

An Efficient Authentication Protocol for RFID Systems Resistant to Active Attacks

Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan M. Estevez-Tapiador,
and Arturo Ribagorda

Computer Science Department, Carlos III University of Madrid,
{pperis, jcesar, jestevez, arturo}@inf.uc3m.es

Abstract. RFID technology is a ubiquitous technology, and seems destined to become more and more ubiquitous. Traditional cryptographic primitives are not supported on low-cost RFID tags since, at most, 4K gates can be devoted to security-related tasks. Despite this, there are a vast number of proposals based on the use of classical hash functions, an assumption that is not realistic (at least at the present time). Furthermore, none of the published authentication protocols are resistant to active attacks. We try to address these two issues in this work by designing a new authentication protocol, secure against passive and active attacks, inspired by Shieh et al.'s protocol for smart-cards, but adapted to RFID systems. The original Shieh et al.'s scheme is considered one of the most secure and efficient protocols in the smart-card field. Because in this protocol tags should support a hash-function on-board, a new lightweight hash function, named *Tav-128*, is also proposed. A preliminary security analysis is shown, as well as a study on its hardware complexity, which concludes that its implementation is possible with around 2.6K gates.

Keywords: IUC, RFID, Security, Active-attacks, Authentication, Lightweight Hash functions

1 Introduction

One of the main problems that ubiquitous computing has to solve before its wide development is privacy [1]. In the RFID context, products labeled with insecure tags reveal sensitive information when queried by readers. Additionally, tags usually answer different queries with the same identifier. These predictable tag responses allow a third party to establish an association between tags and their owners. In addition to the previous threats, there are some other aspects that must be considered: eavesdropping, counterfeiting, physical attacks, active attacks, etc. To depth in all these matters we recommend the reading of [2–4] where surveys of the most important advances in RFID technology are presented.

Low-cost RFID tags are very computationally limited devices due to its severe price restriction (.05 - 0.1 €). Tags can only store hundreds of bits, and have 250-4K gates to implement security functions [5]. Even under these conditions, most of the proposed solutions in the literature are based on hash functions or

PRNGs [6–9]. From a theoretical point of view, these proposals have helped to increment the security level of RFID systems. However, none of these proposals are realistic. Note that for implementing traditional hash functions significantly more resources are needed. On the other hand, lightweight protocols can be fitted in low-cost RFID tags, because they only perform very simple operations. Nevertheless, none of the existing proposals are resistant to active attacks. In most of the cases, these kind of attacks are simply discarded as not applicable, which may be false in some real-life scenarios. Recently, Cui et al. have proposed the use of asymmetric cryptography to solve active attacks [10]. However, nowadays the usage of asymmetric cryptography, although being an active research field [11, 12] is not considered to be possible in low-cost RFID tags.

The kind of attacks applicable to RFID technologies are not much different to those that can happen in wireless, bluetooth, or smart-card systems. We have found interesting resemblances in the field of smart-card security, which is by now a consolidated technology. Since the pioneer work of Lamport (1981) where he proposed a remote authentication scheme, many researchers suggested alternative schemes improving the efficiency and security of remote authentication processes. Recently, Shieh et al. have proposed a very interesting scheme in their work entitled “Efficient remote mutual authentication and key agreement” [13]. This protocol is considered to be one of the most secure and efficient security protocols for smart-cards. Taking advantage of this work, we have updated their protocol to the characteristics of RFID systems. The resulting protocol is not only resistant to the standard passive attacks, such as privacy, tracking and eavesdropping, etc. but also to active attacks. As the protocol is based on the use of hash functions, we have also designed a new lightweight hash function, named *Tav-128*. A security and performance analysis of this new function is presented, showing its applicability to low-cost RFID tags.

The rest of the paper is organized as follows. In *Sect. 2*, Shieh et al.’s protocol is described. *Sect. 3* proposes a new protocol inspired in Shieh et al.’s scheme but adapted to RFID systems. A security analysis is presented in *Sect. 4*. A new lightweight hash function is proposed in *Sect. 5*, including a preliminary security and performance analysis. Finally, we draw some conclusions in *Sect. 6*.

2 Review of Shieh et al.’s scheme

The security of Shieh et al.’s scheme (2006) is based on the use of secure one-way hash functions (Merkle, 1989; NIST FIPS PUB 180, 1993; Rivest, 1992). Time stamps are used but no time-synchronization is required. The scheme consists in two phases: the registration phase, and the login and key agreement phase.

2.1 Registration phase

Assume an user U_i submits his identity ID_i and password PW_i to the server over a secure channel for registration. If the request is accepted, the server computes $R_i = h(ID_i \oplus x) \oplus PW_i$ and issues U_i a smart-card containing R_i and $h()$, where

(1) U_i	→ Server:	ID_i, T_u, MAC_u
(2) Server	→ U_i :	T_u, T_s, MAC_s
(3) U_i	→ Server:	T_s, MAC'_u
$a_i = h(ID_i \oplus x)$		$MAC_u = h(T_u a_i)$
$MAC_s = h(T_u T_s a'_i)$		$MAC'_u = h(T_s (a_i + 1))$

Fig. 1. Messages transmitted in Shieh's scheme

$h()$ is a one-way hash-function, x is the secret key maintained by the server, and the symbol " \oplus " denotes the exclusive-or operation.

2.2 Login and key agreement phase

Fig. 1 is an illustration of messages transmitted during the login and key agreement phase in Shieh's scheme. When user U_i wants to login to the server, he first inserts his smart-card into a card reader then inputs his identity ID_i and password PW_i . Next, the smart-card performs the followings steps:

1. Compute $a_i = R_i \oplus PW_i$.
2. Acquire current time stamp T_u , store T_u until the end of the session, and compute $MAC_u = h(T_u || a_i)$.
3. Send message (ID_i, T_u, MAC_u) to the server.

After receiving message (ID_i, T_u, MAC_u) from U_i , the server performs the following steps to assure the integrity of the message, answer to U_i , and challenge U_i to avoid replay attacks:

1. Check the freshness of T_u . If T_u has already appeared in a current execution session of user U_i , reject U_i 's login request and stop the session. Otherwise T_u is fresh.
2. Compute $a'_i = h(ID_i \oplus x)$, $MAC'_u = h(T_u || a'_i)$ and check whether MAC'_u is equal to the received MAC_u . If it is not, reject U_i 's login and stop the session.
3. Acquire current time stamp T_s . Store temporarily paired time stamps (T_u, T_s) and ID_i for freshness checking until the end of the session. Compute $MAC_s = h(T_u || T_s || a'_i)$ and session key $K_s = h((T_u || T_s) \oplus a'_i)$. Then, send the message (T_u, T_s, MAC_s) back to U_i .

On receiving the message (T_u, T_s, MAC_s) from the server, the smart-card performs the following steps to authenticate the server, achieves a session key agreement, and answers to the server.

1. Check if the received T_u is equal to the stored T_u to assure the freshness of the received message. If is not, report login failure to the user and stop the session.

2. Compute $MAC'_s = h(T_u || T_s || a_i)$ and check whether it is equal to the received MAC_s . If not, report login failure to the user and stop. Otherwise conclude that the responding party is the real server.
3. Compute $MAC''_u = h(T_s || a_i + 1)$ and session key $K_s = h((T_u || T_s) \oplus a_i)$, then send the message (T_s, MAC''_u) back to the server.

When the message (T_s, MAC''_u) from U_i is received, the server performs the following steps to authenticate U_i and achieve key agreement:

1. Check if the received T_s is equal to the stored T_s . If it fails reject U'_i login request and stop the session.
2. Compute $MAC'''_u = h(T_s || (a'_i + 1))$ and check whether this is equal to MAC''_u . If it is not, reject U_i 's login request and stop the session. Otherwise, U_i is a legal user and U_i 's login is permitted. At this moment, mutual authentication and session key agreement between U_i and the server are achieved.

3 Our scheme

In this section, a new protocol adapted to RFID systems and resistant to passive and active attacks (inspired in Shieh et al.'s protocol) is proposed. First, we will mention some peculiarities of RFID systems which should be considered in the new design. These will force changes in the protocol, which will be presented next.

In Shieh et al.'s protocol, when the user wants to login in the server “*he first inserts the card into a card-reader...*”. In a RFID system, tags (T) will be equivalent to smart-cards and readers to card-readers, respectively. Note RFID readers (R) are assumed to be connected to back-end databases (B) over a secure channel. Additionally, both devices have “non-limited” computing and storing capabilities. In the following, when we refer to a RFID reader an entity composed by a reader and a back-end database is considered.

However, there are significant differences between smart-card and RFID systems. RFID technology operates through the radio channel, so communication could be eavesdropped. Another particularity is the asymmetry of the communication channel, which allows monitorization of the forward channel (reader-to-tag) from a much longer distance than the backward channel (tag-to-reader). Smart-cards are usually tamper resistant devices, which is not the case of RFID tags. Furthermore, when the smart-card is inserted in the reader an user intervention is necessary, entering his identity and password. In RFID technology, however, interactions between tags and readers are automatic.

Taking into account all these considerations, Shieh et al.'s scheme has been adapted. Our proposed scheme consists on two phases: the registration phase, and the mutual authentication and index-pseudonym update phase. The following symbols have been used:

- | | |
|---|--------------------------------------|
| x_i : secret key maintained by the reader | N_z : random number generated by z |
| $h()$: secure one-way hash function | \oplus : exclusive-or operation |
| $ $: string concatenation operation | |

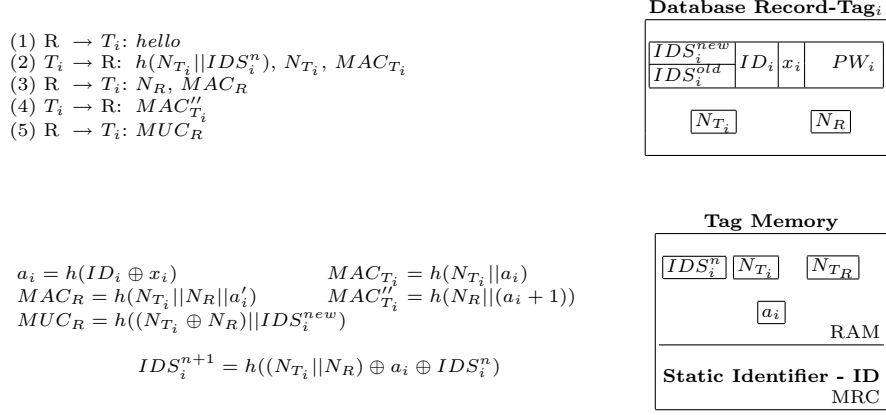


Fig. 2. Messages transmitted in our protocol

3.1 Registration phase

The user or holder of the tag submits his static identifier ID_i^1 and a freely chosen password PW_i to the reader over a secure channel for registration. If the request is accepted, the reader generates a random index-pseudonym IDS_i^0 and computes $a_i = h(ID_i \oplus x_i)$.¹ The tag will replace its identifier ID_i by IDS_i^0 and store a_i . The IDS_i^n will be used as searching-index of a database in which all the sensitive information (ID_i , x_i , PW_i) and the temporary data session (N_{T_i} , N_R) associated with each tag are stored. IDS_i^{new} and IDS_i^{old} are initially set to IDS_i^0 . The password PW_i will be used by the holder of the tag (over a secure channel) to temporarily deactivate the tag. In this case, a_i will be replaced by $R_i = a_i \oplus PW_i$.

3.2 Mutual authentication and index-pseudonym update

The messages interchanged in our scheme are shown in *Fig. 2*. First, the reader usually applies a probabilistic (ie. Aloha-based algorithm) or determinist (ie. Binary tree-walking protocol) collision avoidance protocol to singulate a tag out of many [9]. Upon singulation condition, the reader will send a “hello” message

¹ A 64-bit length identifier is compatible with all the encoding schemes (SGTIN, SSCC, GLN, etc) defined by EPCGlobal [14]. Due to this reason, we assume that tag static identifier (ID_i), and index-pseudonyms (IDS_i^n) are 64-bit length. Additionally, the secret key x_i is xored with ID_i to compute a_i , so x_i length is also set to 64-bits.

to the tag. To start the mutual authentication, the tag accomplishes the following steps:

1. Generate a random number N_{T_i} ², and store N_{T_i} temporarily until the end of the session.
2. Compute $h(N_{T_i}||IDS_i^n)$, and $MAC_{T_i} = h(N_{T_i}||a_i)$.
3. Send message $(h(N_{T_i}||IDS_i^n), N_{T_i}, MAC_{T_i})$ to the reader and wait for response.

Once the previous message is received, its integrity is checked and the reader answer includes a challenge to avoid replay attacks:

1. Check the newness of N_{T_i} . If N_{T_i} has already come out in a current mutual authentication, the protocol is stopped in this point. Otherwise N_{T_i} is fresh.
2. Compute $p' = h(N_{T_i}||IDS_i^{new})$ and $p'' = h(N_{T_i}||IDS_i^{old})$ and check whether any of the two values is equal to the received $h(N_{T_i}||IDS_i^n)$. The above procedure is repeated for each entry (row) in the database until a match is found. If not found, the protocol is stopped at this point.
3. Compute $a'_i = h(ID_i \oplus x_i)$, $MAC'_{T_i} = h(N_{T_i}||a'_i)$, and check if it is equal to MAC_{T_i} . If not, the protocol is stopped and a check over tag deactivation is taken by computing $R'_i = a'_i \oplus PW_i$, $MAC''_{T_i} = h(N_{T_i}||R'_i)$ and verifying if it is equal to MAC_{T_i} . A match will imply that the tag has been deactivated temporarily by its holder.
4. Acquire a fresh random number N_R ². For avoiding replay attacks, the pair (N_{T_i}, N_R) is stored until the end of the session.
5. Compute $MAC_R = h(N_{T_i}||N_R||a'_i)$. Then, send the message (N_R, MAC_R) back to the tag and wait for response.

After receiving the message (N_R, MAC_R) , the following steps are accomplished to authenticate the reader, achieve new material to update the index-pseudonym, and finally answer to the reader:

1. Compute $MAC'_R = h(N_{T_i}||N_R||a_i)$ and check if its value is equal to the received MAC_R . If not, stop the protocol at this point. Note that the newness of this message is guaranteed by N_{T_i} . For preventing loss of synchronization attacks, N_R is also stored in the tag.
2. Compute $MAC''_{T_i} = h(N_R||a_i + 1)$ and send it back to the reader.

When the message MAC''_{T_i} is received, the reader computes $MAC'''_{T_i} = h(N_R||a'_i + 1)$ and checks whether it is equal to MAC''_{T_i} . If not, the protocol is stopped. At this point, both the reader and the tag have mutually authenticated. Additionally, both possess two nonces (N_{T_i}, N_R) , which have been interchanged. Shieh et al. proposed using this fresh material to establish a session key agreement. In our case this material is employed to update the index-pseudonym. Obviously, the tag and reader have to be synchronized.

² Tags conforming with EPC Class-1 Gen-2 specification support a 16-bit PRNG [15]. We suggest that 32-bit PRNGs should be supported on low-cost RFID tags, as mentioned in [16, 17]. So, 32-bit length could be an adequate value to N_{T_i} and N_R .

The glib solution for the synchronization problem will be to update the index-pseudonym in the tag when message 4 is sent, and this updating will be performed in the reader when checking this message. Under this scenario an attacker (active attack) could intercept message 4 avoiding the update of the index-pseudonym in the reader with the consequently losing of synchronization. A naive solution will consist on assuming that after the end of the protocol, completion messages are sent between the involved entities. However, these messages could be also intercepted. Additionally, note that tags are much more constrained devices than readers. For this reason, a new message 5 has been added to the protocol (Message Update Code - MUC), and readers will have to store the old and the new index-pseudonym to prevent the interception of this message. To complete the protocol, the following steps are performed by the reader:

1. Store the current session index-pseudonym $IDS_i^{old} = IDS_i^{new}$ to avoid de-synchronization attacks.
2. Compute the new index-pseudonym $IDS_i^{new'} = h((N_{T_i} || N_R) \oplus a'_i \oplus IDS_i^{new})$.³
3. Compute $MUC_R = h((N_{T_i} \oplus N_R) || IDS_i^{new'})$ and send it to the tag, including the two nonces interchanged between reader and tag and the new index-pseudonym.

When the message MUC_R is received from reader, the tag accomplishes the following steps to verify a successfully index-pseudonym update has been performed in the reader:

1. Compute the potential-new index-pseudonym $IDS_i^{n+1} = h((N_{T_i} || N_R) \oplus a_i \oplus IDS_i^n)$.³
2. Compute $MUC_R'' = h((N_{T_i} \oplus N_R) || IDS_i^{n+1})$ and check whether MUC_R'' is equal to MUC_R . If this is the case, update the index-pseudonym.

4 Security analysis

The robustness of the proposed protocol against the main important attacks is analyzed in the following.

1. User Privacy

Tag ID_i must be kept secure to guarantee user's privacy. In order to protect it, both the tag's memory and the radio channel have been taken into account. In the registration phase, the static identifier ID_i and the password PW_i are submitted to the reader over a secure channel. To avoid radio access to the static identifier, ID_i is replaced by the hash of $ID_i \oplus x_i$. Note, x_i is a secret key only known by the reader. Additionally, and similarly to what happens in e-passports, we recommended the ID_i to be printed as a

³ If tags support on board the proposed *Tav*-128 hash function, a_i 's length will be fixed to 128-bits ($a_i = h(ID_i \oplus x_i)$). In this case, we suggest the following update equation: $IDS_i^{new'} = h((N_{T_i} || N_R) \oplus a'_i[0:63] \oplus a'_i[64:127] \oplus IDS_i^{new})$.

machine-readable code as illustrated in *Fig. 2*. In the radio channel, the value of IDS_i^n is protected by the use of a secure one-way hash function $h(\cdot)$. In the same way, a_i can not be derived from the messages authentication codes MAC_{T_i} , MAC_R and MAC''_{T_i} .

2. Location Privacy

The secure protection of tag information does not ensure location privacy. Constant answers would allow an attacker to identify each tag with its holder. To protect the index-pseudonym, only its hash is transmitted. As the index-pseudonym is not updated until the completion of the protocol, and the protocol may be accidentally or intentionally interrupted, the hash of the IDS_i concatenated with nonce N_{T_i} is really sent. Similarly, a_i is anonymized by means of the use of message authentication codes where a kind of challenge-response nonces are included. Finally, sending the message update code $MUC_R = h((N_{T_i} \oplus N_R) || IDS^{n+1})$, the new index-pseudonym is hidden. So, in order to avoid tracking, all the information is anonymized.

3. Data Integrity

Based on the use of a mutual authentication approach, our protocol guarantees data integrity between tag and reader. On the other hand, tag's memory is rewritable so modifications are possible. In this memory, both a_i and the index-pseudonym IDS_i^n are stored. If an attacker does succeed in modifying this part of the memory, the reader would not recognize the tag, having to carry out the registration phased again (see *Sect. 3.1*).

4. Mutual Authentication

Due to the fact that both tag and reader authenticate each other, by means of message authentication codes MAC_R and MAC''_{T_i} , mutual authentication is accomplished. These message authentication codes include a_i , a secret only shared between them, preventing any other to create correct MAC s, and in this way guaranteeing the legitimacy of each part. Therefore, it is infeasible for a fraudulent reader or tag to impersonate another entity.

5. Replay Attack

Our protocol is based on a challenge-response scheme, so replay attacks are prevented because challenges are different each time and long enough to prevent attacks based on storing them. In our scheme, any replay attack will not be able to correctly answer the challenges that form part of the protocol. In message 2, tag sends $(h(N_{T_i} || IDS_i^n), N_{T_i}, MAC_{T_i})$ where a nonce N_{T_i} is included. Therefore, the reader must include N_{T_i} in the answer message, so in message 3 the reader sends $(N_R, MAC_R = h(N_{T_i} || N_R || a'_i))$, including not only the response nonce N_{T_i} but also a challenge nonce N_R . Then, tag sends $MAC''_{T_i} = h(N_R || (a_i + 1))$ back, including N_R , to the reader. So, only legitimate parties (reader+tag) can send valid answers as challenge nonces are joined with the message authentication codes requiring the knowledge of a_i .

6. Forgery Resistance

All the sensitive information stored in the tag (IDS_i^n, a_i) is never sent in clear in the communication channel. In all cases, this information is concate-

nated with a nonce and hashed before passed on the channel. Therefore, the simple copy of information by eavesdropping is not useful to an adversary.

7. Active Attacks

- (a) Man-in-the-middle attack: If an attacker tries to impersonate a legitimate reader to obtain information from a tag, perhaps to be able to impersonate it in a future. This kind of attack is not feasible because all messages include a message authentication code, which requires the knowledge of the secret a_i shared only between the tag and the reader. In the previous scenario, the fraudulent reader will not be able to generate message 3, so the capture of the message 4 sent back by the tag will be a vain attempt. Moreover, in future sessions, a new challenge would be used by the reader preventing any advantage from knowing old messages.
- (b) Parallel session: Because of the asymmetric structure of the message authentication codes $MAC_{T_i} = h(N_{T_i} || a_i)$ and $MAC''_{T_i} = h(N_R || a_i + 1)$ this attack fails. Another important point is that both reader and tag store the session nonces, N_{T_i} and N_R .
- (c) Synchronization loss: The tag updates the index-pseudonym only when the message update code (MUC) is received. An attacker could interrupt this message, trying to de-synchronize reader and tag. To avoid this sort of attack, each time the reader updates the index-pseudonym, the old index-pseudonym is still maintained. Under the interception of the MUC from the reader, the tag will use the old index-pseudonym to build $h(IDS_i^n || N_{T_i})$. When the reader checks its integrity, it first will try with the new index-pseudonym, and if it fails, then he will try with the old index-pseudonym. Next, the rest of the protocol will be accomplished ensuring the recovery of a synchronization loss.

5 Hash-Function

Traditional cryptographic primitives exceed the capabilities of low-cost RFID tags. The required hardware complexity of these devices may be weighted up by its circuit area or the number of equivalent logic gates. At most, around 4K gates are assumed to be devoted to security-related task [5]. The best implementation of SHA-256 requires around 11K gates and 1120 clock cycles to performing a hash calculation on a 512-bit data block [18]. As the number of needed resources are quite higher than those of a low-cost RFID tag, it may seem natural to propose the use of another smaller hash functions. However, functions such as SHA-1 (8.1K gates, 1228 clock cycles) or MD5 (8.4K gates, 612 clock cycles) can not be fitted either in a tag [18]. Recently, some authors suggest the usage of a “universal hash function” [19]. Although this solution only needs around 1.7K gates, a deeper security analysis is needed and has not yet been accomplished. Furthermore, this function has only a 64-bit output, which does not guarantee an appropriate security level because finding collisions is a relatively easy task due to the birthday paradox (around 2^{32} operations). For this reason, we propose a 128-bit hash function named *Tav-128* that can be fitted in low-cost RFID tags and provides a suitable security level for most applications.

5.1 *Tav*-128 Security analysis

Some of the recent cryptanalytic attacks on many of the most important hash functions [20, 21] rely in the fact that these constructions generally use a very linear (LFSR-based) expansion algorithm. In order to avoid this, we have decided to make the expansion of the *Tav*-128 hash function (corresponding to algorithms C and D in the code shown in the *Appendix A*) highly nonlinear. As, on the other hand, the resulting function should be very efficient and lightweight both from the gate count and the throughput point of view, we have found these functions by evolving compositions of extremely light operands by means of genetic programming, as described in [22].

We have also tried to include a filter phase (corresponding to algorithms A and B in the *Appendix A*) in the input of the *Tav*-128 function, in order to avoid the attacker to have direct access to any bit of the internal state. Not having this possibility, some attacks that have been found on other cryptographic primitives in the past are precluded. So, decreasing the control the attacker has over the hash functions inputs significantly complicates his task.

An output length of 128 bits was found to be a reasonable compromise between speed and robustness to realistic attacks in the intended scenarios. Additionally, we propose the use of eight rounds in the internal loop (*r2* parameter) for having an adequate security margin, though we have found that even with six rounds (which will significantly improve its performance) the overall scheme seems to be secure.

We have performed an additional security analysis of *Tav*-128, consisting on examining the statistical properties of its output over a very low entropy input. Specifically, 2^{25} 32-bit inputs have been generated by means of an incremental counter (x , $x+1$, $x+2$, etc). After randomly initializing (with values obtained from <http://randomnumber.org>) the internal state and the accumulated hash $a0$ value, we compute the output of *Tav*-128 for each counter value input ($Tav(x)$, $Tav(x+1)$, $Tav(x+2)$, etc). The resulting hashes have been analyzed with two well-known suites of randomness tests, namely ENT [23] and DIEHARD [24]. The results are presented in *Tables 1* and *2* (see *Appendix A*). *Tav*-128 also passed the very demanding -because it is oriented to cryptographic applications- NIST [25] statistical battery. We have computed 100 p-values for each test, being all the results compatible with a uniform $U(0, 1)$. The whole report is available in <http://163.117.149.137/tav/> due to the huge amount of p-values generated.

Authors acknowledge that successfully passing these statistical batteries, even over a very low-entropy input, does not prove security, but we believe that it points out the nonexistence of trivial weaknesses.

5.2 Hardware Complexity

One of the most relevant aspects considered in the design of *Tav*-128 was the sort of operations that can be employed. As tags are very restricted computationally, only simple operations have been used. For example, multiplication has been ruled out due to its high cost [26]. Concretely, the following operators have

been finally used: right shifts, bitwise xor, and addition mod 2^{32} . The necessary architecture to implement *Tav-128* can be divided in two main blocks:

- **Memory blocks.** All the used variables are stored in this part: state (128-bits), accumulated hash *a0* (32-bits), internal variables *h0* (32-bits) and *h1* (32-bits), and the input *a1* (32-bits).
- **Arithmetic logic Unit.** In this unit the addition mod 2^{32} and the bitwise xor operation are implemented. As the *h0* and *h1* functions consist of three or more components, an auxiliary register to store the intermediate results is necessary.

Although we have not implemented *Tav-128* in hardware, an overestimation of its gate counting can be easily obtained. The function bitwise xor requires 32 logic gates as we are operating with 32-bit variables. For implementing the add with carry circuit, a parallel architecture is proposed. Six logic gates are needed for each bit added in parallel.⁴ The registers will be implemented by means of flip-flops. A gate count of 8 has been chosen for implementing a flip-flop as in [27]. So, 2304 logic gates are necessary to store the memory block and the auxiliary register. Additionally, around 50 extra logic gates are employed to control the internal state of the hash function. Therefore, 2578 logic gates are needed for implementing *Tav-128*.

Another key aspect to consider is throughput. We reckon that 1568 clock cycles are needed for executing one *Tav-128* hash. Due to the fact that low-cost RFID tags imply serious powers restrictions, we assume that the clock frequency is set to 100 KHz. Under this conditions, the throughput obtained by a tag that would have on-chip *Tav-128* will be around 65 hashes/sec. It is generally accepted that at least between 50-100 tags should be authenticated per second [28]. In order words, a tag may use up at the most 2000 clock cycles (@100Khz) to answer a reader. In some applications 65 hashes/sec may not be enough, so we have analyzed how to increment the speed of *Tav-128*. In the initial proposed scheme (see *Appendix A*), we have a parameter (*r2*), which fits the number of rounds computed in the *C* and *D* algorithms. This parameter has been initially fixed to eight rounds in order to guarantee a high avalanche effect. After accomplishing a deeper study, we have determined that *r2* may be reduced to six rounds. So, the speed of the tag will be incremented in a 25% or in other words, the tag may compute around 80 hashes/sec. Note that for non-high speed demanding applications, we recommend to fix *r2* to eight rounds.

6 Conclusions

Since 2002, there has been a great number of publications concerned with the security of RFID technology. In the majority of those proposals, the security objectives are focused on privacy, tracking, counterfeiting, etc. All this kind of attacks are passive, but active attacks can not be ruled out in many scenarios.

⁴ $S = A \oplus [B \oplus C_{ENT}] \quad C_{SAL} = BC_{ENT} + AC_{ENT} + AB$

A new protocol not only resistant to standard passive attacks but also resistant to active attacks is proposed. Another interesting property is that tags can be temporally deactivated without data loss. Instead of beginning from scratch, we have tried to avoid past errors in the designing of our protocol. RFID technology has similarities with other technologies such as wireless, bluetooth, smart-card, etc. Indeed, we focused our attention to smart-card, which is a mature technology. Concretely, we spotlight on remote authentication protocols, which started to be developed in 1980. During years many researchers have been working in order to propose more secure and efficient schemes. Recently, Shieh et al. have proposed a new scheme that can be considered one of the most secure and efficient protocols. For this reason we decide to propose a new protocol for RFID systems inspired in Shieh et al.'s protocol.

The proposed protocol is based on the use of a secure hash function. As traditional cryptographic primitives such as SHA-256 or MD5 lie well beyond the capabilities of low-cost RFID tags, a new hash function (*Tav-128*) is proposed. *Tav-128* can be implemented with only around 2.6K gates, and 1568 cycles (1248 if r_2 parameter is set to six). Therefore, *Tav-128* can be fitted in a real low-cost RFID tags. Although further security analysis of the new hash function is needed, this preliminary analysis seems to point out that it gives an adequate security level for the intending application (mutual authentication of low-cost tags). To conclude, although this hash function constitutes a great advance, as a future work we plan to design a new version where the number of processing cycles was reduced without incrementing the number of logical gates.

References

1. Weiser, M.: The computer for the 21st century. *Scientific American* **265**(3) (1991) 94–104
2. Juels, A.: RFID security and privacy: A research survey. Manuscript (2005)
3. Peris-Lopez, P., Hernandez-Castro, J.C., Estevez-Tapiador, J., Ribagorda, A.: RFID systems: A survey on security threats and proposed solutions. In: Proc. of PWC06. Volume 4217 of LNCS. (2006) 159–170
4. Piramuthu, S.: Protocols for RFID tag/reader authentication. *Decision Support Systems* **43**(3) (2007) 897–914
5. Ranasinghe, D., Engels, D., Cole, P.: Low-cost RFID systems: Confronting security and privacy. In: Auto-ID Labs Research Workshop. (2004)
6. Ohkubo, M., Suzuki, K., Kinoshita, S.: Cryptographic approach to “privacy-friendly” tags. In: Proc. of RFID Privacy Workshop. (2003)
7. Molnar, D., Wagner, D.: Privacy and security in library RFID: Issues, practices, and architectures. In: Proc. of ACM CCS’04. (2004) 210–219
8. Rhee, K., Kwak, J., Kim, S., Won, D.: Challenge-response based RFID authentication protocol for distributed database environment. In: Proc. of SPC’05. Volume 3450 of LNCS. (2005) 70–84
9. Sarma, S., Weis, S., Engels, D.: RFID systems and security and privacy implications. In: Proc. of CHES’02. Volume 2523., LNCS (2002) 454–470
10. Cui, Y., Kobara, K., Matsuura, K., Imai, H.: Lightweight asymmetric privacy-preserving authentication protocols secure against active attack. In: Proc. of Per-Sec’07. (2007)

11. Batina, L., Guajardo, J., Kerins, T., Mentens, N., Tuyls, P., Verbauwhede, I.: Public key cryptography for RFID-tags. In: Proc. of PerSec'07. (2007)
12. McLoone, M., Robshaw, M.: Public key cryptography and RFID tags. In: Proc. of CT-RSA'07. LNCS (2007)
13. Shieh, W.G., Wang, J.M.: Efficient remote mutual authentication and key agreement. *Computers & Security* **25**(1) (2006) 72–77
14. EPCGlobal: EPC Generation-1 Tag Data Standards version 1.1. <http://www.epcglobalinc.org/standards/>
15. EPCGlobal: Class-1 Generation-2 UHF Air Interface Protocol Standard version 1.0.9: "Gen 2". <http://www.epcglobalinc.org/standards/>
16. Nguyen Duc, D., Park, J., Lee, H., K., K.: Enhancing security of EPCglobal gen-2 RFID tag against traceability and cloning. In: Proc. of Symposium on Cryptography and Information Security, Hiroshima, Japan (2006)
17. Kim, K.H., Choi, E.Y., Lee, S.M., Lee, D.H.: Secure EPCglobal Class-1 Gen-2 RFID system against security and privacy problems. In: Proc. of OTM-IS'06. Volume 4277 of LNC. (2006) 362–371
18. Feldhofer, M., Rechberger, C.: A case against currently used hash functions in RFID protocols. Proc. of RFIDSec'06 (2006)
19. Yksel, K., Kaps, J., Sunar, B.: Universal hash functions for emerging ultra-low-power networks. In: Proc. of CNDS'04. (2004)
20. Wang, X., Feng, D., Lai, X., Yu, H.: Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. *Cryptology ePrint Archive*, Report 2004/199 (2004)
21. Wang, X., Lisa Yin, Y., Yu, H.: Finding collisions in the full SHA-1. In: Proc. of CRYPTO'05. (2005) 17–36
22. Hernandez-Castro, J., Estevez-Tapiador, J., Ribagorda-Garnacho, A., Ramos-Alvarez, B.: Wheedham: An automatically designed block cipher by means of genetic programming. In: Proc. of CEC'06. (2006) 192–199
23. Walker, J.: Randomness Battery, <http://www.fourmilab.ch/random/>. (1998)
24. Marsaglia, G., Tsang, W.: Some difficult-to-pass tests of randomness. *Journal of Statistical Software* **Volume 7, Issue 3** (2002) 37–51
25. Suresh, C., Charanjit, J., Rao, J., Rohatgi, P.: A cautionary note regarding evaluation of AES candidates on smart-cards. In: Second Advanced Encryption Standard (AES) Candidate Conference. (1999)
26. Lohmann, T., Schneider, M., Ruland, C.: Analysis of power constraints for cryptographic algorithms in mid-cost RFID tags. In: Proc. of CARDIS'06. Volume 3928 of LNCS. (2006) 278–288
27. Hell, M., Johansson, T., Meier, W.: Grain - a stream cipher for constrained environments. Proc. of RFIDSec'05 (2005)
28. Roberts, C.: Radio frequency identification (RFID). *Computers and Security* **25**(1) (2006) 18–26

APPENDIX A

A Hash Tav-128 Ansi C & Statistical Tests

```

/*****
Process the input a1 modifying the
accumulated hash a0 and the state
*****/
void tav(unsigned long *state, unsigned long
*a0, unsigned long *a1)

{
unsigned long h0,h1;
int i,j,r1,r2,nstate;

/* Initialization */
r1=32; r2=8; nstate=4;
h0=*a0; h1=*a0;

/* A - Function */
for(i=0;i<r1;i++){h0=(h0<<1)+((h0+(*a1))>>1);}
/* B - Function */
for(i=0;i<r1;i++){h1=(h1>>1)+(h1<<1)+h1+(*a1);}

/* C and D - Function */
for(j=0;j<nstate;j++) {
for(i=0;i<r2;i++)
{
/* C - Function */
h0^=(h1+h0)>>3;
h0=(((h0>>2)+h0)>>2)+(h0<<3)
+(h0<<1)^0x736B83DC;
/* D - Function */
h1^=(h1~h0)>>1;
h1=(h1>>4)+(h1>>3)+(h1<<3)+h1;
} // round-r2
state[j]+=h0;
state[j]^=h1;
} // state

/* a0 updating */
*a0=h1+h0;
}

/*****
Initialization of the state and a0 with
random values obtained from www.random.org
*****/
void init_state(unsigned long *state, unsigned
long *a0)

{
state[0]=0xa92be51d;
state[1]=0xba9b1ef0;
state[2]=0xc234d75a;
state[3]=0x845c2e03;
a0[0]=0x768c7e74;
}

```

Table 1. Results obtained with ENT

Test	Tav-128
Entropy	7.999999 bits/byte
Compression Rate	0%
χ^2 Statistic	269.73 (50%)
Arithmetic Mean	127.4993
Monte Carlo π estimation	3.14178848 (0.01%)
Serial correlation coefficient	-0.000073

Table 2. Results obtained with the Diehard suite

Test	Tav-128 p-value
Birthday Spacings	0.725
	0.868
GCD	0.229
	0.138
Gorilla	0.779
Overlapping Permutations	0.823
	0.849
	0.349
	0.897
	0.939
Ranks of 31×31 and 32×32 Matrices	0.556
	0.241
Ranks of 6×8 Matrices	0.315
Monkey Tests on 20-bit Words	0.312
Monkey Test OPSO, OQSO, DNA	OK
Count the 1's in a Stream of Bytes	0.473
Count the 1's in Specific Bytes	OK
Parking Lot Test	0.235
Minimum Distance Test	0.580
Random Spheres Test	0.912
The Squeeze Test	0.487
Overlapping Sums Test	0.106
Runs Up and Down Test	0.147
The Craps Test	0.3211
	0.067
	0.775
	0.261
Overall KS p-value	0.826