Generalized "Yoking-Proofs" for a Group of RFID Tags[*]

Leonid Bolotnyy and Gabriel Robins
Department of Computer Science
University of Virginia, Charlottesville, VA 22904
{lb9xk, robins}@cs.virginia.edu

Abstract

Recently Ari Juels suggested a "yoking-proof" where a pair of radio-frequency identification (RFID) tags are both read within a specified time bound, and left open for future research the problem of generating a proof for larger groups of tags. We generalize his protocol by developing a proof which ensures that a group of tags is read within a certain time period. The tags generate such a proof even if the reader is untrusted. The proof is improbable to forge, and is verifiable off-line by a trusted verifier. Juels's problem formulation does not take privacy into account and the resulting protocol offers no privacy to the tags. We modify the problem statement to require the "yoking-proof" to maintain privacy, and we give a protocol for this new anonymous yoking *problem, along with suggestions for speed ups.*

Keywords: RFID, Security, Yoking-Proofs.

1 Introduction

Some radio-frequency identification (RFID) scenarios require a proof of action (e.g., that a group of objects tagged with RFID tags were identified simultaneously). For example, pharmaceutical distributors may want to prove that a bottle of medicine was sold together with its instructions leaflet [5]; manufacturers may want to prove that safety devices were sold together with a tool or that a number of matching parts were delivered simultaneously; banking centers or security stations may want to prove that several forms of ID were read simultaneously; meeting organizers may want to prove that a group of people were present together at a meeting, etc. Third-parties can verify the validity of the proofs. For the examples given above, the verifying third parties can be regulatory agencies, company headquarters, etc.

We seek to ensure that if a group of tags is not read simultaneously, an entity will not be able to forge reality by constructing such a proof. Inspired by this problem, Ari Juels developed a protocol that creates such a proof for a *pair* of RFID tags [5]. He left open for future research the problem of generalizing his protocol to three or more tags. In this paper, we generalize his protocol to arbitrarily large groups of tags, give an algorithm for *anonymous yoking*, and show how these yoking protocols can be sped up.

2 Assumptions

We assume that RFID tags are passive and have limited computational capabilities. We require that they can execute keyed hash functions and maintain some state information such as a key, a counter, and some data computed during the protocol. These requirements can be satisfied in practice by Class-2 Generation-2 EPC tags [4]. Since tags are passive, they cannot communicate directly with each other, but they can communicate with each other indirectly through the reader. For now, we also assume that the adversary cannot physically steal tags' secret information. Later we discuss how this requirement can be relaxed.

Our verifier is assumed to be a trusted and computationally powerful machine. The verifier is considered to be off-line in the sense that it does not have to verify the proof immediately after it is created, and does not need to communicate with the tags. A reader is assumed to be adversarial, and we want the protocol to be secure against a reader that attempts to create the proof without reading all the tags simultaneously.

Replay attacks may or may not be considered a threat, depending on the application. In the protocol, we consider replays by an adversarial reader of previously seen proofs to be a viable threat, and thus we design the protocol accordingly. To avoid replay attacks, the verifier stores some information about previous correct proofs. The verifier is not required to store this information, if replays of valid proofs are not considered to be attacks. This will be elaborated upon in the discussion following the protocol specification.

We require that the tag accessed first by the reader be able to implement a timeout after a specific time period t has elapsed. Timeouts can be implemented on clock-less RFID tags. FCC regulations require the termination of tag-reading

[*]This research was supported by a Packard Foundation Fellowship and by NSF grant CCF-0429737. For a full version of this paper see http://www.cs.virginia.edu/robins.

protocols within 400ms. However, if the reader is malicious and violates these regulations, a capacitor discharge rate on-board a tag can be used for timing [5].

3 The Group Yoking Protocol

The idea of a 'yoking-proof'[1] for a group of tags is to construct a circular chain of mutually dependent message authentication code (MAC) computations. The purpose of the construction is to ensure that if an untrusted reader 'breaks the chain' (i.e. does not read all the tags within t time units), it will not be able to mount a replay attack nor create a proof that will be accepted by the verifier.

Assume that the system contains n tags, which are denoted by $T_1, T_2, \ldots, T_n$. Each tag T_i is assigned a unique key x_i that consists of d bits and a counter c_i. A tag has the ability to compute a keyed hash function and a standard message authentication code, which can be implemented as a keyed hash function, such as HMAC, in order to simplify the circuit. A reader will read k tags and produce a proof P that the tags were read near-simultaneously, i.e. within t time units. The verifier V knows all key assignments to tags and will verify that the proof P was indeed not forged.

Let $f : \{0,1\}^d \times \{0,1\}^* \rightarrow \{0,1\}^d$ be a keyed hash function, and let $MAC : \{0,1\}^d \times \{0,1\}^* \rightarrow \{0,1\}^d$ denote a standard *message authentication code*. Let $f_x[m]$ and $MAC_x[m]$ denote the computation of f and MAC with a secret x on input m, respectively.

In our protocol, we assume that the reader generates a proof for $T_1, T_2, \ldots, T_k$ ($k \leq n$) and queries them in that order. (Tags are queried based on their IDs or the random numbers that they generate.) The first tag will compute $r_1 = f_{x_1}[c_1]$ and send $a_1 = (1, c_1, r_1)$ to the reader. The reader will then send a_1 to the second tag. The second tag will compute $r_2 = MAC_{x_2}[c_2, a_1]$ and send $a_2 = (2, c_2, r_2)$ to the reader. The reader will then send a_2 to the third tag, which will perform the same computation as the second tag, i.e. $r_3 = MAC_{x_3}[c_3, a_2]$ and send $a_3 = (3, c_3, r_3)$ to the reader. The reader will then send a_3 to the fourth tag and so on, until the last tag k has computed r_k and sent $a_k = (k, c_k, r_k)$ to the reader.

The reader then sends a_k to the first tag, which computes $m = MAC_{x_1}[a_1, a_k]$ (assuming that t time units have not yet passed since the initial tag access), and sends m to the reader. The reader creates a proof $P_{1,2,\ldots,k} = (1, 2, \ldots, k, c_1, c_2, \ldots, c_k, m)$. To verify the proof $P_{1,2,\ldots,k}$, the verifier performs the same computations as the tags $1, 2, \ldots, k$, maintaining the order, and compares its generated proof P to the reader-provided proof $P_{1,2,\ldots,k}$. If the proofs match, the verifier outputs $success$; otherwise, it outputs $failure$. The algorithms for tag initialization, the reader, a tag, and the verifier are shown below.

[1]The term "yoking" suggests the *joining together*, or the simultaneous presence of all the tags.

Algorithm 1: Tag Initialization

```
for $i = 1$ to $n$ do
    $c_i = 0$
    $x_i \leftarrow$ choose randomly from $\{0,1\}^d$
end
```

Algorithm 2: Reader

```
$send(first, 0)$ to $T_1$
$receive(a_1 = (T_1, c_1, r_1))$ from $T_1$
for $i = 2$ to $k$ do
    $send(not\ first, a_{i-1})$ to $T_i$
    $receive(a_i = (T_i, c_i, r_i))$ from $T_i$
end
$send(final, a_k)$ to $T_1$
$receive(m)$ from $T_1$
construct $P_{1,2,\ldots,k} = (1, 2, \ldots, k, c_1, c_2, \ldots, c_k, m)$
```

Notice that each tag computes a MAC of a message that is a function of a MAC computed by the preceding tag in a chain. This ensures that the reader has at most t time units to create the proof. To avoid replay attacks and allow temporal ordering of the proofs, each tag increments its counter immediately after it sends a_i to the reader.

If the first tag does not update its counter right after it sends its first message, a possibly malicious reader can create a proof P that will successfully pass through the verifier, without reading all the tags within the specified time bound t. In such a scenario, a proof can be forged as follows. The malicious reader can ask the first tag to compute a_1, then wait for t time units to elapse in order to cause T_1 to timeout, then send a_1 to T_2 to obtain a_2. Then, the reader will access T_1 for a dummy computation of a_1, and send a_2 to T_1 to obtain m, and construct a valid proof $P = (1, 2, c_1, c_2, m)$. Juels' basic yoking protocol [5] suffers from this problem unless the counter on the first tag is incremented on a timeout, but this is not specified in his paper.

The security of our scheme hinges on the (exponentially-small) probability δ that an adversary A is able to construct a 'yoking-proof' $P_{1,\ldots,k}$, which could fool the verifier V into reporting "success", without actually reading the IDs of all the tags involved in the protocol within t time units, as intended.

Theorem: Given random-oracle assumptions for f and MAC [1], the success probability δ of an adversary A for a grouping protocol is bounded above by 2^{-d} where d is the message length. $\square$

The statement of the security property of our group protocol is similar to the one presented in [5]. A detailed proof can be found in the full version of our paper [3].

Algorithm 3: Tag

```
Input: Tag's order in the chain: $mode$ and output
       value from the previous tag in the chain: $value$
switch mode do
    case 'first'
        start timer
        $r = f_x[c]$
        $a = (ID, c, r)$
        $store(a)$
        $send(a)$
        $c = c + 1$
        break
    end
    case 'not first'
        $r = MAC_x[c, value]$
        $c = c + 1$
        $send(a = (ID, c, r))$
        break
    end
    case 'final'
        if timer has not timed out then
            $m = MAC_x[a, value]$
            $send(m)$
            timer times out
        else
            abort
        end
        break
    end
end
```

Algorithm 4: Verifier

```
Input: Proof $P = (1, 2, \ldots, k, c_1, c_2, \ldots, c_k, m)$
$r_1 = f_{x_1}[c_1]$
$a_1 = (1, c_1, r_1)$
for $i = 2$ to $k$ do
    $r_i = MAC_{x_i}[c_i, a_{i-1}]$
    $a_i = (i, c_i, r_i)$
end
$m^* = MAC_{x_1}[a_1, a_k]$
if $m == m^*$ then
    return $success$
else
    return $failure$
end
```

4 Discussion

Our group-yoking protocol can be adapted to the 'Minimalist MAC' (one-time MAC) protocol proposed by Juels. However, a one-time proof is not very useful given the possibility that the reader may be malicious.

To prevent an adversary from replaying old proofs, the verifier can store counter values of tags from the latest correct proofs in which the tags participated. A replay attack will use the counter value that is less than or equal to the last recorded counter for the tag. Storing counter values also allows for a temporal ordering of the proofs, which may be desired. If there is more than one verifier in the system, the verifiers need to be able to share the tags' counter values.

Having counters on-board tags allows for temporal ordering of the proofs and guarantees that a keyed hash function will never be recomputed on the same input. However, such counters require tags to maintain a persistent state between several runs of the protocol. Counters can be substituted with random numbers generated on-board the tags (the tags will need to be equipped with pseudo-random number generators and maintain secret seeds). These random numbers should have enough bits (e.g., 64+ bits) to prevent "birthday attacks". However, if random numbers are used instead of counters, and a replay of previous proofs is a possible attack, all past proofs need to be stored and compared against the new proof.

We assumed above that the reader cannot physically steal secrets from the tags. We can relax this assumption and allow a malicious reader to physically steal secrets from one or more tags *after* the timer of the yoking protocol times out. In this scenario, possessing the key of the "first" tag will enable a malicious reader to complete the broken protocol on its own and thereby construct a forged proof (stealing any other key will not enable a malicious reader to complete the proof and is therefore not a threat).

To prevent such an attack, each tag can update its secret key in a forward-secure manner. For example, a one-way hash function can be used to update the key. The old key is then securely discarded. For this scheme to be practical, tags should maintain counters, rather than use random numbers as seeds, to allow a verifier to quickly determine the secret used by each tag during the computation.

5 Anonymous Yoking

Juels's "yoking-proof" protocol and our generalization do not hide the identities of tags - each tag sends its identifier and its counter value to the reader. Before the execution of the protocol, the reader is unaware of the identities of tags, and running the yoking protocol will reveal them to it. In practical scenarios, we would like to preserve the privacy of objects associated with the tags (i.e. not reveal tag IDs to untrusted readers). We therefore introduce a new problem formulation, called *anonymous yoking*, which in addition to the requirements of a "yoking-proof" scenario, also requires tags to preserve their privacy.

The protocol we propose for anonymous yoking is similar to the generalized "yoking-proof" protocol discussed above. Let $f : \{0,1\}^d \times \{0,1\}^* \rightarrow \{0,1\}^d$ be a keyed hash function. Upon the reader's request, each tag will generate a random number r, and compute $a = f_x(r, value)$, where x is a secret key stored on a tag and $value$ is an output of the previous tag in the chain, as in our "yoking-proof" scenario above. The first tag sets $value$ equal to 0. Each tag will respond to the reader's request by sending (r, a) to it, and the first tag will close the chain. The proof of security is very similar to the one for the generalized protocol. The verifier will try to determine which secrets were used to compute each a_i given r_i and a_{i-1}. Since a_i is a function of a_{i-1} for all $2 \leq i \leq k$, the process of determining the secrets and verifying the proof coincide.

6 Speeding Up the Yoking Protocols

In our generalized "yoking-proof" protocol there is one tag that starts and closes the chain. This implies that the time it takes to create the proof is the sum of the reader communication times with each tag. This is acceptable for as long as the number of tags participating in the protocol is small. However, if the number of tags is large, speeding up the protocol may not be a simple optimization, but a requirement to ensure that all tags remain within the reader(s) reading range, and that the proof can be created within the required time limits.

The yoking-proof creation can be sped up by splitting the circular chain of dependent MACs into a group of arcs, where each arc consists of a sequence of dependent MACs, and where the adjacent arcs are inter-dependent. Each arc has a single element that plays the role of the "first" and the "last" tag. Let $ID_1, \ldots, ID_k$ be the tags' identifiers sorted by the tags ID, or by the random numbers generated onboard the tags for anonymous yoking. We split the sorted list of identifiers into the desired, g, number of groups (e.g., $ID_1, \ldots, ID_{i_1}, ID_{i_1+1}, \ldots, ID_{i_2}, \ldots, ID_{i_g}, \ldots, ID_k$).

For example, ID_1 is the "first" and the "last" element of the first arc. It starts the chain of its group ($ID_1, \ldots, ID_{i_1}$) as described in the generalized "yoking-proof" protocol, and it closes the chain of the group $ID_{i_g}, \ldots, ID_k$. In other words, the first element of each arc starts the chain of the arc and closes the chain of the preceding arc.

Note that the protocol requires multiple readers, or a single reader with multiple antennas, as well as a medium access control scheme that allows the reader(s) to communicate with more than one tag at a time (so that collisions are minimized/avoided). The time to create the yoking proof is the sum of the reader communication times with the tags belonging to the longest arc. Therefore, the overall speedup factor resulting from partitioning the tag set into arcs, can ideally approach the number of arcs.

7 Related Work

The authors of [6] suggest a group protocol which relies on time stamps provided by the back-end database. In their scheme a reader receives a time stamp TS from the database, which it sends to all the tags. Each tag T_i computes $m_i = MAC_{x_i}[TS]$ and sends m_i back to the reader. The authors assume the existence of one powerful/leader tag among those read. The reader sends all m_i to this leader tag, which encrypts them along with the time stamp TS using encryption function SK keyed with a secret key x. The encryption result C_p is sent to the reader and the proof is $P_n = (TS, C_p)$.

First, the assumption in [6] that one of the tags is more powerful than the others is not true in many practical scenarios. The second and main weakness of their protocol is that an untrusted reader can pick the time stamp TS as a future time, and then use it on one tag and later on another, thus violating the near-simultaneous read requirement. Even if the time stamp TS is encrypted, the reader can still separate tag accesses arbitrarily far apart in time, since each access is independent of the others.

8 Conclusion

We designed a protocol that creates a proof that an arbitrarily large group of RFID tags are read within a specified time bound. This "yoking-proof" is improbable to forge and is verifiable off-line by a trusted verifier. We then modified the security requirements for the problem, requiring the system to maintain privacy, and described a protocol for anonymous yoking, where the tag identities are hidden. We showed how group yoking protocols can be sped up. Future research opportunities include the development of other "yoking-proofs" for RFID, which utilize the ability of tags to communicate with each other through the reader [2].

References

[1] M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, First ACM Conference on Computer and Communications Security, 1993, pp. 62-73.

[2] L. Bolotnyy and G. Robins, *Inter-Tag Communication and Tag Cooperation in RFID Systems*, Technical Report CS-2006-11, Department of Computer Science, University of Virginia, 2006.

[3] L. Bolotnyy and G. Robins, *Generalized "Yoking-Proofs" for a Group of Radio Frequency Identification Tags*, Technical Report CS-2006-12, Department of Computer Science, University of Virginia, 2006.

[4] EPCglobal, Specification for RFID Air Interface, *EPC Radio-Frequency Identity Protocols Class-1 Generation-2 UHF RFID - Protocol for Communications at 860MHz-960MHz, Version 1.0.9*.

[5] A. Juels, *'Yoking-Proofs' for RFID Tags*, IEEE Workshop on Pervasive Computing and Communication Security (PerSec), 2004, pp. 138-143.

[6] J. Saito and K. Sakurai, *Grouping Proof for RFID Tags*, 19th International Conference on Advanced Information Networking and Applications (AINA), 2005, Volume 2, pp. 621-624.