

# A Flyweight RFID Authentication Protocol

Mike Burmester<sup>1</sup> and Jorge Munilla<sup>2</sup>

<sup>1</sup> Department of Computer Science  
Florida State University, Tallahassee, FL 32306, USA  
`burmester@cs.fsu.edu`

<sup>2</sup> Departamento de Ingeniería de Comunicaciones  
Universidad de Málaga, Spain  
`munilla@ic.uma.es`

**Abstract.** In this paper we first discuss the security threats that have to be addressed when dealing with lightweight RFID protocols: in particular, privacy/integrity attacks that compromise the forward and backward security of tags. We then analyze some recently proposed EPCGen2 compliant protocols. Finally, we propose a lightweight RFID authentication protocol that supports session unlinkability with forward and backward security. The only cryptographic mechanism that this protocol uses is a synchronized pseudorandom number generator (RNG), that is shared with the backend Server. Authentication is achieved by using a few numbers (3 or 5) drawn from the RNG. The protocol is optimistic with constant key-lookup, and can easily be implemented on an EPCGen2 platform.

**Keywords:** RFID, authentication, privacy, forward and backward security, optimistic protocols, EPCGen2.

## 1 Introduction

Radio Frequency Identification (RFID) is a promising new technology that is widely deployed for supply-chain and inventory management, retail operations and more generally, automatic identification. The advantage of RFID over barcode technology is that it does not require direct line-of-sight reading. Furthermore, RFID readers can interrogate tags at greater distances, faster and concurrently. One of the most important advantages of RFID technology is that tags have read/write capability, allowing stored tag information to be altered dynamically.

To promote the adoption of RFID technology and support interoperability, EPCGlobal [10] has recently ratified the EPC Class 1 Gen 2 (EPCGen2) standard for RFID deployments. This defines a platform for RFID protocol interoperability, and supports basic reliability guarantees, provided by an on-chip 16-bit pseudo-random number generator (RNG) and a 16-bit Cyclic Redundancy Code (CRC16). The EPCGen2 standard is designed to strike a balance between cost and functionality, with less attention paid to security.

Several lightweight RFID authentication protocols that address security have been proposed in the literature. Most use hash functions [23, 20, 13, 2, 9, 18],

which are beyond the capability of low-cost tags and not supported by EPCGen2. Some RFID protocols use pseudorandom functions [7, 25, 4], or RNGs (as in [4, 11]), mechanisms supported by EPCGen2, but these are not optimized for EPCGen2 compliance. We refrain from a detailed review of the literature for RFID security which is quite extensive, and refer the reader to a comprehensive repository available online at [1].

Recently five RFID authentication protocols specifically designed for compliance with EPCGen2 have been proposed [8, 21, 24, 22, 11]. These combine the CRC16 of EPCGen2 with its 16-bit RNG to hash, randomize and link protocol flows, and prevent cloning, impersonation and denial of service attacks. In this paper we analyze these protocols and show that they do not achieve their security goals, or are unduly complex. One may argue that, because EPCGen2 supports only a very basic RNG, any RFID protocol that complies with it is potentially vulnerable, for example to ciphertext-only attacks that exhaust the range of the components of protocol flows. While this is certainly the case, such attacks may be checked by refreshing key material and/or constraining the application (e.g., the life-time of tags).

In this paper we are concerned with the security of lightweight low cost RFID protocols. In particular, their *forward* and *backward security* [3]. The former protects past tag interrogations from being linked to a captured tag. Tags are not tamper-resistant, and therefore the adversary can access the private data of a captured tag. Backward security protects future tag interrogations from traffic analysis (correlation) attacks in which the adversary uses the information leaked by tags to find their inner state. Such attacks exploit the fact that the state of lightweight tags has (typically) low entropy.

Our main contribution in this paper is to propose a lightweight mutual authentication RFID protocol that supports session unlinkability, forward and backward security. For this protocol tag authentication is achieved by drawing numbers from a RNG, shared with the backend Server. The protocol is optimistic with constant key-lookup, and can easily be implemented on an EPCGen2 platform.

We note that all cryptographic protocols use shared keys (symmetric or asymmetric) to support security. In threshold cryptography shared functionalities are used to distribute cryptographic applications. Closer to our application, quantum cryptography uses quantum mechanics to support private channels. For these channels, any attempt by the adversary to measure a communicated “qubit” introduces detectable anomalies. Quantum channels can therefore be regarded as shared channels with “no clone” qubits. In our protocol, shared synchronized RNGs generate sequences of numbers that “cannot be cloned”.

The rest of this paper is organized as follows. Section 2 discusses RFID deployments and the EPCGen2 standard. Section 3 analyzes some recently proposed RFID protocols that are EPCGen2 compliant. In Section 4 we propose an optimistic lightweight RFID authentication protocol that supports session unlinkability with forward/backward security, and consider an EPCGen2 implementation. In Section 5 we discuss its security.

## 2 RFID Deployments

### 2.1 Threat and Attacks

There are several general types of adversarial attacks on RFID deployments. Below we list the more important ones.

1. *Tag disabling (an availability attack)*:  $\mathcal{A}$  causes tags to assume a state from which they can no longer function.
2. *Tag cloning (an integrity attack)*:  $\mathcal{A}$  captures the identifying information of a tag.
3. *Tag tracking (a privacy attack)*:  $\mathcal{A}$  traces tags from their protocol flows.
4. *Replay (an integrity attack)*:  $\mathcal{A}$  uses a tag's response to a Reader's challenge to impersonate the tag.
5. *Offline man-in-the-middle attacks (an integrity attack)*:  $\mathcal{A}$  interposes between a tag and a Reader and exchanges their (possibly modified) messages.

There are also attacks that are usually excluded from the security model used, such as *power analysis* (or *side-channel*) attacks [17], and *man-in-the-middle relay attacks* [16]. Sometimes these attacks may be prevented by using “out of the system” protection mechanism. In this paper we are specially concerned with two types of attack that target low cost RFID tags: attacks that disambiguate *past* tag interrogations, and attacks that compromise *future* tag interrogations. To secure against such attacks we use forward and backward security mechanisms.

### 2.2 Security Requirements

**Authentication.** *Client authentication* is a process in which one party, the Server  $\mathcal{S}$ , is assured of the identity of another party, the client (a tag  $\mathcal{T}$ ), by acquiring corroborative evidence. We have *anonymous* client authentication when the identity of  $\mathcal{T}$  remains private to third parties that may eavesdrop on the communication or invoke the protocol and interact with the parties directly. We have *mutual* authentication if both  $\mathcal{S}$  and  $\mathcal{T}$  are authenticated. In our protocol the Server is *implicitly* authenticated: that is, the assurance for tags is only implicit.

**Session unlinkability.** Two interrogations of a tag  $\mathcal{T}$  cannot be linked if, either the first one completed successfully, or an intermediate interrogation of  $\mathcal{T}$  completed successfully.

**Forward Security.** Past tag outputs, prior to refreshment, look random to an adversary even if the adversary can access the internal state of the tag after it is refreshed.

**Backward Security.** Future tag outputs, after refreshment, look random to an adversary even if the adversary can access the state of the RNG of the tag (e.g., by analyzing its outputs) before it is refreshed. In particular, a tag  $\mathcal{T}$  whose RNG is compromised will recover after its RNG is refreshed, and its outputs will look random. We will also consider *weak backward security*, for which the adversary cannot impersonate  $\mathcal{T}$  after its RNG is refreshed, but the tag  $\mathcal{T}$  never recovers (it is de-synchronized).

### 2.3 The EPCGen2 Standard

The EPC Global UHF Class-1 Generation-2 standard (EPCGen2), defines the physical and logical requirements for a passive-backscatter, Interrogator-talks-first, radio-frequency identification system operating in the 860 - 960 MHz range. The protocol defines two layers: a physical layer and a Tag-identification layer. The system comprises Interrogators (Readers), and Tags. An Interrogator manages Tag populations using three basic operations: *Select* —the operation of choosing a Tag population, *Inventory* —the operation of identifying Tags, and *Access* —the operation of reading from and/or writing to a Tag.

The Inventory protocol has (at least) four passes that involve: a *Query*, a 16-bit number  $RN16$ , an acknowledgment  $ACK(16)$  and the Tag’s identifying data, *EPC-data*. The Interrogator first starts by sending a *Query* command that includes a parameter  $Q \in [0 : 15]$  —a random-slotted collision algorithm (“Q-protocol”) is used to singulate tags. Tags that receive *Query* load a random  $Q$ -bit number into a slot counter, and decrease this counter whenever they receive the command *QueryRep*. When their counter is zeroed, Tags send a number  $RN16$  to the Interrogator. When the Interrogator detects a reply from a Tag, it sends an acknowledgement  $ACK(RN16)$ , which requests from the tag its *PC* (protocol control), *EPC* (electronic product code), and *CRC16*. If the Tag does not receive a valid  $ACK(RN16)$  (possibly because of a collision), it transitions to its initial state and the process is repeated.

For security, link cover-coding can be used to obscure information during Reader to Tag transmissions. To cover-code a data, an Interrogator first requests a random number from the Tag. Then, the Interrogator performs a bit-wise XOR of the data with this random number, and transmits the result (cover coded or ciphertext) to the Tag. Tags may also store a 32-bit Kill Password, and a 32-bit Access Password, and implement a 16-bit pseudo-random number generator (RNG), and a 16-bit Cyclic Redundancy Code (CRC16). CRCs are error-detecting codes that check (non-malicious) errors caused by faults during transmission. Observe that CRC16 is an additive operator with strong linearity aspects (the modulo operator is homomorphic), and therefore its use as a cryptographic tool is not appropriate.

The probability that the adversary guesses the next number of a RNG increases with the number of outcomes observed. Consequently correlation and exhaustive search attacks get easier as more numbers are drawn. This issue is not addressed adequately by EPCGen2. The standard specifies that a drawn  $RN16$  is not predictable with probability better than 0.025%, given the outcomes of prior draws. This bound is very crude: it is too high in the case when only one number is drawn, and too low when many numbers are drawn (e.g. more than a cycle of the RNG). In general we have to make certain that the entropy of a RNG is sufficiently large and/or regularly refreshed to prevent correlation attacks and/or exhaustive search attacks. We refer the reader to [4] for further discussion regarding the RNG of EPCGen2.

### 3 An Analysis of Recently Proposed EPCGen2 Protocols

We consider five recently proposed EPCGen2 compliant protocols and show that they either fall short of their claimed security, have weaknesses that may be exploited by an adversary, or are unduly complex.

1. The Chen-Deng protocol [8]. This is subject to a replay attack because the flows of the Reader and tag use independent randomness (for details see [6]).
2. The Sun-Ting protocol Gen2<sup>+</sup> [24]. This is also subject to a replay attack because only the tag provides randomness (for details see [6]).
3. The Qingling-Yiju-Yonghua protocol [21]. This protocol uses CRC16 as a cipher. So private information can easily be manipulated, and only one eavesdropped interrogation is needed to clone a tag (for details see [6]).
4. Seo-Baek propose two protocols [22].
  - (a) The first is subject to a replay attack (causing de-synchronization) because tag authentication does not involve any randomness from the Reader. Only one eavesdropped interrogation is needed. Again CRC16 is used as a cipher, so private information can be manipulated.
  - (b) The second is also subject to a replay attack because the randomness of the flows is determined entirely by the tag. Only one previous impersonation of a Reader (sending a *query*) is needed.
5. The Choi-Lim anti-cloning protocol [11]. In this protocol each tag  $\mathcal{T}$  shares three private 32-bit values with the Server  $\mathcal{S}$ : a kill password  $PW_{kill}$ , an access password  $PW_{access}$  and a tag serial number  $T_{sn}$ . Below we describe a simplified version:
  - (a)  $\mathcal{S} \Rightarrow \mathcal{R} \rightarrow \mathcal{T}$ :  $Q$ , a query.  
 $\mathcal{T}$ : Select a 32-bit random number  $R_t$  and:
  - (b)  $\mathcal{T} \rightarrow \mathcal{R} \Rightarrow \mathcal{S}$ :  $M_1 = R_t \oplus PW_{kill}$ .  
 $\mathcal{S}$ : Select a 32-bit random number  $R_r$  and:
  - (c)  $\mathcal{S} \Rightarrow \mathcal{R} \rightarrow \mathcal{T}$ :  $M_2 = R_r \oplus PW_{access}$  and  $M_3 = RNG(R_t \oplus R_r) \oplus PW_{access}$ .  
 $\mathcal{T}$ : get  $R_r$  from  $M_2$ . Compute  $RNG(R_t \oplus R_r)$  and check  $M_3$ . If it is correct:
  - (d)  $\mathcal{T} \rightarrow \mathcal{R} \Rightarrow \mathcal{S}$ :  $M_4 = RNG(RNG(R_t \oplus R_r)) \oplus T_{sn}$ .  
 $\mathcal{S}$ : check that  $M_4$  is correct. If it is correct, accept  $\mathcal{T}$  as an authorized tag.

This protocol has two weaknesses: (a) the Reader can be impersonated, and (b) it is subject to a *related-key attack* [4]. For the impersonation attack, the adversary  $\mathcal{A}$  first eavesdrops on an interrogation to get:  $Q, M_1, M_2, M_3, M_4$ , and then impersonates the Reader  $\mathcal{R}$  as follows: when  $\mathcal{T}$  sends a new  $M'_1$ ,  $\mathcal{A}$  computes  $M'_2 = M_2 \oplus M'_1 \oplus M_1$ , and  $M'_3 = M_3$ , and sends these to  $\mathcal{T}$ . These are clearly valid. Although the Choi-Lim protocol does not claim mutual authentication, if this service is not provided, it is unduly complex —see e.g. [4].

For the related-key attack, observe that the adversary can obtain “ciphertexts”  $M4 (= RNG(K \oplus N_i) \oplus T_{sn})$  and “plaintexts”  $M3 (= N_i)$  that are related by the key  $PW_{access} (= K)$ . Note also that the number of the plaintexts-ciphertexts pairs is not bounded because the adversary can impersonate the Reader (attack (a)).

## 4 A Flyweight RFID Authentication Protocol

Our protocol uses a synchronized RNG that can be refreshed by the Reader. RNGs are finite state machines with two distinguished components: *state* and *generate*. To *draw* a number from the RNG, algorithm *generate* uses *state* to generate a new value for *state* and an output number. Refreshing a RNG involves updating *state* with fresh (high entropy) randomness —see Figure 1. In our case this randomness is provided by the Reader through the cryptographic function *refresh*.

RNGs are refreshed to ensure resilience against: (a) traffic analysis attacks that exploit the correlation between successive numbers drawn from a RNG (*state entropy leakage*) and, (b) impersonation attacks resulting once the state of the RNG is fully compromised. That is: to ensure that the adversary cannot guess the next output with probability better than a certain threshold (a correlation attack), or use an exhaustive analysis of all possible values of *state* that produce the tag’s output, and to restrict the impact of a compromised *state* until it is next refreshed. For a detailed discussion on security issues of RNGs, see [3],[15].

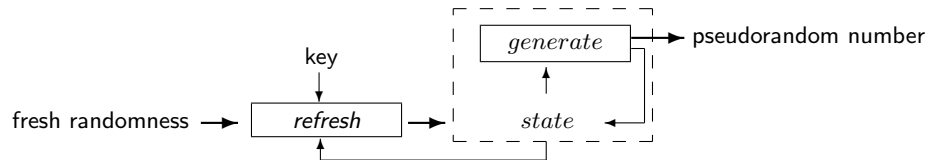


Fig. 1. Refreshing the *state* of a RNG.

### 4.1 The Protocol

Each tag  $\mathcal{T}$  shares with the backend Server  $\mathcal{S}$  a synchronized RNG (same algorithm, key, seed), say  $g_{tag} = g_{tag}(state)$ .  $\mathcal{T}$  and  $\mathcal{S}$  are mutually authenticated by exchanging either three (optimistic case), or five consecutive numbers from  $g_{tag}$ . Five numbers are required only when the interrogation is interrupted (i.e., when the first number has already been used: *alarm* is ON). The security of the protocol is based on the fact that: (a) random numbers drawn from a RNG cannot be predicted by the adversary, and (b)  $\mathcal{T}$  and  $\mathcal{S}$  are synchronized at all times. Synchronization is guaranteed by making certain that  $\mathcal{T}, \mathcal{S}$  always share at least one number. The protocol supports mutual authentication and a certain

degree of privacy (session unlinkability), with forward and backward security, and is provably secure, as we shall see.

Each tag  $\mathcal{T}$  stores in non-volatile memory two numbers, its identifier  $ID_{tag}$ ,  $g_{tag}$  (the current state), a key used for refreshing  $K^r$ , and a 1-bit flag  $cnt$ :  $(RN_1, RN_2; ID_{tag}, g_{tag}, K^r, cnt)$ . The Server  $\mathcal{S}$  stores in a database for each  $\mathcal{T}$ , a list of six numbers,  $ID_{tag}$ ,  $g_{tag}(state)$ ,  $K^r$  and a 1-bit flag  $cnt'$ :

$$DB = \{(RN_1^{cur}, RN_1^{next}, RN_2, RN_3, RN_4, RN_5; ID_{tag}, g_{tag}, K^r, cnt')\}.$$

The lists in  $DB$  are doubly indexed by  $RN_1^{cur}$  and  $RN_1^{next}$  respectively. To initialize the values of its variables, the tag draws two successive values  $RN_1, RN_2$  from  $g_{tag}$  and sets  $cnt \leftarrow 0$ . The Server  $\mathcal{S}$ , sets  $cnt' \leftarrow 0$ , draws six successive numbers from the RNG of each tag and assigns their values to the variables in the tags' lists:

$$RN_1^{cur}, RN_2, RN_3, RN_4, RN_5, RN_1^{next} \text{ (in this order).}$$

To update these values,  $\mathcal{S}$  uses the function *update* in which:  $cnt' \leftarrow 0$ ,  $RN_1^{cur} \leftarrow RN_1^{next}$  and the five values  $RN_2, RN_3, RN_4, RN_5, RN_1^{next}$ , are updated by drawing new numbers from  $g_{tag}$ . In the protocol, each  $\mathcal{T}$  shares at all times with  $\mathcal{S}$  at least one number: either  $RN_1 = RN_1^{cur}$  or  $RN_1 = RN_1^{next}$ .

### Protocol

1.  $\mathcal{R} \rightarrow \mathcal{T}$  : Query  
 $\mathcal{T}$  : Set  $alarm \leftarrow cnt$ ,  $cnt \leftarrow 1$ , and broadcast  $RN_1$ .
2.  $\mathcal{T} \rightarrow \mathcal{R} \Rightarrow \mathcal{S}$  :  $RN_1$   
 $\mathcal{S}$  : Check in  $DB$   
 If  $RN_1 = RN_1^{cur}$  for an item in  $DB$  then set  $alarm' \leftarrow cnt'$ ,  $cnt' \leftarrow 1$  and broadcast  $RN_2$ .  
 Elseif  $RN_1 = RN_1^{next}$  for an item in  $DB$  then set  $alarm' \leftarrow 0$ , *update* and broadcast  $RN_2$ .  
 Else abort.
3.  $\mathcal{S} \Rightarrow \mathcal{R} \rightarrow \mathcal{T}$  :  $RN_2$   
 $\mathcal{T}$  : Check  $RN_2$ .  
 If  $RN_2$  is correct then draw five successive numbers from  $g_{tag}$ , assign them to the variables  $RN_3, RN_4, RN_5$  (volatile),  $RN_1, RN_2$ , and set  $cnt \leftarrow 0$ .  
 If  $alarm = 0$  then broadcast  $RN_3$ .  
 Else broadcast  $RN_4$ .  
 Else abort.
4.  $\mathcal{T} \rightarrow \mathcal{R} \Rightarrow \mathcal{S}$  :  $RN^*$ , which is either  $RN_3$  ( $alarm = 0$ ) or  $RN_4$  ( $alarm = 1$ )  
 $\mathcal{S}$  : Check the received value  $RN^*$ .  
 If  $RN^* = RN_3$  and  $alarm' = 0$  then *update*, and ACCEPT the tag as the authorized  $\mathcal{T}$ .  
 Elseif  $RN^* = RN_4$  then set  $RN_5^{cur} \leftarrow RN_5$ , broadcast  $RN_3$  and *update*.  
 Else abort.

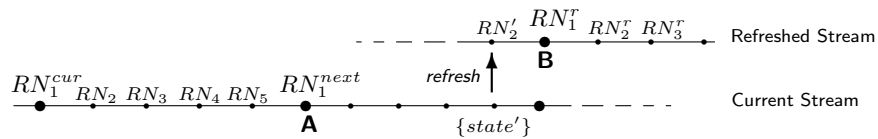
5.  $\mathcal{S} \Rightarrow \mathcal{R} \rightarrow \mathcal{T} : RN_3$   
 $\mathcal{T}$ : Check  $RN_3$ .  
 If it is valid ( $alarm = 1$  and  $RN_3$  is correct) then broadcast  $RN_5$ .  
 Else abort.
6.  $\mathcal{T} \rightarrow \mathcal{R} \Rightarrow \mathcal{S} : RN_5$   
 $\mathcal{T}$ : Check  $RN_5$ .  
 If  $RN_5 = RN_5^{cur}$ , then ACCEPT the tag as the authorized  $\mathcal{T}$ .  
 Else abort.

This protocol is *optimistic* because an interrogation needs only three numbers to be drawn when the adversary is passive. If an active adversary  $\mathcal{A}$  tries to replay flows, this will cause  $\mathcal{T}$  to activate *alarm*, and two additional numbers will be needed (Pass 5 and Pass 6). It must be noted that the numbers  $RN_3$ ,  $RN_4$  and  $RN_5$  are always fresh (never sent more than once), because at this point  $\mathcal{T}$  and  $\mathcal{S}$  have already updated their values for the next interrogation.  $\mathcal{S}$  needs to perform at most two lookups in  $DB$  (for  $RN_1^{curr}$ , and  $RN_1^{next}$ ) to identify  $\mathcal{T}$ .

Although we have not included timers to simplify the presentation, these have to be included in any implementation so that sessions can be closed. Thus, parties will abort the process if no response is received within a certain time  $t_h$  after sending a challenge. These timers can prevent certain kinds of active attack, but it is assumed that they are not precise enough to avoid “on-line man-in-the-middle relay attacks”. Thus, any active attack that involves relaying flows between the parties faster than  $t_h$  will succeed. It is considered as a “on-line man-in-the-middle relay attack” and the protocol will be subject to it. Naturally, the more accurate  $t_h$  is, the harder these attacks become [19].

## 4.2 Refreshing a RNG

RNGs need to be refreshed for resilience. In our protocol this randomness is provided by the Reader when needed: i.e., when the probability that the *state* of the RNG of a tag may be compromised is higher than a certain threshold (it will depend on the specific features of the implemented RNG). Some new non-volatile variables are required. The Server requires to store an intermediate *state'* of the RNG, two numbers  $RN_1^{start}$  and  $RN_1^{end}$  which mark the start and end of a refreshment, and a number  $R$  that provides the randomness. We illustrate in Figure 2 the effect of refreshing a stream of numbers generated by a shared RNG. The Server uses a 1-bit trigger *refresh* to refresh a tag.



**Fig. 2.** Transition from the current stream to the refreshed stream. **A** and **B** mark the start ( $RN_1^{start}$ ) and the end ( $RN_1^{end}$ ) of the refreshing.

Next we describe the two passes of the protocol that need to be modified.

### Refresh RNG

$\mathcal{S}$  decides to refresh the *state* of  $\mathcal{T}$ : Set *refresh* ON,  $RN_1^{start} \leftarrow RN_1^{next}$ , get *state'* (by drawing four numbers from  $g_{tag}$ ) and generate a random number  $R$ .

2!  $\mathcal{T} \rightarrow \mathcal{R} \Rightarrow \mathcal{S}$  :  $RN_1$

$\mathcal{S}$  : Check if  $RN_1$  is in *DB* and *refresh* is ON (Else, go to normal execution).

If  $RN_1 = RN_1^{end}$  then set *refresh* OFF and go to normal execution.

Elseif  $RN_1 = RN_1^{start}$ :

If  $RN_1 = RN_1^{next}$  then *update*.

Set  $state^{ref} \leftarrow refresh(K^r; R, state')$ , draw two numbers from  $g_{tag}(state^{ref})$  and assign their values to  $RN_2', RN_1^{next}$ .

Set  $RN_1^{end} \leftarrow RN_1^{next}$ ,  $alarm' \leftarrow cnt'$ ,  $cnt' \leftarrow 1$ . Broadcast  $R$  and  $RN_2'$ .

Else go to normal execution

3!  $\mathcal{S} \Rightarrow \mathcal{R} \rightarrow \mathcal{T}$  :  $R, RN_2'$

$\mathcal{T}$  : If the format corresponds to “refresh” (Else, go to normal execution):

Store the current *state* of  $g_{tag}$ , draw three numbers from  $g_{tag}(state)$ , and assign their values to  $RN_3, RN_4, RN_5$ .

Set  $state^{ref} \leftarrow refresh(K^r; R, state')$ , where *state'* is the current state of  $g_{tag}$ .

Draw one number from  $g_{tag}(state^{ref})$  and assign its values to:  $RN_2'$ .

If it is correct, draw two more numbers and assign their values to  $RN_1, RN_2$ .

The protocol continues normally (broadcast  $RN_3$  or  $RN_4$  according to the value of *alarm*).

Else restore the state of  $g_{tag}$  to *state* and abort.

Another possible way to refresh the RNG of a tag with entropy from the Reader involves flipping the order of the numbers drawn (e.g., flipping  $RN_2$  and  $RN_3$ ), so that one bit of *state* (determined by a counter) is refreshed. This would support resilience against correlation attacks if the information leaked when five numbers are drawn from a RNG is no more than one bit. We shall discuss the security of our protocol in Section 5.

### 4.3 EPCGen2 Implementation

The EPCGen2 Protocol has four passes for identification (acknowledged state) that involve: a *Query*, a number  $RN16$ , an acknowledgment  $Ack(RN16)$  and *EPCdata*. To enable authentication and session unlinkability, we replace  $RN16$  by  $RN_1$ ,  $Ack(RN16)$  by  $RN_2$  and *EPCdata* by  $RN_3$  (optimistic case). If  $RN_1$  has been used previously (*alarm* is ON), then two more numbers have to be exchanged.

To ensure that it is hard to find the state of an EPCGen2 RNG by using an exhaustive search over all possible state values that produce a given sequence of numbers, the entropy of the state of RNG must be sufficiently large. If a 32-bit state with refreshment provides adequate security then we may use the following simple implementation:  $refresh(K^r; R, state) = g(K^r \oplus R \oplus state)$ , where  $R$  is a

32-bit random number and  $K^r$  a 32-bit key (e.g.  $K_{access}$ ). Alternatively we can use  $F_{g_{tag}}(K^r \oplus R \oplus state)$ , where  $F_{g_{tag}}$  is the pseudo-random function defined by  $g_{tag}$  [12].

We have not discussed collisions occurring during Inventory in our protocol, for simplicity. We note here that bits of  $RN_1$  can be used to load the slot counter with the  $Q$ -bit number. If a collision occurs and new bits of  $RN_1$  have to be used, the *alarm* is set to 1 and six passes are required. If collisions become a serious problem other solutions can be employed, such as, for example, using bits of  $RN_2$  or generating an extra number.

## 5 Security

We adopt the Byzantine threat model. The adversary  $\mathcal{A}$  and all other parties (honest and adversarial) are represented by probabilistic Turing machines.  $\mathcal{A}$  controls the delivery schedule of all communication and may eavesdrop or modify messages.  $\mathcal{A}$  may also initiate new communication and directly interact with honest parties: Readers  $\mathcal{R}$  or tags  $\mathcal{T}$ . However the channels that link the backend Server  $\mathcal{S}$  to (honest) Readers  $\mathcal{R}$  are assumed to be secure.

### 5.1 The security framework

The universal composability (UC) framework specifies a particular approach to security proofs for protocols  $\pi$ , and guarantees that proofs that follow this approach remain valid if  $\pi$  is, say composed with other protocols (modularity) and/or under arbitrary concurrent protocol executions (including with itself). The UC framework defines a *real-world simulation*, an *ideal-world simulation*, an *emulation* that translates protocol runs of  $\pi$  from the real-world to the ideal-world, and an interactive environment  $\mathcal{Z}$  that captures whatever is external to the current protocol execution. The components of a UC formalization are:

1. A *mathematical model* of real executions of protocol  $\pi$  in which the honest parties execute as specified, whereas adversarial parties can deviate from  $\pi$  arbitrarily. These are controlled by the adversary  $\mathcal{A}$  that has full knowledge of the state of adversarial parties, and can arbitrarily schedule the communication channels and activation periods of all parties, and interact with the environment  $\mathcal{Z}$  in arbitrary ways.
2. An *idealized model* of executions, where the security properties of protocol  $\pi$  depend on the behavior of an *trusted functionality*  $\mathcal{F}_\pi$ .  $\mathcal{F}_\pi$  controls the ideal model adversary  $\mathcal{S}_\mathcal{A}$  so that it reproduces as faithfully as possible the behavior of  $\mathcal{A}$ .
3. A proof that, for each adversary  $\mathcal{A}$  there is a simulator  $\widehat{\mathcal{S}}$  that translates real-world runs of  $\pi$  in the presence of  $\mathcal{A}$  into ideal-world protocol runs of  $\pi$  in the presence of  $\mathcal{S}_\mathcal{A}$  such that, no environment  $\mathcal{Z}$  can distinguish (with better than negligible probability) whether  $\mathcal{A}$  is communicating with an instance of  $\pi$  in the real-world or  $\mathcal{S}_\mathcal{A}$  is communicating with  $\mathcal{F}_\pi$  in the ideal-world.

## 5.2 An informal security proof

Below we state our main result and sketch an outline of the proof. We note that our proof is significantly simplified because the protocol flows are pseudorandom numbers drawn from a RNG, and a symmetric key trust model with star topology is used.

**Theorem 1.** *The protocol in Section 4 guarantees: availability, authentication, session unlinkability, forward security, and weak backward security in the UC framework provided that the RNG used is a cryptographically secure pseudorandom number generator.*

**Proof.** The structure of our proof is similar to the proof for O-RAKE (Section 7.2, [5]). We adapt it appropriately for our requirements.

The functionality  $\mathcal{F}_{auth}$  of the protocol is captured by the specifications in Section 2.2. In particular:

1. Availability requires that the Server and tags be always synchronized.
2. Authentication (mutual) requires that each party corroborate the values produced by the other in terms of their shared private key (the *state* of their shared RNG).
3. Session unlinkability requires that the adversary cannot link tag interrogations in different sessions.
4. Forward security requires that past protocol flows, prior to refreshment, look random even if the internal state of a tag gets compromised after refreshment.
5. Weak backward security requires that a tag whose RNG gets compromised cannot be impersonated after the Reader requests a refreshment.

To access  $\mathcal{F}_{auth}$  the adversary uses commands of type: INITIATE (start a new session), GET( $flow, i$ ) (get the  $i$ -th flow), SEND( $flow, i$ ) (send the value of  $flow$  to the  $i$ -th flow-entity), IMPERSONATE(tag) (reveal the internal state of tag) and COMPROMISE(tag) (reveal the *state* of the RNG of tag). The simulator  $\mathcal{S}$  performs the following actions to emulate the real world.

- $\mathcal{S}$  simulates copies of the parties  $\widehat{server}$  and  $\widehat{tag}_i$ , initialized by  $\mathcal{Z}$ , and activates an adversary  $\widehat{\mathcal{A}}$ .
- $\mathcal{S}$  adds or removes keys to a database  $\widehat{D}$  of  $\widehat{server}$  that contains persistent keys of adversarially controlled tags as well as transient keys of honest tags.
- $\mathcal{S}$  faithfully translates real world messages into their ideal world counterparts for all protocol parties, including the adversary.
- $\mathcal{S}$  simulates the interactions with  $\mathcal{Z}$ , i.e., the externally visible part of the protocol. More specifically, it invokes  $\mathcal{F}_{ake}$  with message ACCEPT( $s, s'$ ) when the real world adversary forwards unmodified inputs between honest tags and the server, and invokes IMPERSONATE( $tag$ ) when the real-world adversary succeeds in authenticating adversarially controlled tags, respectively.
- $\mathcal{S}$  prevents honest parties from outputting ACCEPT( $\cdot, \cdot$ ) in the ideal world when  $\mathcal{A}$  tampers with messages created by honest tags.

In the UC framework all parties involved in a protocol execution instance share the same session identifier  $sid$ . In the protocol the receiving party of any message in the session  $sid$  is activated next. The security proof cannot make any assumptions about extraneous knowledge that may or not be available to  $\mathcal{Z}$  through interactions with other entities (including other instances of the protocol).

The real world executions of the protocol involve the (pseudorandom) numbers drawn from the RNG of the queried tags or the Server. The idealized world executions involve random numbers selected by  $\mathcal{F}_{auth}$  to reproduce as faithfully as possible the behavior of the protocol in the presence of  $\mathcal{A}$ .  $\mathcal{F}_{auth}$ . We have UC-security if no environment  $\mathcal{Z}$  can distinguish real-world from ideal-world simulations. For more details on security proofs of RFID deployments in the UC framework, the reader is referred to [25].

The proof is based on showing that the environment  $\mathcal{Z}$  cannot distinguish protocol execution in the presence of a real-world adversary  $\mathcal{A}$  in the real-world from those in the ideal-world in the presence of an ideal world adversary  $\mathcal{S}_{\mathcal{A}}$ . This involves emulating real-world actions in the ideal-world. For this purpose we simulate copies  $\mathcal{S}_{\mathcal{A}}$ , of the real adversary,  $\widehat{server}$ , of the real server,  $\widehat{tag}$ , of real tags, and emulate the interactions of the protocol with  $\mathcal{Z}$ , in particular its invocations of  $\mathcal{F}_{auth}$ . For our protocol it is straightforward to show that any interrogation in the real-world that is accepted by the Server  $\mathcal{S}$  is also accepted in the idealized world by the functionality  $\mathcal{F}_{auth}$  because:

1. At all times each tag shares at least one number with the Server (availability).
2. If the Server accepts the tag then a fresh flow of numbers must have been used (authentication).
3. If for any two interrogations either the first one completed successfully, or an intermediate interrogation completed successfully, then the tag will have updated the values of the numbers it stores.
4. After refreshing the RNG of a tag  $\mathcal{T}$ , its earlier states are not accessible even if the current internal state of  $\mathcal{T}$  is known.
5. After refreshing the RNG of a tag  $\mathcal{T}$ , its next states are not accessible even if the state of RNG prior to refreshment is known.  $\square$

We can extend our protocol to capture (full) backward security if the Server  $\mathcal{S}$  keeps a record of the *initial refresh state* of the RNG of each tag. If the RNG of a tag gets compromised, then only the actual tag will be able to refresh the state of its RNG and get authenticated by  $\mathcal{S}$ . In this approach to simulate compromised tags additional *DB* lookups will be needed.

### Acknowledgement

Research partly supported by the Spanish Ministry of Science and Innovation and the European FEDER Funds, under Project TIN 2008-02236/TSI.

### References

1. AVOINE, G. <http://www.avoine.net/rfid/>.

2. AVOINE, G., AND OECHSLIN, P. A scalable and provably secure hash based RFID protocol. *Proc. IEEE Int. Workshop on Pervasive Computing & Communication Security (PerSec 2005)*, IEEE Computer Society Press.
3. BARAK, B., AND HALEVI, S. A model and architecture for pseudo-random generation with applications to /dev/random. *ACM Conf. on Computer and Communications Security (2005)*, V. Atluri, C. Meadows, and A. Juels, Eds., ACM, pp. 203–212.
4. BURMESTER, M., AND DE MEDEIROS, B. The Security of EPC Gen2 Compliant RFID Protocols. *ACNS (2008)*, S. M. Bellovin, R. Gennaro, A. D. Keromytis, and M. Yung, Eds., *Lecture Notes in Computer Science*, vol. 5037, pp. 490–506.
5. BURMESTER, M., VAN LE, T., DE MEDEIROS, B., AND TSUDIK, S. Provably Secure Ubiquitous Systems: Universally Composable RFID Authentication Protocols. *ACM Transactions on Information and System Security (TISSEC) (2009)*.
6. BURMESTER, M., DE MEDEIROS, B., MUNILLA, J., AND PEINADO, A. Secure EPC Gen2 Compliant Radio Frequency Identification. E-print #2009/147, International Association for Cryptological Research, 2009.
7. BURMESTER, M., VAN LE, T., AND DE MEDEIROS, B. Provably secure ubiquitous systems: Universally composable RFID authentication protocols. *Proc. 2nd IEEE CreateNet Int. Conf. on Security and Privacy in Communication Networks (SECURECOMM 2006)*, IEEE Press.
8. CHEN, C.-L., AND DENG, Y.-Y. Conformation of EPC Class 1 Generation 2 standards RFID system with mutual authentication and privacy protection. *Engineering Applications of Artificial Intelligence, Elsevier, In Press (2009)*.
9. DIMITRIOU, T. A secure and efficient RFID protocol that can make big brother obsolete. In *Proc. Intern. Conf. on Pervasive Computing and Communications, (PerCom 2006) (2006)*, IEEE Press.
10. EPC GLOBAL. EPC Tag Data Standards, <http://www.epcglobalinc.org>
11. EUN YOUNG CHOI, D. H. L., AND LIM, J. I. Anti-cloning protocol suitable to epcglobal class-1 generation-2 rfid systems. *Computer Standards & Interfaces Available online, In press, Corrected Proof (2008)*.
12. GOLDREICH, O., GOLDWASSER, S., AND MICALI, S. How to construct pseudorandom functions. *Journal ACM* 33, 4 (1986).
13. HENRICI, D., AND MÜLLER, P. M. Hash-based enhancement of location privacy for radio-frequency identification devices using varying identifiers. *Proc. IEEE Int. Conf. on Pervasive Computing and Communications (2004)*, 149–153.
14. JUELS, A. Minimalist cryptography for low-cost RFID tags. In *Proc. Int. Conf. Security in Communication Networks (SCN 2004) (2004)*, LNCS, vol. 3352, Springer, pp. 149–164.
15. KELSEY, J., SCHNEIER, B., WAGNER, D., AND HALL, C. Cryptanalytic attacks on pseudorandom number generators. In *FSE (1998)*, S. Vaudenay, Ed., LNCS, vol. 1372, Springer, pp. 168–188.
16. KIM, C. H., AVOINE, G., KOEUNE, F., STANDAERT, F.-X., AND PEREIRA, O. The Swiss-Knife RFID Distance Bounding Protocol. In *ICISC (2008)*, P. J. Lee and J. H. Cheon, Eds., vol. 5461 of *Lecture Notes in Computer Science*, Springer, pp. 98–115.
17. MANGARD, S., POPP, T., AND OSWALD, M. E. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*, Springer - ISBN: 0-387-30857-1. 2007.
18. MOLNAR, D., SOPPERA, A., AND WAGNER, D. A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags. In *Proc. Workshop on Selected Areas in Cryptography (SAC 2005) (2006)*, LNCS vol. 3897, Springer.

19. MUNILLA, J., PEINADO, A. Distance bounding protocols with void-challenges for RFID. In *In Workshop on RFID Security - RFIDSec '06* (2006).
20. OHKUBO, M., SUZUKI, K., AND KINOSHITA, S. Cryptographic approach to “privacy-friendly” tags. In *Proc. RFID Privacy Workshop* (2003).
21. QINGLING, C., YIJU, Z., AND YONGHUA, W. A minimalist mutual authentication protocol for RFID systems and BAN logic analysis. *Computing, Communication, Control and Management, ISECS International Colloquium* (2008), 449–453.
22. SEO, D., BAEK, J., AND CHO, D. Secure RFID Authentication Scheme for EPC Class Gen2. In *Proc. 3rd Int. Conf. on Ubiquitous Information Management and Communication (ICUIMC-2009)*, pp. 221–227.
23. SHARMA, S. E., WEISS, S. A., AND ENGELS, D. W. RFID systems and security and privacy implications. *Proc. Workshop Cryptographic Hardware and Embedded Systems (CHES 20002)* (2003), LNCS, vol. 2523, Springer, pp. 454–469.
24. SUN, H.-M., AND TING, W.-C. A gen2-based rfid authentication protocol for security and privacy. *IEEE Transactions on Mobile Computing* 99, 1 (2009).
25. VAN LE, T., BURMESTER, M., AND DE MEDEIROS, B. Universally Composable and Forward-Secure RFID Authentication and Authenticated Key Exchange. *Proc. ACM Symp. on Information, Computer, and Communications Security (ASIACCS 2007)*, ACM Press, pp. 242–252.