_{K_2}(r, r')$,
3. repeat this verification, level after level from the root to the leaves,
4. and then identify which leaf (tag) comes with $F_{K_s}(r, r')$.

Fig. 1 below illustrates this scheme.

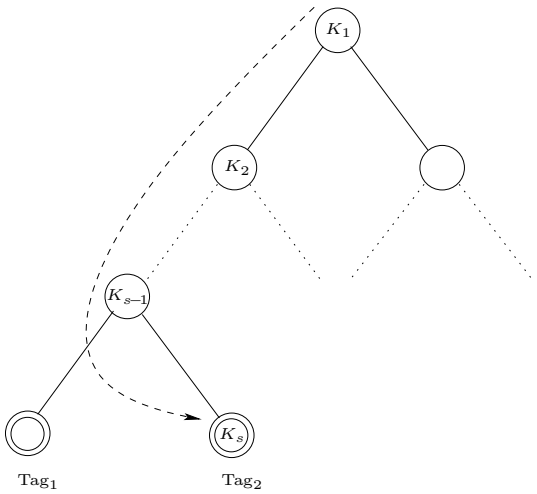

Fig. 1. One descent in a binary tree of secrets

A practical example. To get a better idea of the involved figures, we take back the example given in [7]. They have 2^{20} tags. The binary tree is replaced by a tree with a branching factor of $Q = 2^{10}$ and is made of 2 levels. Each tag stores two 64-bit secrets. Using tree-based authentication protocol enables to reduce the number of try a Trusted Center needs to do. In this example, a Trusted Center needs to compute only $2 * 2^{10}$ times the function F, 2^{10} for each round, instead of 2^{20} without this protocol. This is a really interesting improvement, because if the system's size is S, the numbers of computation for the Trusted Center is always in $O(log_Q(S)Q)$ computation.

2.2 Physical Unclonable Function (PUF)

Gassend in [4] introduces the concept of PUF which has very interesting properties:

1. easy to evaluate,
2. hard to characterize, from physical observation or from chosen challenge-response pairs,
3. hard to reproduce.

For a given challenge, a PUF always gives the same answer. The hardness of characterization and the reproduction is stronger than polynomial; i.e. it is impossible to reproduce or to characterize the PUF thanks to a polynomial amount of resources (time, money ...). And PUF can be viewed as pseudo-random function where the randomness is insured thanks to physical properties.
One example of PUF, as mentioned in [10] as I-PUF for Integrated Physical Unclonable Function, has another interesting properties:

1. The I-PUF is inseparably bound to a chip. This means that any attempt to remove the PUF from the chip leads to the destruction of the PUF and therefore of the chip.
2. It is impossible to tamper with the communication (measurement data) between the chip and the PUF.
3. The output of the PUF is inaccessible to an attacker.

These properties insure the impossibility to analyse physically a PUF without changing its output. Hence, physical attacks corrupt the PUF and the chip leaving the attacker without any information about the PUF. Silicon PUF have been already described in [5] and can be taken as relevant examples of I-PUF, they are based on delay comparison among signals running through random wires. Moreover, they only require a few resources to be implemented.

In this paper, we suppose the output of a PUF will not contain any errors, whatever the external conditions are. As of today, it is still difficult to get experimentally this result but in [10], it is explained how to handle this problem thanks to error correcting codes. The analysis of such technique is beyond the scope of this paper.

2.3 Physicaly Obfuscated Key (POK)

In [4], it is shown how to implement a key with a PUF in a physically obfuscated manner, i.e. how to get a POK from a PUF:

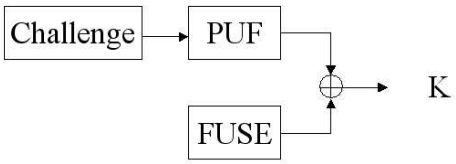

Fig. 2. First example of POK

Suppose you have a key K, and you want to use it in different tags, but with different PUF in each tag. To create a POK, one solution is to hardwire the PUF with a challenge. That response is combined with the contents of some fuses via an exclusive-or operation to produce K as shown in Fig. 2.
In a tag, it would not be reasonable to have a lot of different PUFs, firstly for a matter of space, and secondly for a matter of security. If there are a multitude of PUF, the possibility to attack some of them without destroying the others is unclear. Fortunately, it is easy to generalize the idea of one POK, as illustrated in Fig. 3.

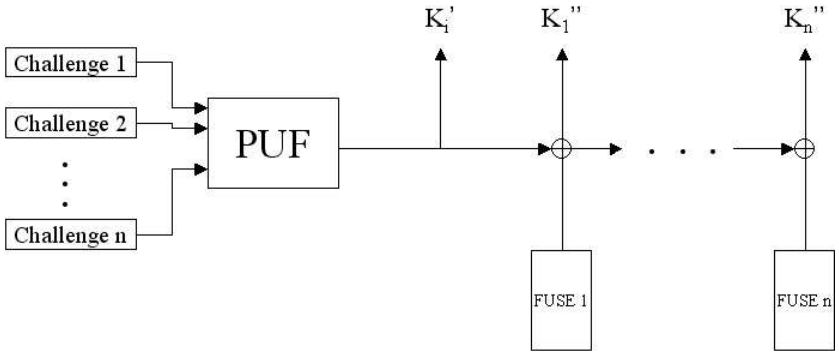

Fig. 3. Multiple POKs with a unique PUFs

To get the key K''_i, the ith challenge is switched on, and the results is an exclusive-or between the output of the PUF and some fuses. The fuses have to be set during the fabrication of the tag.

3 Our Proposition

3.1 POKs used

Whatever is the use of the tag, the key has to be stored digitally when involved in some computations. Consequently, it could be possible to get dump of the volatile memory and then to obtain the value of the key. This type of attack has been described in [2] with a general line of defense: split the computations with the key in two steps. Of course, the difficulty we here encounter is to cope with cryptographic computations and to find a way to split them.
A key K of the tree would be hard-coded thanks to two POKs K' and K'' such that $K = K' \oplus K''$. More precisely, K' will be the result of the PUF to a challenge, K'' is also the result of another challenge combined with a fuse by an exclusive-or to get the desired K (cf. Fig. 3). Consequently, K' and K'' will be different for each tag. Therefore, the non-volatile memory needed to store one key as two POKs is just a fuse and some small challenges. This is about the same amount of memory as the one needed to store one key in the usual case.
Note that challenges used to stimulate the PUF to generate keys are stored in the tag. Because the equality $K = K' \oplus K''$ stands for all tags in the same branch, neither K' and K'' need to be known from the outside, nor pairs of input/output from the PUF do.

3.2 The Protocol

In Fig. 4 is the description of one stage i of our new protocol in a tree with branching factor Q, where H is a pseudo-random function and $\hat{K}_i^j$ is the j^{th} key in the current branch of the system tree at the i^{th} level.

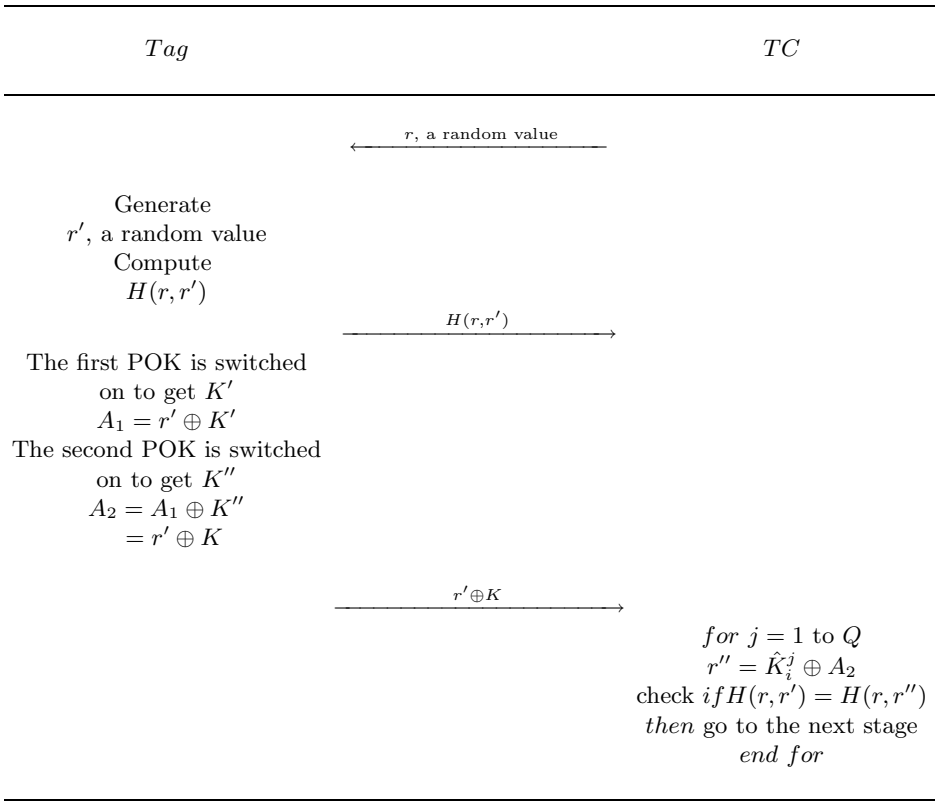

Fig. 4. One stage of the Authentication protocol

We here describe the protocol to authenticate a tag from a trusted center. Suppose the system is composed of 2^{20} tags, and the tree of secret has a branching factor of 2^{10}.

The authentication begins when the tag is brought near a reader which supplies power by electromagnetic field. Then the TC sends to the tag a challenge r, which is a random value. The tag then computes a random value r' and sends $H(r, r')$. This is equivalent to a signature of r with the key r'. Then the tag switches on the first POK to get K_1' and then computes $A_1 = r' \oplus K_1'$. This operation will erase in volatile memory r'. Then the second POK is switched on to get K_1'', and this erases K_1'. Finally the tag computes $A_2 = A_1 \oplus K_1''$ and sends A_2. Then the TC will try amongst all the key $\hat{K}_1$ in the tree's first level whether it gets the equality $H(r, A_2 \oplus \hat{K}_1) = H(r, r')$. If he finds one correct key, the protocol will continue in the branch of our tree under the node marked by the found key. Then the computation are made once more but the tag switches on two other POKs to get K_2' and K_2'' and the TC will make another exhaustive search amongst keys in the current branch. If this time again, the TC is able to determine a key, then the tag will be authenticated.

This protocol has all the advantages of the tree based authentication, it allows delegation and to have less computation for the TC but with an increased time of computation for the tag. In fact, a trade-off has to be found:

- with a small branching factor in the above protocol, tags will be used an important numbers of times which leads, because of the cpu power of tags, to a long computation,
- with a big branching factor, tags will just be used a small number of times, but the TC will have to compute more try for each stage.

The best trade-off for the branching factor is reached when the time taken by tags and the one taken by the TC during the protocol are equals.
Concerning memory aspect, our proposal does not change anything for the TC, as the different PUF challenges and split keys are only used on tag's side.

3.3 Size of the parameters

We propose for our protocol, as an example, the following parameters:

– the size of r' is 100 bits and r is 64 bits,
– K' and K'' are 100 bits' keys,
– $H(r, r')$ will be the first 42 bits of $AES_{r||r'_{1..64}}(r_{1..28}||r')$.

They have been chosen to minimize the non-volatile memory inside the tag and the communication between tags and readers, but they should lead to a sufficient security to insure the secrecy of the keys and the impossibility to authenticate without the knowledge of the keys.

The parameters have been chosen to get a security of 2^{64} AES (cf. Sec. 4) but they can be augmented to reach standard security of 2^{80} AES. This could be done with the following parameters:

– the size of r' is 116 bits and r is 64 bits,
– K' and K'' are 116 bits' keys,
– $H(r, r')$ will be the first 50 bits of $AES_{r||r'_{1..64}}(r_{1..12}||r')$.

The two tables Tab. 1 and Tab. 2 summarize the concrete resources used in our scheme in the two previous cases for some example parameters. We use a branching factor of 2^{10} in all cases.

Tag			Tag → TC	TC
numbers	non-volatile memory	computation	communication	computation
2^{20}	200 bits	2 AES and 2 random	284 bits	2×2^{10}
2^{30}	300 bits	3 AES and 3 random	426 bits	3×2^{10}

Table 1. Resources needed in the first case

Tag			Tag → TC	TC
numbers	non-volatile memory	computation	communication	computation
2^{20}	232 bits	2 AES and 2 random	332 bits	2×2^{10}
2^{30}	348 bits	3 AES and 3 random	498 bits	3×2^{10}

Table 2. Resources needed in the second case

4 Security Analysis

4.1 Opening a tag will leak no information about the keying material

If we propose to use POK in tags, it is because it is quite easy to steal an unprotected key inside one, and then to authenticate using its key. Here we prove such a thing is not possible using our implementation and the usual model for PUFs, particularly because inside each tag, the keys K' and K'' are different.

First of all, it is interesting to notice that the output of a tag is indistinguishable from a random output. Because of the randomness of r', $r' \oplus K$ is indistinguishable from a random value, and because H is a hash function, this property is also true for $H(r, r')$. This leads to the fact that an observer could not retrieve keys only with tags's outputs, without inverting the hash function by brute force search.
During the computation of the tag, an attacker by opening a tag, and therefore destroying it, could only get by reading the versatile memory one of the following couples:

- r' and $H(r, r')$,
- r' and K',
- A_1 and K''.

We can suppose the attacker knows $H(r, r')$ in all the cases because it is sent in clear during the protocol. In the two first cases, he only gets random informations. In the last case, the problem to find K' thanks to A_1 and $H(r, r')$ is as difficult as the problem to recover K from A_2 and $H(r, r')$, which are indistinguishable from a random value, consequently, in our model of PUF, it is useless to open tags to get information about keys.

4.2 Classical Security Analysis

We will here study the difficulty for an attacker to authenticate without the key in a first part. In a second part, we will study the difficulty for a simulator to discover the keys inside a tag just by interrogate it.

To study the security, we will suppose that the attacker is able to get 2^{15} tags. We assume that a tag support at most 2^{20} executions of the

protocol. Our tree has a branching factor of $Q = 2^{10}$ and the total system is composed with 2^{20} tags, which means that a tag has exactly two keys. The total number of keys in our system is $2^{20} + 2^{10}$, 2^{10} in the first stage of the tree, and 2^{20} in the second stage. It is also important to notice that between two tags, there is no hint to distinguish whether or not they have one common key.

Authentication without keys An adversary who wants to manufacture forged tags will encounter difficulty. Firstly, he would not be able to duplicate an existing tag because of the PUF inside. Secondly, without keys inside a tag, he will have a probability $Q \times 2^{-42}$ to authenticate at each step. Therefore, the forger tag will have a probability to authenticate in 2^{-64}.
An attacker could try to store answers of tags for different r in our protocol. He could thus get 2^{20} answers from a key. The probability to get one of these r from the TC at the same stage is 2^{-42}, so this leads to a total probability 2^{-84} to achieve a complete authentication. It is noticeable that with a probability $Q \times 2^{-42}$, a genuine tag could fail in the protocol, in the case where the equality $H(r, r') = H(r, r' \oplus K \oplus K')$ is true with K' another key in the tree. The probability that a genuine tag authenticates with a wrong ID is $(Q \times 2^{-42})^d$ where d is the depth of our tree, which is 2^{-64} in our case.

Getting keys of a tag A simulator to get a key K will challenge its tags with the same r. He will get a total amount of $2 \times 2^{15} \times 2^{20} = 2^{36}$ couples $(H(r, r'), A_2)$. Note that, because the tags are indistinguishable, he can only determine more than 2^{20} couples for one key with a negligible probability. On one hand, he could try a brute force search on the key K and then check the key by simulating the protocol, but to recover one key amongst the 2^{16} ones associated to its own tags, he would have to try 2^{84} keys. On the other hand he could try to perform brute force search to find one r' pre-images of one of the $H(r, r')$ pre-computed. A random r' has a probability $2^{-6} = 2^{36-42}$ to be a pre-image of one of the precomputed $H(r, r')$ but it will be the r' used in the protocol with a probability $2^{42-100} = 2^{-58}$, therefore to find one r' and thus one key, 2^{64} execution of AES are needed.

5 Conclusion

We modify the Tree-Based Hash protocols to allow the integration of POKs. Then thanks to the fact that keys inside a tag are now physically obfuscated, we show that an adversary even with a large number of tags is not able to obtain one key. We thus believe that our work helps to strengthen the security of the overall protocol.

Our ideas seems to follow a general trend in inserting PUFs inside RFIDs. As of today, to the best of our knowledge, no results are published for tags on the ability to integrate this new physical cryptographic primitive nor to validate the security model they imply, we propose this topic as a possible avenue for future research.

References

1. G. Avoine, L. Buttyán, T. Holczer, and I. Vajda. Group-based private authentication. In Proceedings of the International Workshop on Trust, Security, and Privacy for Ubiquitous Computing (TSPUC 2007), IEEE., 2007.
2. Leonid Bolotnyy and Gabriel Robins. Physically unclonable function-based security and privacy in RFID systems. In *International Conference on Pervasive Computing and Communications – PerCom 2007*, pages 211–220, New York, USA, March 2007. IEEE, IEEE Computer Society Press.
3. Levente Buttyán, Tamás Holczer, and István Vajda. Optimal key-trees for tree-based private authentication. In *Privacy Enhancing Technologies*, pages 332–350, 2006.
4. Blaise Gassend. Physical random functions. Master's thesis, Computation Structures Group, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 2003.
5. Blaise Gassend, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas. Silicon physical random functions. In Vijayalakshmi Atluri, editor, *ACM Conference on Computer and Communications Security*, pages 148–160. ACM, 2002.
6. D. Molnar and D. Wagner. Privacy and security in library RFID: issues, practices, and architectures. In *Proceedings of the ACMConference on Computer and Communications Security*, pages 210–219, 2004.
7. David Molnar, Andrea Soppera, and David Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags. In *Selected Areas in Cryptography*, pages 276–290, 2005.
8. Yasunobu Nohara, Sozo Inoue, Kensube Baba, and Hiroto Yasuura. Quantitative evaluation of unlinkable id matching systems. In *Workshop on Privacy in the Electronic Society*, 2006.
9. Karsten Nohl and David Evans. Quantifying information leakage in tree-based hash protocols (short paper). In *ICICS*, pages 228–237, 2006.
10. Pim Tuyls and Lejla Batina. RFID-tags for anti-counterfeiting. In David Pointcheval, editor, *CT-RSA*, volume 3860 of *Lecture Notes in Computer Science*, pages 115–131. Springer, 2006.