A Scalable and Provably Secure Hash-Based RFID Protocol

Gildas Avoine and Philippe Oechslin
EPFL, Lausanne, Switzerland
{gildas.avoine, philippe.oechslin}@epfl.ch

Abstract

The biggest challenge for RFID technology is to provide benefits without threatening the privacy of consumers. Many solutions have been suggested but almost as many ways have been found to break them. An approach by Ohkubo, Suzuki and Kinoshita using an internal refreshment mechanism seems to protect privacy well but is not scalable. We introduce a specific time-memory trade-off that removes the scalability issue of this scheme. Additionally we prove that the system truly offers privacy and even forward privacy. Our third contribution is an extension of the scheme which offers a secure communication channel between RFID tags and their owner using building blocks that are already available on the tag. Finally we give a typical example of use of our system and show its feasibility by calculating all the parameters.

1 Introduction

The main goal of Radio Frequency Identification (RFID) systems is to identify objects remotely by embedding tags, tiny devices capable of transmitting data, into these objects. Goods in stores can be tagged in order to prevent shoplifting, or to speed up the goods registration process by using wireless scanning instead of human or optical scanning. RFID tags are different from bar-codes. Firstly, each tag contains a unique identifier while bar-codes represent a lot identifier. Secondly, tags can be identified at distance, e.g. several meters, without any optical or visual contact. These aspects introduce important privacy issues, in particular the traceability of the tags by unauthorised parties.

Privacy issues. Privacy is one of the most serious problems related to RFID. As long as this technology suffers from privacy issues, consumers will reject it. Some famous examples are the boycotts against Benetton or against Gillette [14].

We define *privacy* as follows: Given a set of readings between tags and readers, an adversary must not be able to find any relation between any readings of a same tag or set of tags. Since tags are not tamper-resistant, an adversary may even obtain the data stored in the memory of the tags additionnaly to the readings from the readers/tags. He thush might become capable of tracking the past events of the tags, given their content. We therefore define *forward privacy*: Given a set of readings between tags and readers and given the fact that all information stored in the involved tags has been revealed at time t, the adversary must not be able to find any relation between any readings of a same tag or set of tags that occurred at a time $t' \leq t$.

Privacy also includes the fact that a tag must not reveal any information about the kind of item it is attached to.

Related work. Privacy can be enforcred in RFID systems by "repressive" techniques (physical destruction, blocker tag [8], etc.) or "preventive" techniques (e.g. [5, 6, 7, 9, 11, 13]). Unfortunately, only few protocols of the latter category are secure (see [1, 2]). We focus in this paper on the scheme proposed by Ohkubo, Suzuki, and Kinoshita [11] which can be proven to be secure, as we will do below. As most of the existing schemes, the basic concept of Ohkubo *et al.* consists of refreshing the identifier of the tag each time it is queried by a reader. However, this scheme differs from some others in the sense that the tag refreshes its identifier by itself. Unfortunately this scheme is not scalable because it has a high complexity in terms of computation during the identification.

Our contribution. After describing Ohkubo *et al.*'s scheme in Sect. 2, we introduce in Sect. 3 a technique to significantly reduce complexity using a time-memory trade-off. We then show that the scheme can be improved by supplying a forward secure channel from the tag to its owner. In Sect. 5, we prove the security of the system and finally we give a real life example in Sect. 6.

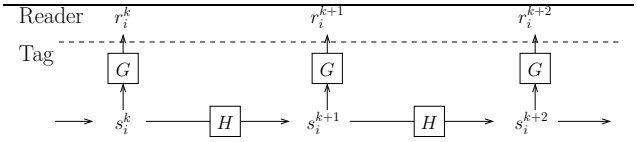

Figure 1. Refreshment of the identifier

2 Ohkubo, Suzuki, and Kinoshita

Description. The basic idea of Ohkubo *et al.* [11] is to modify the identifier of the tag each time it is queried by a reader such that the identifiers can be recognized by authorised parties only. The tag refreshes its identifier autonomously, using two hash functions G and H as described below. We consider a back-end database B (the only party authorised to track the tags), readers R, and tags T_i $(1 \leq i \leq n)$. Readers are (untrusted) devices that do not have cryptographic functionalities; We do not assume that the tags are tamper resistant, but we suppose that a hash function can be embedded into the tags, which may soon be a rather realistic assumption (see [3]).

Setup. The personalisation of a tag T_i consists of storing in its memory a random identifier s_i^1, which is also recorded in the database B. Thus, B initially contains the set of random values $\{s_i^1 \mid 1 \leq i \leq n\}$. Two hash functions G and H are chosen

Interrogation. When a reader queries T_i, it sends an identification request to the tag and receives back $r_i^k := G(s_i^k)$ where s_i^k is the current identifier of T_i. While T_i is powered, it replaces s_i^k by $s_i^{k+1} := H(s_i^k)$. Finally, the reader sends r_i^k to B (see Figure 1).

Identification. From r_i^k, B has to identify the corresponding tag. In order to do this, B constructs the hash chains from each n initial value s_i^1 until it finds the expected r_i^k or until it reaches a given maximum limit m on the chain length.

Complexity analysis. There are two main usage scenarios of RFID tags: usage in *closed environments* and usage in *open environments*. In closed environments, all tags are owned by B and queried by B's readers only. This scenario is typically the usage of tags within a shop to facilitate the handling of goods in the shop and the scanning at the check-out. The tags only need to be active while they are confined to the premises of the shop.

In open environments, some queried tags are not owned by B and some of B's tags are also queried by foreign readers. A typical example is a library that lends books to its customers. The same book can leave the library and be returned many times. The customers do not want any malicious readers outside the library to know what they are reading or to be able to trace them.

Ohkubo *et al.*'s scheme has a complexity in terms of hash computations of mn in closed environment (2 hash operations are carried out $mn/2$ times), and of $2mn$ in open environment since B computes all the hash chains when trying to identify a foreign tag. Thus, when the number of tags n or the number of read operations m in their lifetime is large the complexity becomes unmanageable.

3 Time-memory trade-off

To reduce the complexity of [11], we suggest using a specific time-memory trade-off based on Hellman's original work [4] and recent optimisations by Oechslin [10]. This type of trade-off reduces the amount of work T needed to invert any given value in a set of N outputs of a one-way function F with help of M units of memory. The efficiency follows the rule $T = N^2\gamma/M^2$ where γ is a small factor depending on the probability of success and the particular type of trade-off being used. Compared to a brute-force attack, the trade-off can typically reduce the amount of work from N to $N^{2/3}$ using $N^{2/3}$ units of memory. In general the trade-off works as follows: A *reduction function* R is defined, which generates an arbitrary input to F from one of its outputs. By alternating F and R, chains of inputs and outputs of F can be built. If enough chains of a given length are generated, most outputs of F will appear at least once in any chain. Only the first and the last element of each chain is stored in a table. Given one output r of F that we need to invert, we generate a chain starting at r. If r was part of any chain that we stored we will eventually generate last element, which is in the table. Looking up the corresponding start of the chain, we can regenerate the complete chain and find the input to F that yields the given output r. To assure a high success rate, several tables have to be generated with different reduction functions. The exact way of doing this is what differentiates existing trade-off schemes. Details can be found in [10].

Finding an appropriate trade-off. In our case the function F is

$$F : (i, k) \mapsto r_i^k = G(H^{k-1}(s_i^1))$$

where $1 \leq i \leq n$ and $1 \leq k \leq m$. In order to apply the trade-off we need an arbitrary reduction function R,

$$R : r_i^k \mapsto (i', k')$$

where $1 \leq i' \leq n, 1 \leq k' \leq m$. For example, we take

$$R(r) = (1 + (r \bmod n), 1 + (\left\lfloor \frac{r}{n} \right\rfloor \bmod m)).$$

There are two particularities in this situation:
(1) The brute force method, as proposed in [11] needs $M_{bf} = n|s|$ units of memory to store the n values of s_i^1. Usually, brute-force methods do not require any memory.
(2) When used in the trade-off, F is more complex than when used in the brute-force. Indeed, in the brute-force, the hash chains are calculated sequentially, thus needing just one H and one G calculation at each step. In the trade-off, i and k are arbitrary results from R and have no incremental relation with previous calculations. Thus, on average, each step needs $m/2 + 1$ operations.

Since the brute-force approach already uses a certain amount of memory, it makes sense to measure the amount of memory needed by the trade-off in multiples of M_{bf}. We call c the ratio between the memory used by the trade-off and the memory used by the brute-force. E.g. $c = 5$ means that we need one M_{bf} of data to store the s_i^1 and 4 times this amount to store the chains for the trade-off. M_{bf} is a multiple of the size of s, whereas the memory used by the trade-off is a multiple of the size of a chain. A chain is stored by storing its start and end point. These points can be either the output of F or its input. In our case the input is smaller, we thus chose to store two pairs of (i, k), thus requiring $2(|n| + |m|)$ bits of memory. The conversion factor from units of brute-force to units of trade-off is thus $\mu = |s|/(2|n| + 2|m|)$. In the scenarios we are interested in, μ is typically between 2 and 4. Knowing all this information we can now rewrite the trade-off relation:

$$T = \frac{n^2 m^2}{(c-1)^2 \mu^2 n^2}(\frac{m}{2} + 1)\gamma \approx \frac{m^3 \gamma}{2(c-1)^2 \mu^2}.$$

The advantage over the brute-force approach is the following, in the average case:

$$\frac{T_{bf}}{T} \approx 2\frac{(c-1)^2 \mu^2 n}{m^2 \gamma}.$$

As expected, the gain in speed increases with the square of the available memory. The trade-off is especially efficient if n is large compared to m. We will see later how m – the number of times a tag can be read – can be reduced.

Optimising the trade-off. So far we have a trade-off where we have to sacrifice one share of the memory to store the beginning of the hash-chains (hence the factor $(c-1)$) and the function F has a complexity of $\frac{m}{2} + 1$ because we need to generate arbitrary elements of the chains. If we not only store the first element of the chains, but also the element at the middle of the chain, we sacrifice even more memory but we reduce the average complexity of F. We will have only $(c-2)$ shares of the memory available for the tables, but F will have a complexity of $\frac{m}{4} + 1$ (we need to generate only a quarter of a chain on average). In general, if we store x values per chain, sacrificing x shares of memory, the complexity of the trade-off becomes:

$$T = \frac{n^2 m^2}{(c-x)^2 \mu^2 n^2}(\frac{m}{2x} + 1)\gamma \approx \frac{m^3 \gamma}{2x(c-x)^2 \mu^2}.$$

We quickly find that $x = \frac{c}{3}$ minimises this expression and that the complexity of the optimised trade-off is :

$$T_{opt} \approx \frac{3^3}{2^3}\frac{m^3 \gamma}{c^3 \mu^2}.$$

(P)recalculation of the tables. Before the trade-off can be used to read the tags, the trade-off chains must be generated and their start and end stored in the memory. Since the chains contain arbitrary hashes, we need to generate slightly more than nm hashes to ensure that each hash appears at least once in the tables with a high probability. Again, the hashes are not created sequentially and each calculation of F incurs about $\frac{m}{2} + 1$ hash calculations. The effort to create the tables is thus $T_{precalc} \approx nm^2/2$. This complexity is reduced by the fact that we store intermediate elements of the chains is some part of the memory.

If the set of tags in the system stays the same, the tables only need to be calculated once. If new tags must be added, the tables must be recalculated. Extra tags can be included in the tables, so that they do not need to be recalculated for every single new tag. Every time the tables are recalculated we can also remove the tags that are no longer in use. Typically the tables could be recalculated off-line every night, week or month.

Keeping m low increases the advantage of the trade-off over the brute-force method. The following procedure can be applied to keep m small. In the database that contains the s_i^1 we can keep track of how many times we have read each tag. We know that the next time we read the tag, the result will be further down the hash chain. If tag i has been read k times, we can thus replace s_i^1 by s_i^k in the database when the next recalculation of the tables occurs. m is thus no longer the number of times a tag is read in its lifetime but the maximum of times it is read between two recalculations of the tables, or the maximum of times it is read by a foreign reader. Note that the adjustment of s_i^1 makes

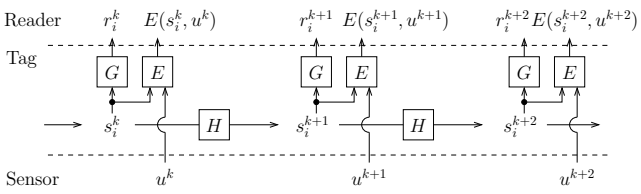

Figure 2. Additional forward secure channel

both the trade-off and the brute-force method faster but it increases the speed-up factor between the two.

Time-memory trade-offs are probabilistic, thus there is an arbitrarily small chance that a tag may not be found in the tables because a particular s_i^k is not part of any chain that was generated. A pragmatic approach to this problem is simply to read the tag a second time in such a case (hoping that s_i^{k+1} will be found). A more deterministic approach would be to keep score of the hash values that are generated when the tables are calculated and to eliminate the s_i^1 for which not all hash values have appeared.

4 Forward secure backward channel

In some applications, tags may contain sensors that supply some environmental information to their database. For instance, the tire manufacturer Michelin has decided to implant RFID tags inside the rubber sidewall of its tires. These tags can identify tires belonging to a defective batch. They could be adapted to directly inform the driver whether the tires are properly inflated [12]. In such applications, communication confidentiality has to be ensured because the data sent by the tag could allow it to be tracked. In our system, the tag T_i can use its identifier s_i^k as a session secret key to encrypt data u^k coming from its sensors, as described on Figure 2, thus supplying a forward secure channel from the tag to B.

5 Security analysis

Given a set of identification sessions $\mathcal{S} = \{(r_i^k, E(s_i^k, u^k)), \mid i \in \{1..n\}, k \in \mathbb{N}\}$ where $E(s_i^k, u^k)$ is the encryption of u^k under the symmetric key s_i^k, the privacy is ensured if an attacker cannot find $(r_i^k, E(s_i^k, u^k)) \in \mathcal{S}$ and $(r_{i'}^{k'}, E(s_{i'}^{k'}, u^{k'})) \in \mathcal{S}$ such that $i = i'$ with a probability better than randomly picking two elements of $\mathcal{S}$ according to the uniform distribution. The proof is straightforward in the random oracle model: all the s values are independent and uniformly distributed due to the properties of H (note that the s_i^1 are chosen randomly according to the uniform distribution). Hence all the r values are also uniformly distributed since G is a random function, and the same finally applies for the encrypted values since the keys i.e. the s values, are used once. Thus *privacy* is ensured. *Forward privacy* follows from this result: reading the content of the tag, the adversary can only obtain the current random s which has not been used yet (previous s have been erased after having used), thus not revealing any information. Given forward privacy and given the fact that the encryption keys are erased and that one knowledge of one key do not allow one to recover the previous ones, *forward secrecy* is straightforward.

6 Privacy-friendly tags in a library

The following example illustrates the use of privacy-friendly tags in a real scenario. A campus library uses RFID tags to identify each of the books and reviews that it lends. Inside the library, the tags make it possible to scan shelves for misordered books and to identify books that have not been placed in the shelves. The tags also make the check-out and check-in of books much easier. When a user takes the books home, a reader that is not connected to the database of the library cannot find out what he is reading nor track him because of the tags. We assume that the library wants to be able to serve up to 1 million items ($n = 2^{20}$) and the number of read operations on a single tag between two recalculations of the tables is 1024 ($m = 2^{10}$). The size of s is chosen to be 128 bits, in order to be compatible with [3]. We further assume that the system is capable of carrying out 2^{24} hash operations per second. The amount of memory needed to store the one million s values is 16 megabytes. The average reading time using the brute-force approach is

$$T_{bf} = nm = 2^{30} \quad \text{i.e. 1 minute.}$$

Since a pair of (i, k) is 30 bits large we need at most 60 bits to store one chain. We can thus store more than two chains in the same amount of memory it takes to store one s ($\mu \geq 2$). Assuming that all calculations are carried out on a single back-end equipped with 1 Gigabyte of memory ($c = 64$) and that we chose a success rate of 99.9% ($\gamma = 8$) the time to read a tag with our method is

$$T_{opt} \approx \frac{3^3}{2^3} \frac{m^3 \gamma}{c^3 \mu^2} \quad \text{i.e. 0.0016 seconds.}$$

The time necessary to calculate the tables is

$$T_{precalc} \approx \frac{nm^2}{2} \text{ i.e. 10 hours.}$$

The advantage of the time-memory trade-off over the brute force is shown in Table 1 and Figure 3.

$n = 2^{20}, m = 2^{10}, \gamma = 8, \lvert s\rvert = 128, \mu = 2$ 2^{24} operations per second, $T_{bf} = 64$ [s]						
c	2	4	8	16	32	64
M[MB]	32	64	128	256	512	1024
T_{opt}[s]	54	6.8	0.84	0.11	0.013	0.0016
T_{bf}/T_{opt}	1	9.5	76	606	4854	38836

Table 1. Amount of memory used, reading time and ratio of reading times of both methods

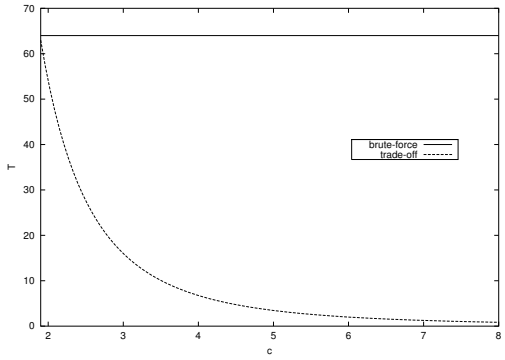

Figure 3. With the trade-off, the reading time reduces with the cube of the amount of memory

7 Conclusion

Building on Ohkubo *et al.*'s scheme, we have suggested a complete solution for RFID systems which is scalable and ensures forward privacy. Our system is also particularly relevant when sensors are embedded into the tags because the data sent to the back-end database through the readers can be encrypted in a forward secure way To ensure the scalability of our scheme we have developed a time-memory trade-off specifically optimized to reduce the amount of computation in the system. Finally we have illustrated the efficiency of our proposal in a real life application, where RFID tags are used to identify books in a library; in this scenario, our scheme requires only 1.6 millisecond to identify a book, while Ohkubo *et al.*'s scheme would take 1 minute. Our solution is not limited to RFID systems but it is best suited for systems that need forward privacy and forward secrecy while they cannot afford asymmetric cryptographic primitives, for example sensor networks.

References

[1] Gildas Avoine. Privacy issues in RFID banknote protection schemes. In *CARDIS*, Kluwer Academic Publishers, 2004.

[2] G. Avoine and Ph. Oechslin. RFID traceability: A multilayer problem. In *Financial Cryptography – FC'05*, *LNCS*, Springer, 2005.

[3] M. Feldhofer, S. Dominikus, and J. Wolkerstorfer. Strong authentication for RFID systems using the AES algorithm. In *Workshop on Cryptographic Hardware and Embedded Systems – CHES 2004*, *LNCS*, Springer, 2004.

[4] M. Hellman. A cryptanalytic time-memory trade off. *IEEE Transactions on Information Theory*, IT-26(4):401–406, 1980.

[5] D. Henrici and P. Müller. Hash-based enhancement of location privacy for radio-frequency identification devices using varying identifiers. In *PerSec 2004*, IEEE Computer Society, 2004.

[6] A. Juels. Minimalist cryptography for low-cost RFID tags. In *The Fourth International Conference on Security in Communication Networks – SCN 2004*, *LNCS*, Springer, 2004.

[7] A. Juels and R. Pappu. Squealing euros: Privacy protection in RFID-enabled banknotes. In *Financial Cryptography – FC'03*, *LNCS*, Springer, 2003.

[8] A. Juels, R. Rivest, and M. Szydlo. The blocker tag: Selective blocking of RFID tags for consumer privacy. In *Conference on Computer and Communications Security – CCS*, ACM Press, 2003.

[9] D. Molnar and D. Wagner. Privacy and security in library RFID: Issues, practices, and architectures. In *Conference on Computer and Communications Security – CCS*, ACM Press, 2004.

[10] Ph. Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology – CRYPTO'03*, *LNCS*, Springer, 2003.

[11] M. Ohkubo, K. Suzuki, and S. Kinoshita. Cryptographic approach to "privacy-friendly" tags. In *RFID Privacy Workshop*, MIT, USA, 2003.

[12] RFID Journal. `http://www.rfidjournal.com`.

[13] S. Sarma, S. Weis, and D. Engels. RFID systems and security and privacy implications. In *CHES 2002*, *LNCS*, Springer, 2002.

[14] Stop RFID. `http://www.stoprfid.org/`.