From Weaknesses to Secret Disclosure in a Recent Ultra-Lightweight RFID Authentication Protocol[*]

Paolo D'Arco and Alfredo De Santis

Dipartimento di Informatica ed Applicazioni
Università di Salerno, 84084 Fisciano (SA), Italy
e-mail: {paodar, ads}@dia.unisa.it

November 5, 2008

Abstract

A recent research trend, motivated by the massive deployment of RFID technology, looks at cryptographic protocols for securing communication between entities in which al least one of the parties has very limited computing capabilities.

In this paper we focus our attention on **SASI**, a new Ultra-Lightweight RFID Authentication Protocol, designed for providing **S**trong **A**uthentication and **S**trong **I**ntegrity. The protocol, suitable for passive Tags with limited computational power and storage, involves simple bitwise operations like *and*, *or*, exclusive *or*, modular addition, and cyclic shift operations. It is efficient, fits the hardware constraints, and can be seen as an example of the above research trend.

We start by showing some weaknesses in the protocol and, then, we describe how such weaknesses, through a sequence of simple steps, can be used to compute in an efficient way *all* secret data used for the authentication process. More precisely, we describe three attacks:

- A *de-synchronisation* attack, through which an adversary can break the synchronisation between the RFID Reader and the Tag.
- An *identity disclosure* attack, through which an adversary can compute the identity of the Tag.
- A *full disclosure* attack, which enables an adversary to retrieve all secret data stored in the Tag.

Then we present some experimental results we have obtained by running several tests on an implementation of the protocol, in order to evaluate the performance of the proposed attacks. The results confirm that the attacks are effective and efficient.

1 Introduction

RFID Technology. Radio Frequency Identification (RFID, for short) is a rapidly growing technology enabling automatic objects identification. Each object is labeled with a tiny integrated circuit equipped with a radio antenna, called *Tag*, whose *information content* can be received by another device, called *Reader*, without physical contact, at a distance of several meters.

RFID Tags can perform computations. They are usually divided in *passive* Tags and in *active* Tags. The first ones do not have a power source. They receive energy for computation from the

[*]A preliminary version of this paper appeared in the Proceedings of AFRICACRYPT 2008, Lecture Notes in Computer Science, Vol. 5023, pp. 27-39, 2008.

readers and can perform very simple operations. The second ones are powered by small batteries and are capable of performing more significant and computational heavy operations.

As Ari Juels [9] has recently pointed out, the RFID technology *"in essence ... is a form of computer vision ... RFID has an advantage over even the most acute eyes and brain: it is in fact a form of X-ray vision...RFID is poised to become one of the sensory organs of our computing networks"*.

An important security concern associated to the RFID technology is the *privacy* of the Tag content. Indeed, it is pretty much easy for anybody with technical skills to set up a device for reading the Tag content. Neverthless, to preserve user privacy, only authorised RFID Readers should be enabled to access the Tag content. At the same time, legal RFID Readers would like to be sure that the Tags they are reading are authentic and have not been counterfeit.

An authentication protocol, which grants access to the Tag content only to a legitimate Reader and, at the same time, guarantees the Reader of the identity of the Tag, is therefore required.

Based on the computational cost and the operations supported on Tags, authentication protocols can be divided in classes. Using the terminology of [2], the *full-fledged* class refers to protocols requiring support on Tags for conventional cryptographic functions like symmetric encryption, hashing, or even public key cryptography. The *simple* class refers to protocols requiring random number generation and hashing. The *lightweight* class refers to protocols which require random number generation and simple checksum functions. The *Ultra-Lightweight* class refers to protocols which only involve simple bitwise operations, like *and*, *or*, exclusive *or*, and modular addition.

State of Art. The interested reader can find an overview of the applications of the RFID technology and of the main security issues in [10]. Moreover, [1] contains references almost to the full body of research papers dealing with RFID technology and its challenges.

Concerning with the design of an authentication protocol for RFID Tags, a deeply studied approach is the one provided by the $HB+$ protocol [11], which builds on the former HB protocol [4], introduced to efficiently authenticate a human to a computer. The security of these protocols is based on the difficulty of solving the *learning parity with noise* (LPN) problem [4]. Unfortunately, the authors of [7] showed that $HB+$ is not resistant against certain active attacks. Since then, several variants of $HB+$ have been proposed in the literature but almost all of them present some problems [6]. The lastest version of an HB-like scheme, which is secure in an adversarial model comprising passive and certain active attacks, has been recently proposed in [5].

Another recent and very promising approach to designing an authentication protocol for RFID Tags can be found in [19]. In this paper, a "light" hash function which can be used in RFID authentication has been described. The security of such an hash function is related to the security of the Rabin public key scheme. The idea is to compute an excellent numerical approximation for a short window of bits in the middle of the ciphertext produced by the Rabin encryption function which uses a modulus of a particular form, in such a way that computing these bits for an adversary is as hard as breaking the full Rabin scheme.

On the other hand, a few lightweight and ultra-lightweight authentication protocols have appeared in the literature during the last two years. For example, a series of ultra-lightweight authentication protocols involving only bitwise operations and modular addition have been proposed in [15, 16, 17]. Unfortunately, after few months, the vulnerabilities of these protocols have been showed [13, 12, 3].

Our Contribution. We focus our attention on a new ultra-lightweight authentication protocol, recently proposed in [2], to provide strong authentication and strong integrity data protection. We identify three attacks, namely, a *de-synchronisation* attack, through which an adversary can break the synchronisation between the RFID Reader and the Tag, an *identity disclosure* attack, through which an adversary can compute the identity of the Tag, and a *full disclosure* attack, which enables an adversary to retrieve all secret data stored in the Tag. The attacks are effective and efficient.

2 The Authentication Protocol

Let us focus on the protocol proposed by Chien in [2] and on its claimed security properties.

2.1 Description of the protocol

Three entities are involved: a Tag, a Reader and a Backend Server. The channel between the Reader and the Backend Server is assumed to be secure, but the channel between the Reader and the Tag is susceptible to all the possible attacks.

Each Tag has a *static identifier*, ID, a *pseudonym*, IDS, and *two keys*, K_1 and K_2. All of them are 96-bit strings. A string is represented as a sequence $X[95]\ldots X[0]$, from the most significant bit to the least significant bit. The pseudonym and the keys are shared with the Backend Server which, for each Tag with static identifier ID, stores in a table the tuple (IDS, K_1, K_2). After each successfull execution of the authentication protocol, the Tag and the Backend Server *update* such values.

The authentication protocol is a four-round protocol. To simplify the description we do not introduce explicitely the Backend Server and will say that the Reader performs some computations. However, the Reader just forwards the values received from the Tag to the Backend Server and gets back the output of the computation the Backend Server performs.

The computations involve the following operations: $\oplus$ (bitwise exclusive or), $\vee$ (bitwise or), $+ \bmod 2^{96}$, and $Rot(x,y)$, where x and y are two 96-bit values, and the $Rot(\cdot,\cdot)$ operator shifts to the left in a cyclic way x by $z = f(y)$ positions[1].

Let us look at Figure 1. The Reader starts the authentication protocol by sending an **Hello** message to the Tag. The Tag replies with the pseudonym **IDS**. Then, the Reader chooses, uniformly at random, two 96-bit random values n_1 and n_2, computes

$$\begin{aligned}
&\mathbf{A} = \mathbf{IDS} \oplus K_1 \oplus n_1\\
&\mathbf{B} = (\mathbf{IDS} \vee K_2) + n_2\\
&\overline{K_1} = Rot(K_1 \oplus n_2, K_1)\\
&\overline{K_2} = Rot(K_2 \oplus n_1, K_2)\\
&\mathbf{C} = (K_1 \oplus \overline{K_2}) + (\overline{K_1} \oplus K_2),
\end{aligned}$$

and sends to the Tag the string $\mathbf{A}||\mathbf{B}||\mathbf{C}$, the concatenation of $\mathbf{A}, \mathbf{B}$ and $\mathbf{C}$. The Tag, upon receiving $\mathbf{A}||\mathbf{B}||\mathbf{C}$, extract n_1 from $\mathbf{A}$, n_2 from $\mathbf{B}$, computes its own values

$$\begin{aligned}
&\overline{K_1} = Rot(K_1 \oplus n_2, K_1)\\
&\overline{K_2} = Rot(K_2 \oplus n_1, K_2)\\
&C = (K_1 \oplus \overline{K_2}) + (\overline{K_1} \oplus K_2),
\end{aligned}$$

and verifies whether $C = \mathbf{C}$. If the equality holds, i.e., the computed value is equal to the received value, then the Tag computes and sends to the Reader the value

$$\mathbf{D} = (\overline{K_2} + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1})$$

and updates its pseudonym and secret keys. Similarly, the Reader, once $\mathbf{D}$ has been received, computes his own value

$$D = (\overline{K_2} + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1}),$$

checks whether $D = \mathbf{D}$, and if the equality holds, updates the pseudonym and the keys shared with the Tag.

The pseudonym and the keys are updated as follows: The Tag sets

$$\begin{aligned}
&IDS_{old} = IDS, \qquad K_{1,old} = K_1, \quad K_{2,old} = K_2,\\
&IDS = (IDS_{old} + ID) \oplus (n_2 \oplus \overline{K_1}), \qquad K_1 = \overline{K_1}, \quad K_2 = \overline{K_2},
\end{aligned}$$

[1]Notice that the author of [2] provided no specification on how to actually compute $Rot(\cdot,\cdot)$. Hence, in analysing the protocol, we considered $z = y \bmod 96$, since this operation has been used before in the literature with this meaning. Later on, in [18], the authors stated that, through a personal communication, they found out that the amount of the rotation z should be computed as the Hamming weight (i.e., z is the number of bits equal to 1) of the string y. However, it is immediate to see that the attacks we describe work for *any* possible specification of z as a function of y.

while the Reader sets

$$\begin{aligned}&IDS_{old} = IDS\\ &IDS = (IDS_{old} + ID) \oplus (n_2 \oplus \overline{K_1})\\ &K_1 = \overline{K_1}, \quad K_2 = \overline{K_2}.\end{aligned}$$

The Reader stores the new tuple (IDS, K_1, K_2). The Tag stores the two tuples, (IDS, K_1, K_2) and $(IDS_{old}, K_{1,old}, K_{2,old})$, because it might happen that the Tag updates the pseudonym and the keys, while the Server does not. Such an event for example might occur if a simple communication fault does not permit the Reader to get the value **D**, sent by the Tag during the 4-th round of the authentication protocol. The old tuple is used as follows: any time the Reader gets **IDS** from the Tag, the Reader/Backend Server looks for a tuple (IDS, K_1, K_2). If no entry is found, the Reader sends another **Hello** message to the Tag and the Tag replies with $\mathbf{IDS_{old}}$. Hence, even if the Reader has not updated the tuple, the authentication protocol can be run by using the old one, i.e., $(IDS_{old}, K_{1,old}, K_{2,old})$. Of course, if no match is found also at the second interaction, the protocol fails.

Tag Identification

	Reader		*Tag*
1.		$\xrightarrow{\textbf{Hello}}$	
2.		$\xleftarrow{\textbf{IDS}}$	
3.	Choose n_1, n_2 $\mathbf{A} = \mathbf{IDS} \oplus K_1 \oplus n_1$ $\mathbf{B} = (\mathbf{IDS} \vee K_2) + n_2$ $\overline{K_1} = Rot(K_1 \oplus n_2, K_1)$ $\overline{K_2} = Rot(K_2 \oplus n_1, K_2)$ $\mathbf{C} = (K_1 \oplus \overline{K_2}) + (\overline{K_1} \oplus K_2)$		
		$\xrightarrow{\mathbf{A}\|\|\mathbf{B}\|\|\mathbf{C}}$	
			Extract n_1, n_2 $\overline{K_1} = Rot(K_1 \oplus n_2, K_1)$ $\overline{K_2} = Rot(K_2 \oplus n_1, K_2)$ $C = (K_1 \oplus \overline{K_2}) + (\overline{K_1} \oplus K_2)$ if $C = \mathbf{C}$ then accept and send $\mathbf{D} = (\overline{K_2} + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1})$
		$\xleftarrow{\mathbf{D}}$	
4.	$D = (\overline{K_2} + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1})$ if $D = \mathbf{D}$ then accept		

Pseudonym and Key Update

Reader	*Tag*
	$IDS_{old} = IDS, K_{1,old} = K_1, K_{2,old} = K_2$ $IDS = (IDS_{old} + ID) \oplus (n_2 \oplus \overline{K_1})$ $K_1 = \overline{K_1}, K_2 = \overline{K_2}$
$IDS_{old} = IDS$ $IDS = (IDS_{old} + ID) \oplus (n_2 \oplus \overline{K_1})$ $K_1 = \overline{K_1}, K_2 = \overline{K_2}$	

Figure 1: SASI: Tag Identification - Pseudonym and Key Update.

2.2 Security Properties

The protocol is claimed to enjoy several security properties. In the following we briefly state such properties and, using the wording of [2], we point out the reasons for which such security properties should be guaranteed.

1. Mutual authentication and data integrity. The Tag and the Reader can authenticate each other, because *only the genuine Reader who has the keys* K_1 *and* K_2 *can generate the consistent values* $\mathbf{A}||\mathbf{B}||\mathbf{C}$, *and only the genuine Tag who has the secret keys can derive the random numbers* n_1 *and* n_2, *and then generate the response* $\mathbf{D}$.

2. Tag anonymity and resistance to tracking. The pseudonyms of each Tag is updated per successful authentication, and the update operation involves random numbers. So the *successive pseudonyms from the same Tag look random, and the attacker cannot identify the identity of the Tag and cannot track the Tag.*

3. Data confidentiality. The calculation of each value of $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}$ involves at least two secret values (including the keys and the random numbers); so, *the static identifier and the secret values are well protected from the eavesdroppers.*

4. Forward security. The forward security property means to protect the past communications from a Tag even assuming the Tag be compromised some day. [...]If an attacker compromises a Tag and acquires the two entries of (IDS, K_1, K_2) some day, the attacker still *cannot infer the previous secret data and keys of the same Tag, because each of the updating equations and the calculations of* $\mathbf{A}||\mathbf{B}||\mathbf{C}$, *and* $\mathbf{D}$ *involves at least two random values.* So, the attacher cannot compromise the past communications from the same Tag.

5. Explicit key confirmation and resistance to de-synchronization attack. [...] The authenticity and the integrity of random values n_1, n_2 are ensured and the potential next keys $\overline{K}_1, \overline{K}_2$ are explicitely confirmed, *because the calculations of* $\mathbf{C}$ *and* $\mathbf{D}$ *explicitely involve these values.* So, the attacker cannot change the data without being noticed. [...] The attacker can intercept $\mathbf{D}$ sent by the Tag to make the Tag updates its local data while the Reader does not.[...] This cannot cause trouble to the scheme, *because the Tag keeps two entries of secret data* [...] *and can still authenticate with the Reader using the old value for this situation.*

6. Resistance to replay attack. The attacker may replay the response $\mathbf{D}$ from the Tag. However, the Reader will find the invalidity of the replay value, *because the challenge random numbers* n_1 *and* n_2 *from the Reader are different and independent each session.* Another replay scenario is: an attacker may intercept the data $\mathbf{D}$ in one session, and then replay an old message $\mathbf{A}||\mathbf{B}||\mathbf{C}$ (corresponding to an old $\mathbf{IDS}$) from the Reader. But, this scenario *will not change any internal state of the Tag, and the attacker gains no secret information nor de-synchronise the Reader and the Tag.*

7. Resistance to man-in-the-middle attack. The man-in-the-middle attack does not work because [...] *any modification to the values* $\mathbf{A}$ *and* $\mathbf{B}$ *will cause, from the attacker's point of view, unpredictable changes on the values* $\mathbf{C}$ *and* $\mathbf{D}$, *which makes the attacker hard to change the data without being noticed.*

8. Resistance to disclosure attack. [...] an attacker can slightly modify the challenge from the Reader and then infer partial information from the response of the Tag. However, the attack does not work [...] *because any slightly modification on the trasmission will be detected.*

Unfortunately, as we will show in the following sections, even if the arguments to support each of the security property seem reasonable, an adversary can efficiently break the authentication scheme in such a way that *none of them* is preserved.

3 De-synchronisation

We start by proposing, in this section, a de-synchronisation attack.

Let *Adv* be an adversary who controls the channel between the Reader and the Tag. *Adv* might simply look at and store the messages exchanged between the parties before forwarding them correctly. Such a behaviour models *passive* attacks. As well as, *Adv* might intercept/delay/inject/modify messages as he likes. Such a behaviour models *active* attacks.

In our attack, *Adv*, in a first stage just looks at an honest execution of the authentication protocol and stores the messages **Hello**, **IDS**, $\mathbf{A}||\mathbf{B}||\mathbf{C}$ and $\mathbf{D}$ the parties send to each other.

Then, in a second stage, *Adv* interacts with the Tag. Roughly speaking, *Adv* resets the Tag to the state in which the Tag was at the time of the interaction with the Reader and, then, by using the transcript of the execution of the authentication protocol, induces the Tag to accept a new sequence $\mathbf{A}'||\mathbf{B}'||\mathbf{C}'$. Such a sequence drives the Tag to overwrite the new tuple $(\mathbf{IDS}, K_1, K_2)$, computed at the end of the honest execution. If *Adv* succeeds, then Tag and Reader are de-synchonised. The new sequence $\mathbf{A}'||\mathbf{B}'||\mathbf{C}'$ is constructed by computing $\mathbf{C}'$ as a modification of $\mathbf{C}$ and by looking for an appropriately chosen $\mathbf{A}'$, obtained by flipping a single bit of $\mathbf{A}$. The value $\mathbf{B}$ stays the same. The attack is described in Figure 2.

Adv's Computation. After looking the values **IDS**, $\mathbf{A}||\mathbf{B}||\mathbf{C}$, and $\mathbf{D}$, it performs the following steps:

1. Computes $\mathbf{C}^{'} = \mathbf{C} + 2^0$ and sets $j = 0$.
2. Sends the **Hello** message to the Tag and gets back the (new) **IDS**, computed *at the end* of the execution Reader-Tag of the authentication protocol. Indeed, the Tag has updated $(\mathbf{IDS}, K_1, K_2)$.
3. Sends again the **Hello** message to the Tag and gets back the (old) **IDS**, the one used *during* the execution Reader-Tag of the authentication protocol.
4. Computes $\mathbf{A}'$ by flipping the j-th bit of $\mathbf{A}$ and sends to the Tag $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$.
5. If the Tag accepts and replies with $\mathbf{D}'$, the attack has succeeded and *Adv* terminates. Otherwise, if $j < 96$ then *Adv* sets $j = j + 1$ and repeats from step 2., else *Adv* sets $\mathbf{C}^{'} = \mathbf{C} - 2^0$ and $j = 0$ and repeats from step 2.

Figure 2: SASI: De-synchronisation Attack.

As we will show in a while, the procedure always halts, i.e., there is a j for which the Tag accepts the message $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$, where $\mathbf{C}^{'}$ is computed either as $\mathbf{C}^{'} = \mathbf{C} + 2^0$ or as $\mathbf{C}^{'} = \mathbf{C} - 2^0$. Moreover, it succedes on average after 96.5 interactions. When the message has been accepted, the Tag updates the pseudonym and the keys (IDS, K_1, K_2), while the old tuple $(IDS_{old}, K_{1,old}, K_{2,old})$, used in the interaction with *Adv* stays the same. At this point, Reader and Tag have been desynchronized. The tuple held by the Reader has been overwritten in the Tag's memory. Hence, they do not share a tuple anymore!

Why does the attack work? Notice that, by definition

$$\mathbf{A} = \mathbf{IDS} \oplus K_1 \oplus n_1.$$

By flipping a bit in $\mathbf{A}$, *Adv* implicitly flips a bit of n_1. On the other hand, n_1 is used to compute

$$\overline{K_2} = Rot(K_2 \oplus n_1, K_2).$$

Hence, by flipping a bit of n_1, *Adv* flips a bit of $\overline{K_2}$, but he *does not know* which one, since K_2 is unknown. Moreover, it is easy to see that, given two different positions, i and j, in n_1, by flipping the corresponding bits, the bits flipped in $\overline{K_2}$ lie in two different positions i' and j'. In other words, any time *Adv* flips a bit in n_1, he flips a bit in a different position of $\overline{K_2}$. Since,

$$\mathbf{C} = (K_1 \oplus \overline{K_2}) + (\overline{K_1} \oplus K_2),$$

and the values of $K_1, \overline{K_1}$, and K_2 stay the same, *Adv*, by flipping a bit of n_1, changes the value of $(K_1 \oplus \overline{K_2})$. If this value is modified in the i-th position, which goes from 1 to 0, then $\mathbf{C}^{'} = \mathbf{C} - 2^i$. On the other hand, if it changes from 0 to 1, then $\mathbf{C}^{'} = \mathbf{C} + 2^i$. For easiness of presentation and due to the role that the first position of the strings plays in the identity disclosure attack, we have presented the desynchronisation attack in Figure 2 assuming that *Adv* computes $\mathbf{C}^{'} = \mathbf{C} \pm 2^0$.

Notice that, *Adv* does not know a-priori if, when he succeedes in changing $\overline{K}_2[0]$, he gets $\mathbf{C} + 2^0$ or $\mathbf{C} - 2^0$. Therefore, the attack, in order to find a sequence $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$, accepted by the Tag, needs to be applied twice: once, let us say, during the first round, by setting $\mathbf{C}' = \mathbf{C} + 2^0$ and, if going through all possible $\mathbf{A}'$ no sequence is found, one more time, during the second round, by setting $\mathbf{C}' = \mathbf{C} - 2^0$ and trying again. Eventually, the procedure halts because, as we have seen before, at each iteraction a different position of $\overline{K}_2$ is flipped, and going through all possible $\mathbf{A}'$ means trying all possible variations of $\overline{K}_2$ in which a single bit is flipped. Hence, there exists a certain iteraction, say the j-th one, which flips $\overline{K}_2[0]$ and produces either $\mathbf{C} + 2^0$ or $\mathbf{C} - 2^0$. During the first round, when the j-th iteraction occurs, if $\mathbf{C}^{'}$ has been properly set, the Tag accepts the sequence $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$ and replies with $\mathbf{D}'$. Otherwise, the Tag will accept at the second round when the same iteraction occurs and $\mathbf{C}^{'}$ has been properly set.

Since $\mathbf{A}$ is a uniformly distributed random 96-bit string, *Adv* has to try on average $\frac{192+1}{2} = 96.5$ times to find the position of A which needs to be flipped. However, the above de-synchronization attack can be improved. Indeed, notice that,

$$2^{95} \bmod 2^{96} = -2^{95} \bmod 2^{96}.$$

Hence, it holds that

$$\mathbf{C} + 2^{95} \bmod 2^{96} = \mathbf{C} - 2^{95} \bmod 2^{96}.$$

Therefore, if *Adv* computes in step 1. of the procedure given in Figure 2 the value $\mathbf{C}' = \mathbf{C} + 2^{95} = \mathbf{C} - 2^{95}$, then he does not need to consider the two cases, i.e., $\mathbf{C}' = \mathbf{C} + 2^0$ and $\mathbf{C}' = \mathbf{C} - 2^0$. Each interaction counts for two interactions and the overall number of interactions is reduced. More precisely, *Adv* has to try on average $\frac{96+1}{2} = 48.5$ times to find a suitable sequence $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$ for the Tag.

Another important observation, which will play a key role in the next section, is the following: once a sequence $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$ which the Tag accepts is found, *Adv* discovers the value used in the rotation $Rot(K_2 \oplus n_1, K_2)$ to compute $\overline{K}_2$. Indeed, if the sequence $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$ has been computed by flipping the bit $\mathbf{A}[j]$, then the value z of the rotation is exactly $95 - j$.

4 Identity Disclosure

By building on the above de-synchronisation attack, in this section we show how *Adv* can compute the static *ID* stored in the Tag.

We start by noticing that, the same argument used before by modifying $\mathbf{A}$ and $\mathbf{C}$, in order to find a sequence the Tag accepts, can be applied to *every* position, i.e., we can compute a sequence $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$ by working on *any one* of $\mathbf{C}[95], \ldots, \mathbf{C}[0]$. Indeed, all *Adv* has to do when working on the i-th position, with $i \in \{0, \ldots, 95\}$, is to set $\mathbf{C}' = \mathbf{C} \pm 2^i$, and then look for an $\mathbf{A}'$ such that the sequence $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$ is accepted.

Moreover, notice that, once *Adv* has received a reply $\mathbf{D}'$ from the Tag, then he can compute a new forged sequence by working on any position of $\mathbf{C}$ in 1.5 interactions on average. Indeed, *Adv* knows the amount z of the rotation and, hence, he knows *exactly* in which position of $\mathbf{A}$ he has to flip a bit in order to add or subtract 2^i to $\mathbf{C}$. Therefore, *Adv* just needs to check if, by flipping a certain bit in $\mathbf{A}$, he gets $\mathbf{C}' = \mathbf{C} + 2^i$ or $\mathbf{C}' = \mathbf{C} - 2^i$.

Let us represent the static identifier ID as $ID[95]\dots ID[0]$. The key idea in the attack is *to collect pairs of values* $\mathbf{D}, \mathbf{D}'$, where $\mathbf{D}'$ is the value sent from the Tag to *Adv* as a reply to a forged sequence $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$, and to analyse the *differences* given by $\mathbf{D} \oplus \mathbf{D}'$. As we will show in a while, the differences give to *Adv* information about the ID and some other values used in the computation both by the Reader and the Tag. Notice that the attack described in this section *does not* enable *Adv* to compute the MSB of the ID, and gives two possible candidate values for the ID (and for $\overline{K}_2$).

We will proceed as follows: we first describe an identity disclosure attack which works only in a *special case*. Then, we show how to turn the general case to the special case through a pre-processing stage. The identity disclosure attack for the special case is given in Figure 3. We assume that $\mathbf{IDS}, \mathbf{A}||\mathbf{B}||\mathbf{C}, \mathbf{D}$ are the transcript of an honest execution *Adv* has looked at, and that the desynchronisation attack, described in Figure 2, in order to compute the amount z of the rotation $Rot(x, z)$ has been applied. Notice that, w.r.t. the description given in Figure 2, for the aforementioned efficiency reasons, in step 1. *Adv* computes $\mathbf{C} = \mathbf{C} + 2^{95}$ instead of $\mathbf{C} = \mathbf{C} \pm 2^0$.

Adv's Computation. After looking the values $\mathbf{IDS}$, $\mathbf{A}||\mathbf{B}||\mathbf{C}$, and $\mathbf{D}$, and computed the amount of the rotation z, it performs the following steps:

1. Let $i = 0$.
2. Using the knowledge of z, computes the two sequences $\mathbf{A}^i||\mathbf{B}||\mathbf{C}^i$, where $\mathbf{A}^i$ is obtained by flipping in $\mathbf{A}$ the bit $\mathbf{A}[(i - z) \bmod 96]$ and the values $\mathbf{C}^i$ is either $\mathbf{C} + 2^i$ or $\mathbf{C} - 2^i$.
3. Interacts with the Tag and starts a section with the old $\mathbf{IDS}$. Sends one of the sequence $\mathbf{A}^i||\mathbf{B}||\mathbf{C}^i$ to the Tag. If it is not accepted, starts a new section with the old $\mathbf{IDS}$ and sends the second one. One of them will be accepted.
4. The Tag sends back to *Adv*, as a reply to one of $\mathbf{A}^i||\mathbf{B}||\mathbf{C}^i$, a value, say $\mathbf{D}^i$.
5. From $\mathbf{D}^i$ and $\mathbf{D}$, the value the Tag sends to the Reader during the honest execution of the authentication protocol *Adv* has looked at, *Adv* computes the i-th bit of the static identifier as
$$ID[i] = \mathbf{D}[i + 1] \oplus \mathbf{D}^i[i + 1]. \qquad (1)$$
6. If $i < 95$, sets $i = i + 1$ and repeats from step 2.

Figure 3: SASI: Identity disclosure attack.

When (and why) does the attack of Figure 3 work? Notice that, by definition

$$\mathbf{D} = (\overline{K_2} + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1}).$$

Let us denote with $\overline{K}_2^i$ the value of $\overline{K_2}$ obtained when the Tag accepts $\mathbf{A}^i||\mathbf{B}||\mathbf{C}^i$. The value $\overline{K}_2^i$ is equal to $\overline{K}_2$ in all bits but the i-th one, which is flipped. Then, it results

$$\mathbf{D}^i = (\overline{K}_2^i + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1}).$$

Therefore,

$$\mathbf{D} \oplus \mathbf{D}^i = (\overline{K_2} + ID) \oplus (\overline{K}_2^i + ID).$$

Let us look at the i-th position. It holds that

$$\mathbf{D}[i] \oplus \mathbf{D}^i[i] = (\overline{K_2}[i] + ID[i] + c_i) \oplus (\overline{K}_2^i[i] + ID[i] + c_i^i),$$

where, for $i = 0, \ldots, 95$, we denote with c_i and c_i^i the carries from the sum of the bits of the previous positions, i.e., c_i is the carry generated by the sum $\overline{K_2}[i-1] + ID[i-1] + c_{i-1}$ and c_i^i is the carry generated by the sum $\overline{K_2}^i[i-1] + ID[i-1] + c_{i-1}$. Since there is no carry in the 0-th position, we set $c_0 = c_0^0 = 0$.

In general, for $i, j = 0, \ldots, 95$, we denote with c_i^j the carry generated by the sum $\overline{K_2}^j[i-1] + ID[i-1] + c_{i-1}^j$ and we set, for $j = 1, \ldots, 95$, $c_0^j = 0$. Moreover, notice that, in the i-th iteration which aims at computing the i-th bit of ID, for $s = 0, \ldots, i-1$, since $\overline{K}_2[s] = \overline{K}_2^i[s]$, it holds that $c_{s+1} = c_{s+1}^i$. In particular, observe that $c_i = c_i^i$.

Let us assume that $c_i = c_i^i = 0$. By definition, the bits $\overline{K}_2[i]$ and $\overline{K}_2^i[i]$ are one the complement of the other. Hence, either if $ID[i] = 0$ or $ID[i] = 1$, it holds that $\mathbf{D}[i] \oplus \mathbf{D}^i[i] = 1$. On the other hand, if $ID[i] = 0$, it holds that $c_{i+1} = c_{i+1}^i = 0$ and, hence, since $\overline{K}_2[i+1] = \overline{K}_2^i[i+1]$, that $\mathbf{D}[i+1] \oplus \mathbf{D}^i[i+1] = 0$; while, if $ID[i] = 1$, then either $(\overline{K}_2[i] + ID[i])$ or $(\overline{K}_2^i[i] + ID[i])$ produces *a carry* to the next position in the computation of $\mathbf{D}$ or of $\mathbf{D}^i$, respectively, i.e., $c_{i+1} \neq c_{i+1}^i$. Therefore, $\mathbf{D}[i+1] \oplus \mathbf{D}^i[i+1] = 1$. Hence, equation (1) holds.

By giving a closer look at the identity disclosure attack presented in Figure 3, it comes out that it works *surely* for the LSB of the ID, i.e., to compute $ID[0]$. However, in general, it does not work: for example, if $c_i = c_i^i = 1$ and $ID[i] = 0$, by computing $ID[i]$ through equation (1), we draw a wrong conclusion! Equation (1) can be applied successfully to compute the bits of ID if, at the i-th iteraction, for $i = 1, \ldots, 95$, it holds that $c_i = c_i^i = 0$, i.e., there is no carry from the sum of the previous bits. The carries are all zero if $c_1 = 0$ and, for $j = 1, \ldots, 95$, it holds that $ID[j] \neq \overline{K}_2[j]$. Indeed, in this case we get, for $j = 0, \ldots, 95$, $c_j = 0$. Moreover, it is immediate to check that $\overline{K}_2^i$, when summed up to ID, produces a carry $c_i^i (= c_i) = 0$.

Besides, notice that the attack described in Figure 3, by reversing equation (1), i.e., computing $ID[i] = 1 - \mathbf{D}[i+1] \oplus \mathbf{D}^i[i+1]$, can be applied when, at the i-th iteraction, for $i = 1, \ldots, 95$, it holds that $c_i = c_i^i = 1$. Again, the carries are all one if $c_1 = 1$ and, for $j = 1, \ldots, 95$, it holds that, $ID[j] \neq \overline{K}_2[j]$. Indeed, in this case we get, for $j = 1, \ldots, 95$, $c_j = 1$. Moreover, as before, $\overline{K}_2^i$, when summed up to ID, produces a carry $c_i^i (= c_i) = 1$.

We will refer to the configuration of the strings ID and $\overline{K}_2$ in which

$$ID[j] \neq \overline{K}_2[j], \text{ for } j = 1, \ldots, 95,$$

as to the *special case.*

We have identified a method, which we present in the following, *to reduce* the general case to the special case. Once the special case has been reached, we compute, up to the MSB, two possible values for ID (and, resp., $\overline{K}_2$). The first one is obtained assuming that $c_1 = 0$ (and, hence, $c_i = c_i^i = 0$, for $i = 1, \ldots, 95$) and applying the procedure described in Figure 3. The second one is computed assuming that $c_1 = 1$ (and, hence, $c_i = c_i^i = 1$, for $i = 1, \ldots, 95$) and simply flipping the bits of ID (and, resp., $\overline{K}_2$) obtained in the previous case. Actually, some more details have to be taken into account, but we will deal with them in the following parts of the paper.

Roughly speaking our reduction works as follows: in the attack described in Figure 3, Adv eavesdrops an execution and stores $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and $\mathbf{D}$. Then, he sends $\mathbf{A}^0, \mathbf{B}, \mathbf{C}^0$ and gets $\mathbf{D}^0$ from the Tag. To retrieve $ID[0]$, Adv computes $\mathbf{D} \oplus \mathbf{D}^0$. As we will show in a while, Adv, from this string also gets information about the condition $ID[j] \overset{?}{\neq} \overline{K}_2[j]$, for $j = 1, \ldots, 95$. If $\mathbf{D} \oplus \mathbf{D}^0$ says that the condition is not met yet, Adv, starting from $\mathbf{A}, \mathbf{B}, \mathbf{C}$, constructs a new pair of forged sequences, let us say $\mathbf{A}_s, \mathbf{B}_s, \mathbf{C}_s$ and $\mathbf{A}_s^0, \mathbf{B}_s^0, \mathbf{C}_s^0$, gets back $\mathbf{D}_s$ and $\mathbf{D}_s^0$, respectively, and checks $\mathbf{D}_s \oplus \mathbf{D}_s^0$. Adv repeats the process by constructing new pairs of forged sequences, starting from the ones used at the previous iteration, as long as the condition is not met. We will show that, in the worst case, the process requires 94 iteractions but, on average, it ends after 47.5 iteractions.

Let us start by presenting a lemma which characterizes the structure of $\mathbf{D}\oplus\mathbf{D}^0$, if certain conditions are satisfied. Let denote by $0^{96-r}1^r$ a 96-bit string composed of $96-r$ bits equal to 0 followed by r bits equal to 1. We remind the reader that $\mathbf{D}$ and $\mathbf{D}^0$ are such that $\overline{K}_2$ and $\overline{K}_2^0$ differ only in the least significant bit, i.e., $\overline{K}_2^0[0] = 1 - \overline{K}_2[0]$. The following result holds:

Lemma 1 $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-r}1^r$, *where* $1 < r \leq 96$, *if and only if the following conditions hold:*

1. $ID[0] = 1$.
2. *For* $j = 1, \dots, r-2$, $\overline{K}_2[j] \neq ID[j]$.
3. *If* $r < 96$, *then* $\overline{K}_2[r-1] = ID[r-1]$.

Proof. (Only if part). The condition $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96}1^r$ can be graphically depicted as follows:

$$\begin{array}{cccccc}
\mathbf{D} & \dots & (\overline{K}_2[r] + ID[r] + c_r) & (\overline{K}_2[r-1] + ID[r-1] + c_{r-1}) & \dots & (\overline{K}_2[1] + ID[1] + c_1) & (\overline{K}_2[0] + ID[0]) \\
 & \dots & \oplus & \oplus & \dots & \oplus & \oplus \\
\mathbf{D}^0 & \dots & (\overline{K}_2^0[r] + ID[r] + c_r^0) & (\overline{K}_2^0[r-1] + ID[r-1] + c_{r-1}^0) & \dots & (\overline{K}_2^0[1] + ID[1] + c_1^0) & (\overline{K}_2^0[0] + ID[0]) \\
\hline
\mathbf{D} \oplus \mathbf{D}^0 = & 0 \dots 0 & 0 & 1 & \dots & 1 & 1
\end{array}$$

Concerning with point 1., by definition $\overline{K}_2[1] = \overline{K}_2^0[1]$ and by assumption $\mathbf{D}[1] \oplus \mathbf{D}^0[1] = 1$. The latter result is possible only if one (and only one) of the two strings $\mathbf{D}$ and $\mathbf{D}^0$, in the 0-th position, has *generated* a carry to the next position, i.e. $c_1 \neq c_1^0$. Since $\overline{K}_2[0] \neq \overline{K}_2^0[0]$, then $c_1 \neq c_1^0$ implies $ID[0] = 1$.

Point 2. can be shown arguing by contradiction. For $j = 1, \dots, 95$, by definition it holds that $\overline{K}_2[j] = \overline{K}_2^0[j]$. If were $\overline{K}_2[j] = ID[j] = 0$ for some $j \in \{1, \dots, r-2\}$, then it would be $c_{j+1} = c_{j+1}^0 = 0$ and, hence, $\mathbf{D}[j+1] \oplus \mathbf{D}^0[j+1] = 0$, which contradicts the assumption. Similarly, if were $\overline{K}_2[j] = ID[j] = 1$ for some $j \in \{1, \dots, r-2\}$, then it would be $c_{j+1} = c_{j+1}^0 = 1$ and, again, $\mathbf{D}[j+1] \oplus \mathbf{D}^0[j+1] = 0$.

Finally, point 3. follows from the same argument we have used to prove point 2. The equalities $\mathbf{D}[r-1] \oplus \mathbf{D}^0[r-1] = 1$ and $\mathbf{D}[r] \oplus \mathbf{D}^0[r] = 0$ hold only if $\overline{K}_2[r-1] = ID[r-1]$. Indeed, $\overline{K}_2[r-1] \neq ID[r-1]$ would keep propagating the carry to position r and would result $\mathbf{D}[r] \oplus \mathbf{D}^0[r] = 1$.

Hence, if $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-r}1^r$, with $1 < r \leq 96$, then points 1., 2. and 3. hold. By using a similar reasoning, we prove the "if part" of the lemma.

(If part). Let us assume that $ID[0] = 1$, that for $j = 1, \dots, r-2$, $\overline{K}_2[j] \neq ID[j]$, and that, if $r < 96$, then $\overline{K}_2[r-1] = ID[r-1]$. Then, the following holds:

Since by assumption $\overline{K}_2[0] \neq \overline{K}_2^0[0]$, then $\mathbf{D}[0] \oplus \mathbf{D}^0[0] = 1$. Moreover, $c_1 \neq c_1^0$.

Then, since for $j = 1, \dots, r-2$, by assumption $\overline{K}_2[j] \neq ID[j]$, by definition $\overline{K}_2[j] = \overline{K}_2^0[j]$, and we have shown that $c_1 \neq c_1^0$, then, for $j = 2, \dots, r-1$, it holds that $c_j \neq c_j^0$. This implies that, for $j = 1, \dots, r-2$, it results $\mathbf{D}[j] \oplus \mathbf{D}^0[j] = 1$.

By assumption $\overline{K}_2[r-1] = \overline{K}_2^0[r-1]$. If $\overline{K}_2[r-1] = ID[r-1] = 0$, then the propagation of the carry to the next position ends, i.e., $c_r = c_r^0 = 0$, and it holds that $\mathbf{D}[r-1] \oplus \mathbf{D}^0[r-1] = 1$ and $\mathbf{D}[r] \oplus \mathbf{D}^0[r] = 0$; else, i.e., $\overline{K}_2[r-1] = ID[r-1] = 1$, in both $\mathbf{D}$ and $\mathbf{D}^0$ the $(r-1) - th$ position generates a carry to the next position, i.e., $c_r = c_r^0 = 1$, and, again, it holds that $\mathbf{D}[r-1] \oplus \mathbf{D}^0[r-1] = 1$ and $\mathbf{D}[r] \oplus \mathbf{D}^0[r] = 0$.

Finally, it is immediate to see that, for $j = r+1, \dots, 95$, it holds that $c_j = c_j^0$. Indeed, $c_r = c_r^0$ and by definition, for $j = r+1, \dots, 95$, $\overline{K}_2[j] = \overline{K}_2^0[j]$. Therefore, for $j = r+1, \dots, 95$, it results $\mathbf{D}[j] \oplus \mathbf{D}^0[j] = 0$.

In conclusion, if points 1., 2. and 3. are satisfied, then $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-r}1^r$, with $1 < r \leq 96$. ■

We point out that the above characterization is *independent of* the value of $\overline{K}_2[0]$.

In order to reduce the general case to the special case, we need to *change* in a suitable way the form of the strings we get back from the Tag. To this aim, notice that, interacting with the Tag in a certain way, we can *extend* the substring of 1s in $\mathbf{D} \oplus \mathbf{D}^0$. Indeed, by flipping in $\overline{K}_2$ and $\overline{K}_2^0$ the r-th bit, the equalities $\overline{K}_2[r-1] = ID[r-1]$ and $\overline{K}_2^0[r-1] = ID[r-1]$ are broken. Hence, it is *extended* the substring of $\mathbf{D}$ or $\mathbf{D}^0$ which *propagates* the carry c_1 or c_1^0, generated in the 0-th position, and it holds that $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-t}1^t$, where $t > r$. More precisely, the following result holds:

Lemma 2 *Let* $\mathbf{D}$ *and* $\mathbf{D}^0$ *such that* $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-r}1^r$, *with* $1 < r < 96$. *Let* $\mathbf{D}_r = (\overline{K}_{2,r} + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1})$ *and* $\mathbf{D}_r^0 = (\overline{K}_{2,r}^0 + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1})$ *such that* $\overline{K}_{2,r}$ *(resp.,* $\overline{K}_{2,r}^0$*) is equal in every position to* $\overline{K}_2$ *(resp.,* $\overline{K}_2^0$*) but in position* $r-1$*, in which the bit has been flipped, i.e.,* $\overline{K}_{2,r}[r-1] = 1 - \overline{K}_2[r-1]$ *(resp.,* $\overline{K}_{2,r}^0[r-1] = 1 - \overline{K}_2^0[r-1]$*). Then,*

$$\mathbf{D}_r \oplus \mathbf{D}_r^0 = 0^{96-t}1^t, \text{ where } r < t \le 96.$$

Proof. We show that $\mathbf{D}_r$ and $\mathbf{D}_r^0$ satisfy conditions 1., 2. and 3. of Lemma 1 for a certain integer $t > r$. Hence, the result follows. More precisely:

Since $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-r}1^r$, with $1 < r < 96$, due to the 'only if part' of Lemma 1, it holds that $ID[0] = 1$, for $j = 1, \ldots, r-2$, $\overline{K}_2[j] \neq ID[j]$, and $\overline{K}_2[r-1] = ID[r-1]$.

By definition, $\overline{K}_{2,r}$ is equal in every position to $\overline{K}_2$ but in the $(r-1)$-th one, in which the bit has been flipped. Therefore, for $j = 1, \ldots, r-2$, it holds that $\overline{K}_{2,r}[j] = \overline{K}_2[j]$. Since for $j = 1, \ldots, r-2$, $\overline{K}_2[j] \neq ID[j]$, it follows that, for $j = 1, \ldots, r-2$, $\overline{K}_{2,r}[j] \neq ID[j]$.

Then, by definition, $\overline{K}_{2,r}[r-1] \neq ID[r-1]$. Finally, either there exists an integer $r < t < 96$ such that $\overline{K}_{2,r}[t-1] = ID[t-1]$ or let us set $t = 96$ and it holds that $\overline{K}_{2,r}[j] \neq ID[j]$, for $j = r, \ldots, t-1$.

Applying the 'if part' of Lemma 1, we get that $\mathbf{D}_r \oplus \mathbf{D}_r^0 = 0^{96-t}1^t$, for $r < t \le 96$. ■

4.1 A Simplified Attack

We are now ready to describe the whole attack. To simplify the analysis and the description, w.l.o.g., let us assume that $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-r}1^r$, with $r > 1$. We will remove such an assumption in subsection 4.3. The attack works in two phases: Pre-processing Stage and Disclosure Stage.

Pre-processing Stage. *Adv* gets efficiently a pair of sequences $\mathbf{A}_s||\mathbf{B}||\mathbf{C}_s, \mathbf{D}$ and $\mathbf{A}_s^0||\mathbf{B}||\mathbf{C}_s^0, \mathbf{D}^0$ such that $\mathbf{D}_s \oplus \mathbf{D}_s^0 = 1^{96}$. To this aim, he proceeds as described in Figure 4.

Notice that step 2. takes on average 1.5 iteractions. Indeed, *Adv*, before sending the sequence $\mathbf{A}^0||\mathbf{B}||\mathbf{C}^0$, has computed the amount z of the rotation. Hence, he knows exactly which bit of $\mathbf{A}$ has to flip in each iteraction. On the other hand, step 4. takes almost 1 interaction. More precisely, only the first time, to compute $\mathbf{A}_r^0||\mathbf{B}||\mathbf{C}_r^0$, *Adv* has to check whether $\mathbf{C}_r^0$ has to be set equal to $\mathbf{C}_r + 2^0$ or equal $\mathbf{C}_r - 2^0$. Then, for any other sequence $\mathbf{A}_t^0||\mathbf{B}||\mathbf{C}_t^0$, constructed from $\mathbf{A}_t||\mathbf{B}||\mathbf{C}_t$, *Adv* has to use the same operation used to set $\mathbf{C}_r^0$, i.e., sum by 2^0 or subtraction by 2^0. Indeed, it is not difficult to see that *Adv*, moving from $\mathbf{A}_r||\mathbf{B}||\mathbf{C}_r$ to $\mathbf{A}_t||\mathbf{B}||\mathbf{C}_t$, does not modify the first r bits of $\mathbf{C}_r$, in constructing $\mathbf{C}_t$ from $\mathbf{C}_r$. Overall the procedure requires $\ell \cdot 1.5 + (\ell - 1) \cdot 1 + 1.5$ interactions, where the parameter ℓ represents the number of positions in which $\overline{K}_2$ and ID are equal. Since the strings are uniformly distributed, it holds that, on average, $\ell = 94/2$. Indeed, by assumption, $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-r}1^r$ with $r > 1$. Let us assume that $r = 2$ (worst case). Then, the preprocessing routine could flip every position j, where $1 \le j \le 94$, in case the event $ID[j] = \overline{K}_2[j]$ occurs. Overall such positions are 94. Notice that we do not consider position 95 because the procedure of Figure 3 cannot determine the MSB.

Disclosure Stage. Once *Adv* has a pair of sequences $\mathbf{A}_s||\mathbf{B}||\mathbf{C}_s, \mathbf{D}_s$ and $\mathbf{A}_s^0||\mathbf{B}||\mathbf{C}_s^0, \mathbf{D}_s^0$ such that $\mathbf{D}_s \oplus \mathbf{D}_s^0 = 1^{96}$, he assumes that $c_1 = 0$ and mounts the identity disclosure attack described in Figure 3. More precisely, *Adv*, for $i = 1, \ldots, 94$, waits for $\mathbf{D}_s^i$ from the Tag as a reply to a sequence $\mathbf{A}_s^i||\mathbf{B}||\mathbf{C}_s^i$,

Adv's Computation. Adv has eavesdropped and stored the values $\mathbf{IDS}$, $\mathbf{A}||\mathbf{B}||\mathbf{C}$, and $\mathbf{D}$. He has constructed $\mathbf{A}^0||\mathbf{B}||\mathbf{C}^0$, sent the sequence to the Tag, and got back $\mathbf{D}^0$ such that $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-r}1^r$, with $1 < r < 96$. Hence, he applies the following steps:

1. Let $F = \{r\}$.
2. Constructs and sends to the Tag a new sequence $\mathbf{A}_r||\mathbf{B}||\mathbf{C}_r$, modifying $\mathbf{A}$ and $\mathbf{C}$, in order to flip the r-th bit of $\overline{K}_2$.
3. The Tag replies with a value $\mathbf{D}_r$.
4. Then, Adv sends to the Tag a new sequence $\mathbf{A}_r^0||\mathbf{B}||\mathbf{C}_r^0$, constructed from $\mathbf{A}_r||\mathbf{B}||\mathbf{C}_r$, in order to flip the first bit of the new $\overline{K}_2$, i.e., $\overline{K}_2[0]$.
5. The Tag replies with $\mathbf{D}_r^0$.
6. Adv computes $\mathbf{D}_r \oplus \mathbf{D}_r^0 = 0^{96-t}1^t$, where $t > r$. If $t = 96$, then Adv has finished; otherwise, Adv repeats from step 2. working on the t-th bit, that is, setting $r = t$, $\mathbf{A}||\mathbf{B}||\mathbf{C} = \mathbf{A}_r||\mathbf{B}||\mathbf{C}_r$, $\mathbf{D} = \mathbf{D}_r$, and $F = F \cup \{t\}$.

Figure 4: Pre-processing for the identity disclosure attack.

constructed from $\mathbf{A}_s||\mathbf{B}||\mathbf{C}_s$. Once received $\mathbf{D}_s^i$, he computes the bit in the i-th position of ID using equation (1), i.e., $ID[i] = \mathbf{D}_s[i+1] \oplus \mathbf{D}_s^i[i+1]$.

Notice that, due to Lemma 1, it holds that $ID[0] = 1$. Hence, if $c_1 = 0$, then $\overline{K}_2[0] = 0$. Through the pre-processing stage the string $\overline{K}_2$ has been turned in the string $\overline{K}_{2,s}$ such that, $\overline{K}_{2,s}[0] = \overline{K}_2[0] = 0$, and for $j = 1, \ldots, 95$, the value $\overline{K}_{2,s}[j] \neq ID[j]$. Applying the identity disclosure attack of Figure 3, Adv computes (up to the MSB) ID_1 and, as a consequence, $\overline{K}^{'}_{2,s}$. The procedure requires $95 \cdot 1.5$ interactions to compute the first 95 bits of the ID.

Then Adv assumes that $c_1 = 1$. Lemma 1 implies that $\overline{K}_2[0] = 1$ and, due to the configuration of ID and $\overline{K}_{2,s}$, the carry is *propagated* until position 95. Hence, Adv computes the second pair of possible values by *flipping* the bits of ID_1 and $\overline{K}^{'}_{2,s}$ one by one, i.e., for $j = 1, \ldots, 94$, $ID_2[j] = 1 - ID_1[j]$ and $\overline{K}^{''}_{2,s}[j] = 1 - \overline{K}^{'}_{2,s}[j]$.

Finally, notice that, $\overline{K}_2$ has been modified in the pre-processing stage, in order to get $\overline{K}_{2,s}$, such that, for $j = 0, \ldots, 95$, $ID[j] \neq \overline{K}_{2,s}[j]$, i.e., some bits of $\overline{K}_2$ have been flipped. However, we know exactly *in which positions* the bits have been flipped. Such positions have been collected in F during the preprocessing stage. Hence, from the two strings $\overline{K}^{'}_{2,s}$ and $\overline{K}^{''}_{2,s}$, we compute the two possible strings $\overline{K}^{'}_2$ and $\overline{K}^{''}_2$, by *reversing* the flipped bits in $\overline{K}^{'}_{2,s}$ and $\overline{K}^{''}_{2,s}$.

We stress that the above method does not enable Adv to compute the MSB of the possible ID (and of $\overline{K}_2$). Therefore, Adv has to guess such bits.

Remark. The whole attack is effective and efficient. It requires (on average) 48.5 interactions with the Tag to find out the amount z of the rotation, 118 interactions (on average) for the pre-processing stage, and 142.5 interactions to compute the first 95 bits of the ID. Therefore, summing up, the whole identity disclosure attack requires, on average, $48.5 + 118 + 142.5 = 309$ interactions.

4.2 A Toy Example: Part One

To make things clearer, let us present a toy example. It works on 7-bit strings. All sum operations are performed mod 2^7. The left cyclic rotation is given by K_2 mod 7. Let

$$ID = 1011011, \quad \mathbf{IDS} = 1001110, \quad K_1 = 1100101, \text{ and } K_2 = 1001101.$$

Eavesdropping an execution. Assume that the Reader chooses uniformly at random $n_1 = 1010111$ and $n_2 = 1101001$. It follows that:

$$\mathbf{A} = 1111100, \mathbf{B} = 0111000, \overline{K}_1 = 1100000, \overline{K}_2 = 0011010, \text{ and } \mathbf{C} = 0101100.$$

Hence, the sequence the Reader sends to the Tag is

$$\mathbf{A}||\mathbf{B}||\mathbf{C} = 1111100||0111000||0101100,$$

and gets back from the Tag the string $\mathbf{D} = 0011101$.

Computing the amount of the rotation. *Adv*, by using the transcript of the above execution, tries to compute a new sequence $\mathbf{A}'||\mathbf{B}||\mathbf{C}'$ which is accepted by the Tag. *Adv* attacks the most significant bit of $\mathbf{C}$. Hence he sets $\mathbf{C}' = 1101100$ and looks for the correct $\mathbf{A}'$. It starts by modifying the bit in the 0-th position of $\mathbf{A}$, i.e., computes $\mathbf{A}' = 1111101$. Unfortunately, the sequence is not accepted. Therefore, it keeps going and, at the 7-th interaction it sends the sequence

$$\mathbf{A}'||\mathbf{B}||\mathbf{C}' = 0111100||0111000||1101100,$$

which is accepted by the Tag. Hence, *Adv* deduces that the amount of the left cyclic rotation is equal to 0.

Pre-processing stage. Then, *Adv* starts working in order to retrieve the ID. He focuses on the first bit. He sets $\mathbf{C}^0 = \mathbf{C} + 2^0 = 0101101$ and $\mathbf{A}^0 = 1111101$ but the sequence is not accepted by the Tag. Hence, he tries with $\mathbf{C}^0 = \mathbf{C} - 2^0 = 0101011$. In this case, the sequence

$$\mathbf{A}^0||\mathbf{B}||\mathbf{C}^0 = 1111101||0111000||0101011,$$

is accepted and the Tag replies with the string $\mathbf{D}^0 = 0011110$. Then, *Adv* computes

$$\mathbf{D} \oplus \mathbf{D}^0 = 0000011.$$

Since the string is different from 1^7, then *Adv* executes the procedure given in Figure 4.

Adv, in order to extend the substrings of 1's, *implicitely* flips the second bit of $\overline{K}_2$. More precisely, starting from $\mathbf{A}||\mathbf{B}||\mathbf{C}$, he constructs a new sequence by flipping the second bit of $\mathbf{A}$ to get $\mathbf{A}_2$ and computing $\mathbf{C}_2 = \mathbf{C} \pm 2^1$. The sequence

$$\mathbf{A}_2||\mathbf{B}||\mathbf{C}_2 = 1111110||0111000||0101010$$

with $\mathbf{C}_2 = \mathbf{C} - 2^1$ is accepted. The Tag replies with $\mathbf{D}_2 = 0011011$. Therefore, *Adv*, using $\mathbf{A}_2||\mathbf{B}||\mathbf{C}_2$, constructs a new sequence $\mathbf{A}_2^0||\mathbf{B}||\mathbf{C}_2^0$, by flipping the first bit of $\mathbf{A}_2$ and computing $\mathbf{C}_2^0 = \mathbf{C}_2 - 2^0$. The sequence

$$\mathbf{A}_2^0||\mathbf{B}||\mathbf{C}_2^0 = 1111111||0111000||0101001$$

is accepted. The Tag replies with $\mathbf{D}_2^0 = 0011100$. Hence, *Adv* computes

$$\mathbf{D}_2 \oplus \mathbf{D}_2^0 = 0000111.$$

Since the string is still different from 1^7, then *Adv* repeats the procedure starting from the sequence $\mathbf{A}_2||\mathbf{B}||\mathbf{C}_2$ and flipping the third bit of $\mathbf{A}_2$. The transcript of the next successful executions is given Figures 5 and 6.

$$\begin{array}{lcl}
\mathbf{A}_3||\mathbf{B}||\mathbf{C}_3 = 1111010||0111000||0100110 & \text{and} & \mathbf{D}_3 = 0011111 \\
\mathbf{A}_3^0||\mathbf{B}||\mathbf{C}_3^0 = 1111011||0111000||0100101 & \text{and} & \mathbf{D}_3^0 = 0010000 \\
\mathbf{A}_4||\mathbf{B}||\mathbf{C}_4 = 1110010||0111000||0011110 & \text{and} & \mathbf{D}_4 = 0000111 \\
\mathbf{A}_4^0||\mathbf{B}||\mathbf{C}_4^0 = 1110011||0111000||0011101 & \text{and} & \mathbf{D}_4^0 = 0011000 \\
\mathbf{A}_5||\mathbf{B}||\mathbf{C}_5 = 1100010||0111000||0001110 & \text{and} & \mathbf{D}_5 = 0110111 \\
\mathbf{A}_5^0||\mathbf{B}||\mathbf{C}_5^0 = 1100011||0111000||0001101 & \text{and} & \mathbf{D}_5^0 = 0001000 \\
\mathbf{A}_6||\mathbf{B}||\mathbf{C}_6 = 1000010||0111000||1101110 & \text{and} & \mathbf{D}_6 = 0010111 \\
\mathbf{A}_6^0||\mathbf{B}||\mathbf{C}_6^0 = 1000011||0111000||1101101 & \text{and} & \mathbf{D}_6^0 = 1101000
\end{array}$$

Figure 5: Pre-processing in the toy example.

$$\begin{array}{c}
\mathbf{D}_2 \oplus \mathbf{D}_2^0 = 0000111 \\
\mathbf{D}_3 \oplus \mathbf{D}_3^0 = 0001111 \\
\mathbf{D}_4 \oplus \mathbf{D}_4^0 = 0011111 \\
\mathbf{D}_5 \oplus \mathbf{D}_5^0 = 0111111 \\
\mathbf{D}_6 \oplus \mathbf{D}_6^0 = 1111111
\end{array}$$

Figure 6: Pre-processing in the toy example.

Disclosure stage. At this point, *Adv* can mount the identity disclosure attack given in Figure 3 by using as starting sequence $\mathbf{A}_6||\mathbf{B}||\mathbf{C}_6 = 1000010||0111000||1101110$. The transcript of the successful execution is given in Figures 7 and 8.

Using the values listed in Figure 8 and the set of positions $F = \{2, 3, 4, 5, 6\}$ in which $\overline{K}_2$ has been flipped, *Adv* gets the 8 pairs of values defined by

$$(ID_1, \overline{K}_2^{'}) = (*011011, *011010),$$

and

$$(ID_2, \overline{K}_2^{''}) = (*100101, *100101),$$

where $*$ denotes the unknown value of the bit and can be 0 or 1. It is immediate to check that the pair $(1011011, 0011010)$ is the right one.

4.3 The General Attack

To simplify the description, in presenting our attack, we have assumed that $\mathbf{D} \oplus \mathbf{D}^0 = 0^{96-r}1^r$, for an integer $1 < r < 96$. Due to Lemma 1, such an assumption implies that $ID[0] = 1$. If $\mathbf{D} \oplus \mathbf{D}^0 = 0^{95}1$, then it holds that $ID[0] = 0$. We can prove similar results to the ones given in Lemma 1 and Lemma

$$\begin{array}{lcl}
\mathbf{A}_6||\mathbf{B}||\mathbf{C}_6 = 1000010||0111000||1101110 & \text{and} & \mathbf{D}_6 = 0010111 \\
\mathbf{A}_6^0||\mathbf{B}||\mathbf{C}_6^0 = 1000011||0111000||1101101 & \text{and} & \mathbf{D}_6^0 = 1101000 \\
\mathbf{A}_6^1||\mathbf{B}||\mathbf{C}_6^1 = 1000000||0111000||1110000 & \text{and} & \mathbf{D}_6^1 = 1101001 \\
\mathbf{A}_6^2||\mathbf{B}||\mathbf{C}_6^2 = 1000110||0111000||1110010 & \text{and} & \mathbf{D}_6^2 = 0010011 \\
\mathbf{A}_6^3||\mathbf{B}||\mathbf{C}_6^3 = 1001010||0111000||1110110 & \text{and} & \mathbf{D}_6^3 = 1101111 \\
\mathbf{A}_6^4||\mathbf{B}||\mathbf{C}_6^4 = 1010010||0111000||1111110 & \text{and} & \mathbf{D}_6^4 = 1100111 \\
\mathbf{A}_6^5||\mathbf{B}||\mathbf{C}_6^5 = 1100010||0111000||0001110 & \text{and} & \mathbf{D}_6^5 = 0110111
\end{array}$$

Figure 7: Identity disclosure in the toy example.

$$\mathbf{D}_6 \oplus \mathbf{D}_6^0 = 1111111$$
$$\mathbf{D}_6 \oplus \mathbf{D}_6^1 = 1111110$$
$$\mathbf{D}_6 \oplus \mathbf{D}_6^2 = 0000100$$
$$\mathbf{D}_6 \oplus \mathbf{D}_6^3 = 1111000$$
$$\mathbf{D}_6 \oplus \mathbf{D}_6^4 = 1110000$$
$$\mathbf{D}_6 \oplus \mathbf{D}_6^5 = 0100000$$

Figure 8: Identity disclosure in the toy example.

2, and, as pointed out before, the attack just needs a small refinement in the pre-processing stage. More precisely, it holds:

Lemma 3 $\mathbf{D} \oplus \mathbf{D}^0 = 0^{95}1$ *if and only if* $ID[0] = 0$.

Proof. (Only if part) Since $\mathbf{D}[1] \oplus \mathbf{D}^0[1] = 0$ and, by definition, $\overline{K}_2[1] = \overline{K}_2^0[1]$, then it must be $c_1 = c_1^0$. But, by definition, $\overline{K}_2[0] \neq \overline{K}_2^0[0]$. Hence, $c_1 = c_1^0 (= 0)$ only if $ID[0] = 0$.

(If part) Let $ID[0] = 0$. Since, by definition, $\overline{K}_2[0] \neq \overline{K}_2^0[0]$, then $\mathbf{D}[0] \oplus \mathbf{D}^0[0] = 1$. Moreover, $c_1 = c_1^0 = 0$. Then, by definition, for $j = 1, \ldots, 95$, $\overline{K}_2[j] = \overline{K}_2^0[j]$. Therefore, for $j = 1, \ldots, 95$, it holds that $\mathbf{D}[j] \oplus \mathbf{D}^0[j] = 0$.

■

Notice that Lemmas 1 and 3 imply that $\mathbf{D} \oplus \mathbf{D}^0$ is a sequence of 0's followed by a sequence of 1's, that is, $\mathbf{D} \oplus \mathbf{D}^0 = 0^{95-r}1^r$ with $1 \leq r \leq 96$. Indeed, if $ID[0] = 0$, then $\mathbf{D} \oplus \mathbf{D}^0 = 0^{95}1$. On the other hand, if $ID[0] = 1$, then $\mathbf{D} \oplus \mathbf{D}^0 = 0^{95-r}1^r$, where the number $r > 1$ of bits equal to 1 on the right side of the string depends on properties 2. and 3. of Lemma 1.

It is easy to see that $\mathbf{D} \oplus \mathbf{D}^j$ has a similar form. More precisely, for $j = 1, \ldots, 95$, it holds that $\mathbf{D} \oplus \mathbf{D}^j = 0^{96-r-j}1^r0^j$, where $1 \leq r \leq 96 - j$. Indeed, for $\ell = 0, \ldots, j-1$, by definition $\overline{K}_2[\ell] = \overline{K}_2^j[\ell]$. Hence, for $\ell = 0, \ldots, j-1$, it holds that $\mathbf{D}[\ell] \oplus \mathbf{D}^j[\ell] = 0$. Then, for $\ell = j, \ldots, 96$, we apply exactly the same arguments we have used to prove Lemmas 1 and 3, i.e., just set the 0-th position in those cases to the j-th position in these ones.

Moreover, the generalisation of Lemma 2 we need is the following:

Lemma 4 *Let* $\mathbf{D}$ *and* $\mathbf{D}^j$ *such that* $\mathbf{D} \oplus \mathbf{D}^j = 0^{96-r-j}1^r0^j$, *for* $0 \leq j \leq 94$ *and* $1 < r < 96 - j$. *Let* $\mathbf{D}_{r+j} = (\overline{K}_{2,r+j} + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1})$ *and* $\mathbf{D}_{r+j}^0 = (\overline{K}_{2,r+j}^0 + ID) \oplus ((K_1 \oplus K_2) \vee \overline{K_1})$ *such that* $\overline{K}_{2,r+j}$ *(resp.,* $\overline{K}_{2,r+j}^0$*) is equal in every position to* $\overline{K}_2$ *(resp.,* $\overline{K}_2^0$*) but in position* $r+j-1$*, in which the bit has been flipped, i.e.,* $\overline{K}_{2,r+j}[r+j-1] = 1 - \overline{K}_2[r+j-1]$ *(resp.,* $\overline{K}_{2,r+j}^0[r+j-1] = 1 - \overline{K}_2^0[r+j-1]$*). Then,*

$$\mathbf{D}_{r+j} \oplus \mathbf{D}_{r+j}^0 = 0^{96-t-j}1^t0^j, \text{ where } r < t \leq 96 - j.$$

The above lemma, for $j = 0$, coincides with Lemma 2. For $j > 0$ it can be shown applying the same technique used to prove Lemma 2.

The pre-processing routine for the general case is given in Figure 9.

Roughly speaking, *Adv*, if $ID[0] = 0$, once got $\mathbf{D} \oplus \mathbf{D}^0 = 0^{95}1$, moves to the second position. In other words, *Adv* "does not care" about the first position and starts again the attack from the next position by sending to the Tag another sequence $\mathbf{A}^1||\mathbf{B}||\mathbf{C}^1$, constructed from $\mathbf{A}||\mathbf{B}||\mathbf{C}$, where $\mathbf{C}^1 = \mathbf{C} \pm 2^1$. *Adv* keeps going from the current position to the next one until, let us say at the j-th iteraction, he gets the string $\mathbf{D} \oplus \mathbf{D}^j = 0^{96-r-j}1^r0^j$, with $r > 1$. At this point, *Adv*, in order to get $\mathbf{D} \oplus \mathbf{D}^j = 1^{96-j}0^j$, applies the same pre-processing steps described before in Figure 4. Actually, it is immediate to check that, if $j = 0$, then the procedure given in Figure 9 coincides with the procedure given in Figure 4.

Adv's Computation. *Adv* has eavesdropped an execution and stored the values **IDS**, $\mathbf{A}||\mathbf{B}||\mathbf{C}$, and **D**. Hence, he applies the following steps:

1. Sets $j = 0$.
   (a) Constructs and sends $\mathbf{A}^j||\mathbf{B}||\mathbf{C}^j$ to the Tag and gets back $\mathbf{D}^j$.
   (b) Let $\mathbf{D} \oplus \mathbf{D}^j = 0^{96-r-j}1^r0^j$.
   (c) If $r = 1$, sets $j = j + 1$ and repeats from step (a).
2. Let $F = \{r + j\}$.
3. Constructs and sends to the Tag a new sequence $\mathbf{A}_{r+j}||\mathbf{B}||\mathbf{C}_{r+j}$, modifying **A** and **C**, in order to flip the $(r+j)$-th bit of $\overline{K}_2$, i.e., $\overline{K}_2[r+j-1]$.
4. The Tag replies with a value $\mathbf{D}_{r+j}$.
5. Then, *Adv* sends to the Tag a new sequence $\mathbf{A}^0_{r+j}||\mathbf{B}||\mathbf{C}^0_{r+j}$, constructed from $\mathbf{A}_{r+j}||\mathbf{B}||\mathbf{C}_{r+j}$, in order to flip the first bit of the new $\overline{K}_2$, i.e., $\overline{K}_2[0]$.
6. The Tag replies with $\mathbf{D}^0_{r+j}$.
7. *Adv* computes $\mathbf{D}_{r+j} \oplus \mathbf{D}^0_{r+j} = 0^{96-t-j}1^t0^j$, where $t > r$. If $t = 96 - j$, then *Adv* has finished; otherwise, *Adv* repeats from step 3. working on the $(t+j)$-th bit of $\overline{K}_2$, that is, setting $r = t$, $\mathbf{A}||\mathbf{B}||\mathbf{C} = \mathbf{A}_{r+j}||\mathbf{B}||\mathbf{C}_{r+j}$, $\mathbf{D} = \mathbf{D}_{r+j}$, and $F = F \cup \{t + j\}$.

Figure 9: Pre-processing: General case.

Let us assume that the preprocessing stage ends when *Adv* sends $\mathbf{A}_s, \mathbf{B}_s, \mathbf{C}_s$ and $\mathbf{A}^0_s, \mathbf{B}^0_s, \mathbf{C}^0_s$, gets back $\mathbf{D}_s$ and $\mathbf{D}^0_s$, respectively, and it holds that $\mathbf{D}_s \oplus \mathbf{D}^0_s = 1^{96-j}0^j$.

At this point, *Adv* (partially) runs the identity disclosure attack given in Figure 3, using the sequence $\mathbf{A}_s, \mathbf{B}_s, \mathbf{C}_s$, instead of $\mathbf{A}, \mathbf{B}, \mathbf{C}$, and starting from position j instead of position 0, i.e., setting in step 1. of the procedure $i = j$. Indeed, *Adv* already knows that, for $\ell = 0 \ldots j - 1$, it holds that $ID[\ell] = 0$. Notice that, for $\ell = 0 \ldots j - 1$, he knows *nothing* about $\overline{K}_{2,s}[\ell]$.

Due to the preprocessing routine, for $\ell = j \ldots 95$, it holds that $ID[\ell] \neq \overline{K}_{2,s}[\ell]$. Hence, assuming that $c_j = 0$, the identity disclosure attack yields the first possible sequence of bits $ID[94] \ldots ID[j]$ (and $\overline{K}_{2,s}[94] \ldots \overline{K}_{2,s}[j]$). Then, assuming that $c_j = 1$ and flipping the bits computed before, *Adv* obtains the second possible sequence of bits $ID[94] \ldots ID[j]$ (and $\overline{K}_{2,s}[94] \ldots \overline{K}_{2,s}[j]$).

At the end, by considering all possibilities for $ID[95]$ and $\overline{K}_{2,s}[95]$, *Adv* gets 8 different pairs

$$(ID[95] \ldots ID[j]0 \ldots 0, \overline{K}_2[95] \ldots \overline{K}_2[j] * \ldots *),$$

where the values $\overline{K}_2[j-1] \ldots \overline{K}_2[0]$ need to be determined. Such bits should be *exaustively* computed.

Hence, the number of possible pairs $(ID, \overline{K}_2)$ *Adv* gets, compared to the number in the case $ID[0] = 1$, grows from 8 to $8 \cdot 2^j$. However, if the *ID*s are chosen uniformly at random, the probability that the *ID* has a long sequence of 0s as least significant bits is low.

Remark. The attack still requires, on average, 48.5 interactions with the Tag to find out the amount z of the rotation and $\ell \cdot 1.5 + (\ell - 1) \cdot 1 + 1.5$ interactions for the pre-processing stage. The parameter ℓ in this case represents the number of positions in which $\overline{K}_2$ and *ID* are different plus the number of least significant bits of *ID* equal to zero. It can be computed as follows: since the strings are uniformly distributed, it holds that

$$\ell = \sum_{i=0}^{93} (\frac{94-i}{2} + i) \cdot \frac{1}{2^{i+1}} \cong 47.50.$$

Notice that the above formula considers the different forms of the ID and the probabilities with which they occur. More precisely, the index i represents the number of bits equal to 0 as least significant bits of ID. For example, when $i = 0$, then $ID[0] = 1$ and, as we have seen before, the number of positions in which ID and $\overline{K}_2$ could be equal, starting from position 1, is, on average, $\frac{94}{2}$. However, this event occurs with probability $\frac{1}{2}$. Similarly, when $i = 1$, then $ID[1]ID[0] = 10$. In this case the number of positions in which ID and $\overline{K}_2$ could be equal, starting from position 2, is, on average, $\ell = \frac{93}{2}$. However, this event occurs with probability $\frac{1}{4}$. The formula takes into account all possible configurations. Finally, the identity disclosure procedure requires, on average,

$$\sum_{i=0}^{94} \frac{95-i}{2^{i+1}} \cdot 1.5 \cong 141$$

interactions to compute the first 95 bits of the ID. Therefore, the whole identity disclosure attack requires, on average, $48.5 + 119.25 + 141 = 308.75$ interactions.

5 Full Disclosure

In this section we show how *Adv* can efficiently extract *all secret data*[2]. Notice that, in the following, for $q = 1, 2, 3$, we denote each value with a superscript index q, e.g., K_2^q instead of K_2, to indicate the protocol execution in which the value is sent, used or computed.

Assume that we have a *black box* procedure, let us say $\mathsf{BB}(\mathbf{IDS}, \mathbf{A}||\mathbf{B}||\mathbf{C}, \mathbf{D})$, to recover the ID of the Tag and the string $\overline{K}_2$. Moreover, let us assume that the black box procedure *does not* update the old tuple stored by the Tag after interacting with the Reader, i.e., the tuple $\mathbf{IDS}, K_1, K_2$. Then, *Adv* can extract all secret data from the Tag as described in Figure 10.

Adv's Computation.

1. Eavesdropping stage. Looks at an execution of the authentication protocol and stores $\mathbf{Hello}, \mathbf{IDS}^1, \mathbf{A}^1||\mathbf{B}^1||\mathbf{C}^1, \mathbf{D}^1$.

2. Identity disclosure stage. Applies $\mathsf{BB}(\mathbf{IDS}^1, \mathbf{A}^1||\mathbf{B}^1||\mathbf{C}^1, \mathbf{D}^1)$ and gets ID and $\overline{K}_2$.

3. Re-synchronisation stage. Re-synchronise the Tag with the Reader by sending the sequence $\mathbf{A}^1||\mathbf{B}^1||\mathbf{C}^1$ as a reply to $\mathbf{IDS}^1$, engaging an instance of the authentication protocol.

4. Eavesdropping stage. Looks at two new executions of the authentication protocol and stores $\mathbf{Hello}, \mathbf{IDS^2}, \mathbf{A}^2||\mathbf{B}^2||\mathbf{C}^2, \mathbf{D}^2$, and $\mathbf{Hello}, \mathbf{IDS^3}, \mathbf{A}^3||\mathbf{B}^3||\mathbf{C}^3, \mathbf{D}^3$.

5. Secret Data Extraction. Computes the secret keys.

Figure 10: Full disclosure attack.

[2]Notice that, the current section, clarifies and corrects the full disclosure attack described in Section 5 of the preliminary version of this paper.

The Reader, at the end of the first execution of the authentication protocol, once received the value $\mathbf{D}^1$, updates the old tuple (IDS^1, K_1^1, K_2^1) to the new tuple $(IDS^2, \overline{K}_1^1, \overline{K}_2^1)$. *Adv*, using the transcript of the execution, computes the ID and $\overline{K}_2^1$, re-synchronises the Tag with the Reader, and eavesdrops a second and a third executions of the authentication protocol. At the end of the second execution the Reader, once received the value $\mathbf{D}^2$, updates the old tuple $(IDS^2, \overline{K}_1^1, \overline{K}_2^1)$ to the new tuple $(IDS^3, \overline{K}_1^2, \overline{K}_2^2)$. After eavesdropping the third execution[3], *Adv* has enough information to compute everything. Indeed:

- From the equality $\mathbf{B}^2 = (\mathbf{IDS^2} \vee K_2^2) + n_2^2$, since he knows $\mathbf{B}^2, \mathbf{IDS}^2$, and K_2^2 (the former $\overline{K}_2^1$), he gets n_2^2, i.e.,
$$n_2^2 = \mathbf{B}^2 - (\mathbf{IDS}^2 \vee K_2^2).$$
- From the equality $\mathbf{IDS}^3 = (\mathbf{IDS^2} + ID) \oplus (n_2^2 \oplus \overline{K}_1^2)$, since he knows $\mathbf{IDS}^3, \mathbf{IDS}^2, ID$ and n_2^2, he computes $\overline{K}_1^2$, i.e.,
$$\overline{K}_1^2 = \mathbf{IDS}^3 \oplus (\mathbf{IDS}^2 + ID) \oplus n_2^2.$$
- From the equality $\overline{K}_1^2 = Rot(K_1^2 \oplus n_2^2, K_1^2)$, since he knows $\overline{K}_1^2$ and n_2^2, then he computes some possible values for K_1^2. More precisely, *Adv*, for $i = 1, \ldots, 96$, rotates $\overline{K}_1^2$ *on the right* for i positions and computes the bitwise xor of the rotated string with n_2^2. Let X be the string obtained in such a way. At this point *Adv* checks whether
$$\overline{K}_1^2 \stackrel{?}{=} Rot(X \oplus n_2^2, X).$$
If the quality holds, then X is a possible candidate for K_1^2. Notice that in the worst case there could be 96 candidates, but on average, just few values.
- From the equality $\mathbf{A}^2 = \mathbf{IDS}^2 \oplus K_1^2 \oplus n_1^2$, since he knows $\mathbf{A}^2, \mathbf{IDS}^2$ and some possible K_1^2, he computes some possible n_1^2, i.e.,
$$n_1^2 = \mathbf{A}^2 \oplus \mathbf{IDS}^2 \oplus K_1^2.$$
- Then, using K_2^2 and n_1^2, he computes some possible $\overline{K}_2^2 = Rot(K_2^2 \oplus n_1^2, K_2^2)$.
- Finally, using the equality
$$\mathbf{C}^2 = (K_1^2 \oplus \overline{K}_2^2) + (\overline{K}_1^2 \oplus K_2^2)$$
since he knows $\mathbf{C}^2$ and some sequences of the possibile four keys, *Adv* can check which candidates are consistent with the equation. We refer in the following to this check as to the C-test.

The complexity of the full disclosure attack is the *same* complexity of the identity disclosure attack, up to some simple computations. Moreover, it is easy to check that *Adv*, once computed the values $K_1^2, K_2^2, n_1^2, n_2^2$ used in the second execution, can also compute the other keys. The former keys, K_1^1 and K_2^1, can be computed through similar steps to the ones used before. More precisely:

- From the equality $\mathbf{IDS}^2 = (\mathbf{IDS^1} + ID) \oplus (n_2^1 \oplus \overline{K}_1^1)$, since he knows $\mathbf{IDS}^2, \mathbf{IDS}^1, ID$ and $\overline{K}_1^1 = K_1^2$, then *Adv* computes n_2^1, i.e.,
$$n_2^1 = \mathbf{IDS}^2 \oplus (\mathbf{IDS}^1 + ID) \oplus \overline{K}_1^1.$$

[3]Actually *Adv* only needs the pseudonym $\mathbf{IDS}^3$, sent by the Tag at the beginning of the third execution, in order to apply the attack. However, to simplify the description we assume that he eavesdrops the whole session.

- From the equality $\overline{K}_1^1 = Rot(K_1^1 \oplus n_2^1, K_1^1)$, since he knows $\overline{K}_1^1$ and n_2^1, then *Adv* computes some possible values for K_1^1. More precisely, *Adv*, for $i = 1, \ldots, 96$, rotates $\overline{K}_1^1$ *on the right* for i positions and computes the bitwise xor of the rotated string with n_2^1. Let X be the string obtained in such a way. At this point *Adv* checks whether

$$\overline{K}_1^1 \stackrel{?}{=} Rot(X \oplus n_2^1, X).$$

  If the quality holds, then X is a possible candidate for K_1^1.

- Finally, from the equality $\mathbf{C}^1 = (K_1^1 \oplus \overline{K}_2^1) + (\overline{K}_1^1 \oplus K_2^1)$, since he knows $\mathbf{C}^1, K_1^1, \overline{K}_2^1$ and $\overline{K}_1^1$, then *Adv* computes K_2^1, i.e.,

$$K_2^1 = [\mathbf{C}^1 - (K_1^1 \oplus \overline{K}_2^1))] \oplus \overline{K}_1^1.$$

At the end of the process, some sequences of possible keys used by Reader and Tag during the executions of the authentication protocol, are obtained.

However, notice that the above attack requires a black box procedure which *uniquely* computes ID and $\overline{K}_2$. Unfortunately, the attack we have described in Section 4, provides $8 \cdot 2^j$ possible pairs $(ID, \overline{K}_2)$. For each possible pair, we can compute some different sequences of secrets $K_1^3, K_2^3, K_1^2, K_2^2, K_1^1, K_2^1$. Fortunately, we can *reduce the ambiguities* by using the data Reader and Tag exchange during the three honest interactions. More precisely, for each sequence of possible keys $K_1^3, K_2^3, K_1^2, K_2^2, K_1^1, K_2^1$ we can check whether the test in Figure 11, hereafter referred to as the D-test, is satisfied.

$$\mathbf{D}^1 \stackrel{?}{=} (K_2^2 + ID) \oplus ((K_1^1 \oplus K_2^1) \vee K_1^2) \text{ and } \mathbf{D}^2 \stackrel{?}{=} (K_2^3 + ID) \oplus ((K_1^2 \oplus K_2^2) \vee K_1^3).$$

Figure 11: D-test for reducing ambiguities.

Only the sequences of secret keys which satisfy the D-test are sequences of possible keys.
To get an idea of the number of possible ambiguities, we have implemented the protocol and the three attacks, and we have run some tests, reported in the next section. However, before moving to the next section, we present a simple example of the steps of the above attack.

5.1 A Toy Example: Part Two

Let us assume that the first execution *Adv* looks at is the same given in Subsection 4.2. Hence, $\mathbf{IDS}^1 = 1001110$, and the first sequence the Reader sends to the Tag is

$$\mathbf{A}^1 || \mathbf{B}^1 || \mathbf{C}^1 = 1111100||0111000||0101100,$$

and gets back from the Tag the string $\mathbf{D}^1 = 0011101$.

Applying the identity disclosure attack, *Adv* gets 8 possible pairs $(ID, \overline{K}_2^1)$ (see Figure 12).

(0011011, 0011010)	(0011011, 1011010)	(1011011, 0011010)	(1011011, 0011010)
(0100101, 0100101)	(0100101, 1100101)	(1100101, 0100101)	(1100101, 1100101)

Figure 12: Possible pairs.

Then, *Adv* re-synchronises Reader and Tag by sending again the sequence $\mathbf{A}^1 || \mathbf{B}^1 || \mathbf{C}^1 = 1111100||0111000||0101100$ to the Tag, and looks at other two honest executions between Reader and

Tag. Assume that, for the second execution, the Reader chooses uniformly at random $n_1^2 = 0110100$ and $n_2^2 = 1000110$. Since $K_1^2 = 1100000$, $K_2^2 = 0011010$ and $\mathbf{IDS}^1 = 1001110$, it follows that:

$$\mathbf{IDS}^2 = 0100000, \mathbf{A}^2 = 1110100, \mathbf{B}^2 = 0000000, \overline{K}_1^2 = 1001001, \overline{K}_2^2 = 1001011, \text{ and } \mathbf{C}^2 = 1111110.$$

Hence, the sequence the Reader sends to the Tag is

$$\mathbf{A}^2||\mathbf{B}^2||\mathbf{C}^2 = 1110100||0000000||1111110,$$

and gets back from the Tag the string $\mathbf{D}^2 = 1011101$. Then, assume that, for the third execution, the Reader chooses uniformly at random $n_1^3 = 1011101$ and $n_2^3 = 1001001$. Since $K_1^3 = 1001001$ and $K_2^3 = 1001011$, it follows that:

$$\mathbf{IDS}^3 = 1110100, \mathbf{A}^3 = 1100000, \mathbf{B}^3 = 1001000, \overline{K}_1^3 = 0000000, \overline{K}_2^3 = 1000101, \text{ and } \mathbf{C}^3 = 1010111.$$

Hence, the sequence the Reader sends to the Tag is

$$\mathbf{A}^3||\mathbf{B}^3||\mathbf{C}^3 = 1100000||1001000||1010111,$$

and gets back from the Tag the string $\mathbf{D}^3 = 0100010$.

At this point, *Adv* for each possible pair $(ID, \overline{K}_2^1)$ performs the computations described step by step before. It is easy to check that, when the third pair of the first row in Figure 12 is used, i.e., $(1011011, 0011010)$, *Adv* gets the following values:

- From $\mathbf{B}^2 = 0000000$, since he knows also $\mathbf{IDS}^2 = 0100000$ and $K_2^2 = 0011010$, then he computes
$$n_2^2 = 1000110.$$
- From $\mathbf{IDS}^3 = 1110100$, since he knows also $\mathbf{IDS}^2 = 0100000$, $ID = 1011011$, and $n_2^2 = 1000110$, he computes
$$\overline{K}_1^2 = 1001001.$$

From $\overline{K}_1^2 = Rot(K_1^2 \oplus n_2^2, K_1^2)$, since he knows $\overline{K}_1^2$ and n_2^2, then he computes some possible values for K_1^2 (see Figure 13.)

Amount of Rshift	0	1	2	3	4	**5**	6
	1000101 $\oplus$ 1000110	1100100 $\oplus$ 1000110	0110010 $\oplus$ 1000110	0011001 $\oplus$ 1000110	1001100 $\oplus$ 1000110	0100110 $\oplus$ 1000110	0010011 $\oplus$ 1000110
K_1^2	0001111	0100010	1110100	1011111	0001010	1100000	1010101
K_1^2 mod 7	1	6	4	4	3	**5**	1

Figure 13: Possible values for K_1^2.

In this case, there is *only* one possible $K_1^2 = 1100000$. Hence, *Adv* has computed the keys used at the second executions if the pair $(ID, \overline{K}_2^1) = (1011011, 0011010)$.
To recover the secret keys used during the first execution of the authentication protocol, *Adv* proceeds as follows:

- From $\mathbf{IDS}^2 = 0100000$, since he knows also $\mathbf{IDS}^1 = 1001110$, $ID = 1011011$, and $\overline{K}_1^1 = 1100000$, he computes
$$n_2^1 = 1101001.$$

Amount of Rshift	0	1	2	**3**	4	5	6
	1100000	0110000	0011000	0001100	0000110	0000011	1000001
	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$
	1101001	1101001	1101001	1101001	1101001	1101001	1101001
K_1^2	0001001	1011001	1110001	1100101	1101111	1101010	0101000
K_1^2 mod 7	2	5	1	**3**	6	1	5

Figure 14: Possible values for K_1^1.

- Then, from $\overline{K}_1^1 = Rot(K_1^1 \oplus n_2^1, K_1^1)$, since he knows $\overline{K}_1^1$ and n_2^1, then he computes some possible values for K_1^1 (see Figure 14.)

Also in this case, there is *only* one possible $K_1^1 = 1100101$. Finally, from $\mathbf{C} = 0101100, K_1^1 = 1100101, \overline{K}_2^1 = 0011010$, and $\overline{K}_1^1 = 1100000$, *Adv* computes

$$K_2^1 = [\mathbf{C} - (K_1^1 \oplus \overline{K}_2^1)] \oplus \overline{K}_1^1 = 1001101.$$

6 Experimental Results

We have implemented the protocol and the attacks in order to evaluate empirically their efficacy and efficiency. The following figures report the results we have obtained running 10000 tests. The first one, given in Figure 15, shows the distribution of the number of interactions *Adv* needs in order to de-synchronise the Reader and the Tag. The average number of iterations is 48.29.

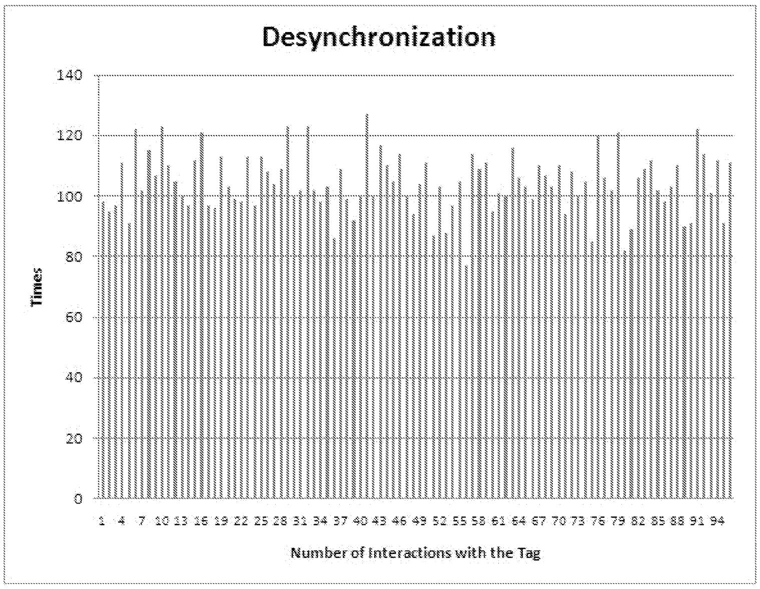

Figure 15: Desynchronisation attack: distribution.

The second one, given in Figure 16, shows the distribution of the number of interactions *Adv* needs for the preprocessing. The average number of iteractions is 119.21.

The third one, given in Figure 17, shows the distribution of the number of interactions *Adv* needs for the identity disclosure attack. The average number of iteractions is 140.77.

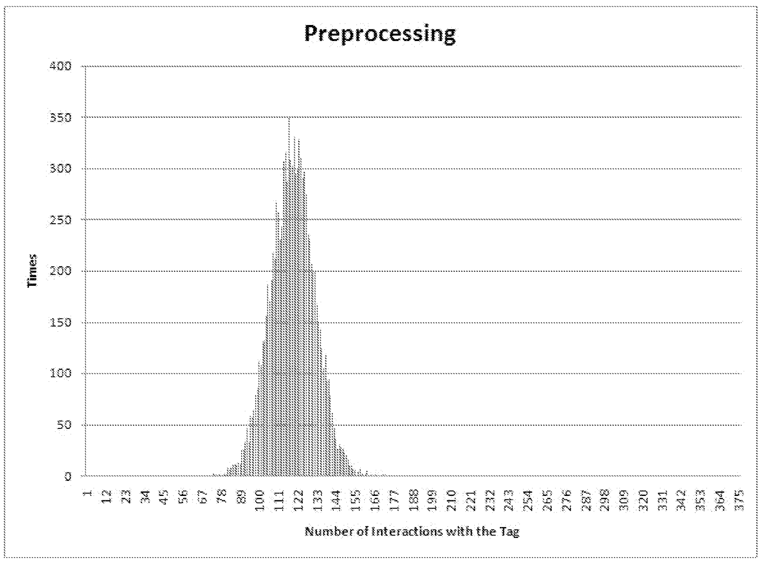

Figure 16: Preprocessing: distribution.

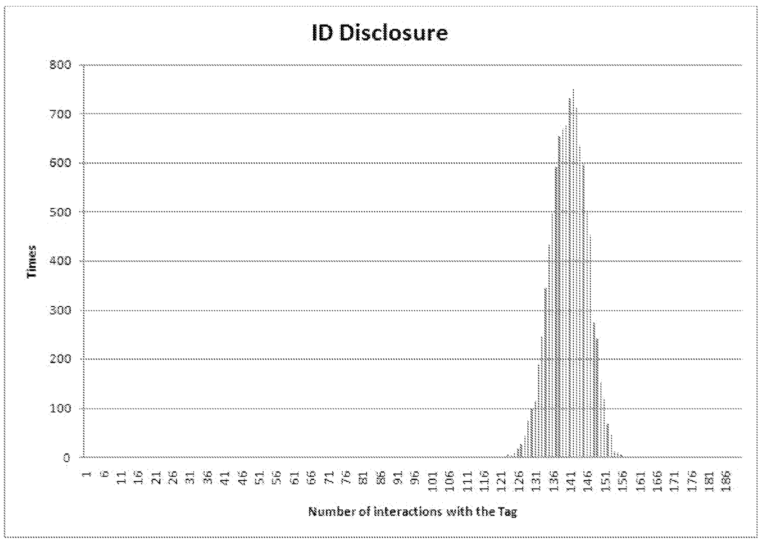

Figure 17: ID Disclosure Attack: distribution.

Notice that, the average number of interactions for the whole ID disclosure attack is 308.27

Finally, Figure 18 reports the average number of keys K_1^2 computed by Adv, the average number of pairs (K_1^2, K_2^2) consistent with the C-test, the average number of keys K_1^1, and the average number of possible sequences of keys $(K_2^3, K_1^3, K_2^2, K_1^2, K_2^1, K_1^1)$, consistent with the D-test.

Full Disclosure			
Avg. Num. K_1^2	Avg. Num. Pairs C-test	Avg. Num. K_1^1	Avg. Num. Seq. D-test
70.11	1.01	2.01	1

Figure 18: Resolution of ambiguities.

7 Conclusions

We have analysed **SASI** [2], a new ultra-lightweight authentication protocol, proposed to provide protection for RFID Tags. We have showed that the protocol presents vulnerabilities which can be used by an adversary who can interact with the Tag. All that is needed is an illegal Reader which can query the Tag and get replies.

We have described three attacks. A *de-synchronisation* attack, through which the adversary can break the synchronisation between the RFID Reader and the Tag. An *identity disclosure* attack, through which the adversary can compute the identity of the Tag. A *full disclosure* attack, which enables the adversary to retrieve all secret data stored in the Tag.

We have showed, analytically and through experiments, that the attacks are effective and efficient.

All properties the protocol was claimed to enjoy are not satisfied. Indeed, it is easy to check that properties from 1. to 5. and from 7. to 8. do not hold. Moreover, one of the scenarios described as harmless in property 6. (Resistance to replay attacks), i.e., resending the sequence $\mathbf{A}||\mathbf{B}||\mathbf{C}$ captured by eavesdropping an honest interaction between Tag and Reader, plays a key role in the proposed attack, in order to reset the Tag to a previous internal state.

Other researchers have also looked at the security of the **SASI** protocol. In [18], the authors have proposed two de-synchronisation attacks. Then, in [8], an identity disclosure attack which enables an adversary to compute a certain number of bits of the ID depending on the number of sessions he can eavesdrop, has been proposed. Finally, in [14], a probabilistic attack against the untraceability property has been described.

The recent history shows that all ultra-lightweight authentication protocols proposed have been broken through efficient attacks relatively soon after they have been published. Almost all of them provide informal security arguments to support the merits of the obtained protocols. The present paper confirms that sound security arguments should be used to support design strategies in deploying secure communication protocols. It might be used as a simple case study to show how, sometimes, from small vulnerabilities it is possible a full break of a cryptographic protocol.

Acknowledgement

We would like to thank Sandro Pecora for implementing the attacks.

References

[1] G. Avoine, *Bibliography on Security and Privacy in RFID Systems*, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA (available online at http://lasecwww.epfl.ch/~gavoine/rfid/, last update in Jun 2007).

[2] H. Chien, *SASI: A new Ultralightweight RFID Authentication Protocol Providing Strong Authentication and Strong Integrity*, IEEE Transactions on Dependable and Secure Computing, Vol. 4, N. 4, pp. 337–340, October-December 2007.

[3] H. Chien and C. Hwang, *Security of ultra-lightweight RFID authentication protocols and its improvements*, ACM SIGOPS Operating Systems Review, Vol. 41, Issue 4, pp. 83-86, July 2007.

[4] N. J. Hopper and M. Blum, *Secure Human Identification Protocols*, Proceedings of Asiacrypt 2001, Lecture Notes in Computer Science, Vol. 2248, pp. 52-66, 2001.

[5] H. Gilbert, M. J. B. Robshaw, and Y. Seurin, *HB: Increasing the Security and Efficiency of $HB+$*, Proceedings of Eurocrypt 2008, Lecture Notes in Computer Science, Vol. 4965, pp. 361-378, 2008.

[6] H. Gilbert, M. J. B. Robshaw, and Y. Seurin, *Good Variants of HB+ are Hard to Find*, Proceedings of Financial Cryptography 2008, Lecture Notes in Computer Science, Vol. 5143, pp. 156–170, 2008.

[7] H. Gilbert, M. J. B. Robshaw, and H. Sibert, *An active attack against HB+ a provably secure lightweight authentication protocol*, Electronics Letters, Vol. 41, N. 21, pp. 11691170, 2005.

[8] J. C. Hernandez-Castro, J. M. E. Tapiador, P. Peris-Lopez, and J-J. Quisquater, *Cryptanalysis of the SASI Ultralightweight RFID Authentication Protocol*, private communication, submitted for publication, Jun 2008.

[9] A. Juels, *The Vision of Secure RFID*, Proceedings of the IEEE, Vol. 95, No. 8, pp. 1507-1508, August 2007.

[10] A. Juels, R. Pappu, and S. Garfinkel, *RFID Privacy: An Overview of Problems and Proposed Solutions*, IEEE Security and Privacy, Vol. 3, No. 3, pp. 34-43, May-June 2005.

[11] A. Juels and S. Weiss, *Authenticating pervasive devices with human protocols*, Proceedings of Crypto 2005, Lecture Notes in Computer Science, Vol. 3126, pp. 293308, 2005.

[12] T. Li and R. Deng, *Vulnerability Analysis of EMAP-An Efficient RFID Mutual Authentication Protocol*, Proceedings of the The Second International Conference on Availability, Reliability and Security, pp. 238-245, 2007.

[13] T. Li, and G. Wang, *Security Analysis of Two Ultra-Lightweight RFID Authentication Protocols.* Proceedings of the 22-nd IFIP SEC 2007, May 2007.

[14] R. C.-W. Phan, *Cryptanalysis of a New Ultralightweight RFID Authentication Protocol - SASI*, IEEE Transactions on Dependable and Secure Computing, 19 June 2008. IEEE Computer Society Digital Library. Available at http://doi.ieeecomputersociety.org/10.1109/TDSC.2008.33

[15] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador and A. Ribagorda *LMAP: A Real Lightweight Mutual Authentication Protocol for Low-cost RFID Tags*, Proceedings of the Second Workshop RFID Security, July11-14, Graz University of Technology, 2006.

[16] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador and A. Ribagorda, *EMAP: An Efficient Mutual-Authentication Protocol for Low-Cost RFID Tags* OTM 2006 Workshop, Lecture Notes in Computer Science, Vol. 4277, pp. 352-361, 2006.

[17] P. Peris-Lopez, J. C. Hernandez-Castro, J. M. Estevez-Tapiador and A. Ribagorda, *M^2AP: A Minimalist Mutual-Authentication Protocol for Low-Cost RFID Tags*, Ubiquitous Intelligence and Computing, Lecture Notes in Computer Science, Vol. 4159, pp. 912-923, 2006.

[18] H. Sun, W. Ting, and K. Wang, *On the Security of Chien's Ultralightweight RFID Authentication Protocol*, eprint archieve, report 83, February 25, 2008.

[19] A. Shamir, *SQUASH - a New MAC With Provable Security Properties for Highly Constrained Devices Such As RFID Tags*, Proceedings of FSE 2008, Lecture Notes in Computer Science, Vol. 5086, pp. 144–157, 2008.