A Family of Dunces: Trivial RFID Identification and Authentication Protocols

Gene Tsudik*

Computer Science Department
University of California, Irvine
gts@ics.uci.edu

Abstract. Security and privacy in RFID systems is an important and active research area. A number of challenges arise due to the extremely limited computational, storage and communication abilities of a typical RFID tag. This paper describes a step-by-step construction of a family of simple protocols for inexpensive untraceable identification and authentication of RFID tags. This work is aimed primarily at RFID tags that are capable of performing a small number of inexpensive conventional (as opposed to public key) cryptographic operations. It also represents the first result geared for so-called *batch mode* of RFID scanning whereby the identification (and/or authentication) of tags is delayed. Proposed protocols involve minimal interaction between a tag and a reader and place very low computational burden on the tag. Notably, they also impose low computational load on back-end servers.

1 Introduction

RFID technology is rapidly becoming ubiquitous. In the near future, it is expected to replace barcodes as the most common means of product and merchandise identification. Current and emerging applications range from toll transponders, passports and livestock/pet tracking devices (on the high end) to miniscule stealthy tags in everyday items, such as clothing, pharmaceuticals, library books and so on. Unlike barcodes, RFID tags do not require close physical proximity between a reader and a scanned object and also do not require a line-of-sight communication channel. Furthermore, RFID tags' smaller form factor takes up less valuable packaging "real estate". However, current and emerging RFID proliferation into many spheres of everyday life raises numerous privacy and security concerns.

One of the main issues has to do with malicious tracking of RFID-equipped objects. While tracking RFID tags is typically one of the key features and goals of a legitimate RFID system (e.g., in a supply-chain environment) *unauthorized* tracking of RFID tags is viewed as a major privacy threat.

In general, in-roads recently made by the RFID technology have prompted some public discontent and controversy. Privacy advocates have pointed out some sinister consequences of malicious tag tracking.

* An earlier (and much shorter) version of this paper appeared in [1]. This paper includes **substantial** revisions, enhancements and extensions to [1].

N. Borisov and P. Golle (Eds.): PET 2007, LNCS 4776, pp. 45–61, 2007.
© Springer-Verlag Berlin Heidelberg 2007

This paper describes a protocol family for inexpensive untraceable identification and authentication of RFID tags. *Untraceable* means that it is computationally infeasible to infer – from interactions with a tag – information about the identity of the tag or link multiple authentication sessions of the same tag. Proposed protocols are inexpensive, requiring as little as one light-weight cryptographic operation on the tag and storage for one key. They are particularly well-suited for the **batch mode** of tag identification whereby a reader interrogates a multitude of tags and later identifies/authenticates them in bulk. Furthermore, real-time computational load on the back-end sever is minimal due to the simple pre-computation technique described below.

1.1 Operating Environment

The *adversary*, in our context, can be either passive (e.g., eavesdropper) or active (e.g., impersonator). It can corrupt or, attempt to impersonate, any entity and we assume that its primary goal is to track RFID tags. (In other words, we say – informally – that the adversary succeeds if it manages, with non-negligible probability over 50%, to link multiple authentication sessions of the same tag.) We point out from the start that we **do not** initially consider *forward security* – adversary's inability to link or trace prior manifestations of a tag in the event of complete tag compromise. At the same time, compromise of a set of tags should not lead to the adversary's ability to track other tags (except by distinguishing among the two sets). Furthermore, our initial goals **do not** include resistance to denial of service (DoS) attacks, e.g., attacks that aim to disable the tags. However, we outline out some DoS-resistant solutions later in the paper.

The legitimate entities are: tags, readers and servers. A reader is a device querying tags for identification information. A server is a trusted entity that knows and maintains all information about tags, their assigned keys and any other such information. A server is assumed to be physically secure and not subject to attacks. Multiple readers are assigned to a single server. A server only engages in communication with its constituent readers. For simplicity, we assume a single logical server that might resolve to multiple physically replicated servers. All communication between a server and its readers is assumed to be over private and authentic channels. Furthermore, servers and readers maintain **loosely synchronized clocks**. Both readers and server have ample storage, computational and communication abilities. (However, in some cases, readers may not be able to communicate with servers in real time; see below.)

We assume that an RFID tag has no clock and small amounts of ROM (e.g., to store a key) and non-volatile RAM (to store ephemeral state, such as a counter or time-stamp). With power supplied by a reader – whether contact or contactless – a tag is able to perform a modest amount of computation and commit any necessary state information – of small constant length – to non-volatile storage.

1.2 Goals

As usual, our goals are to *minimize everything*, including:

1. non-volatile RAM on the tag
2. code (gate count) complexity
3. tag computation requirements
4. number of rounds in reader-tag interaction[1]
5. message size in reader-tag interaction
6. server real-time computation load
7. server storage requirements

It is easy to see that the first three items directly influence tag cost. Also, the 4th item (number of rounds and messages) is important since more rounds imply more protocol logic and, hence, higher complexity and gate count. In fact, having more than two rounds in reader-tag interaction implies that the tag MUST keep soft state while the protocol executes. This would necessitate either bigger non-volatile RAM or continuous power from the reader (while the protocol executes) to store soft state in volatile RAM.

Finally, we need to avoid features currently not realistic for most low-cost RFID tags, such as public key cryptography, tamper-resistant shielding or an on-board clock.

1.3 Modes of Operation

We consider two modes of tag identification: *real-time* and *batch*. Here we make an assumption that the back-end server is necessary as a reader is unable to identify/authenticate tags on its own, primarily because of the scale, i.e., large numbers of deployed tags. In situations where this assumption is false, the discussion in this section does not apply. For example, one could imagine an RFID-equipped driver's license reader carried by law enforcement officers which is capable of storing information about all locally-issued driver's licenses, e.g., on the order of tens of millions. (Also, some recent work [16] shows how to perform, under some circumstances, *serverless* RFID authentication.)

The *real-time* mode is the one typically considered in the literature: it involves on-line contact between the reader and the server, in order to quickly identify (and, optionally, authenticate) the tag in question. If immediate feedback about a tag is needed – e.g., in facility access, retail or library check-out scenarios – the server must be contacted in real time.

In *batch* mode, a reader scans numerous tags, collects replies and sometime later performs their identification (and optionally, authentication) in bulk. From the security perspective, the batch mode seems relevant wherever immediate detection of fraudulent/counterfeit tags is not the the highest-priority issue and,

[1] We use the term "interaction" – as opposed to "protocol" – since the actual tag authentication protocol may involve interaction between a reader and a server, in addition to that between a tag and a reader. We are understandably less concerned about the complexity of the former.

instead, emphasis is on security against fraudulent readers. In practical terms, however, the batch mode is appropriate when circumstances prevent or inhibit contacting the back-end server in real time. For example, consider an inventory control application, where readers are deployed in a remote warehouse and have no means of contacting a back-end server in real time. More generally, some of the following factors might prompt the use of the batch mode:

- The server is not available in real time, either because it is down, disconnected or because readers do not have sufficient means of communication.
- The server is available, but is over-loaded with requests, causing response time to be jittery, thus making each tag interrogation instance unacceptably slow.
- The server is available and not over-loaded but is located too far away, causing response time to be too long. (Or, the network is congested, which cause unacceptable delays).
- A mobile/wireless reader has limited resources and, in order to conserve battery power, simply can not afford to contact the server for each scanned tag.

1.4 Tag Requirements

Each tag $RFID_i$ is initialized with at least the following values:

$$K_i,\ T_0,\ T_{max}$$

K_i is a tag-specific value that serves two purposes: (1) tag identifier, and (2) cryptographic key. Thus, its size (in bits) must be the greater of that required to uniquely identify a tag (i.e., a function of the total number of tags) and that required to serve as sufficiently strong cryptographic key for the purposes of Message Authentication Code (MAC) computation. In practice, a 160-bit K_i will most probably suffice.

T_0 is the initial timestamp assigned to the tag. This value does not have to be a discrete counter, *per se*. For example, T_0 can be the time-stamp of manufacture. T_0 need not be tag-unique; an entire batch of tags can be initialized with the same value. The bit-size of T_0 depends on the desired granularity of time and the number of times a tag can be authenticated. T_{max} can be viewed as the highest possible time-stamp. Like T_0, T_{max} does not need to be unique, e.g., a batch of tags can share this value.

Each tag is further equipped with a sufficiently strong, uniquely seeded pseudo-random number generator (PRNG). In practice, it can be resolved as an iterated keyed hash (e.g., HMAC) started with a random secret seed and keyed on K_i. For a tag $RFID_i$, $PRNG_i^j$ denotes the j-th invocation of the (unique) PRNG of that tag. No synchronization whatsoever is assumed as far as PRNG-s on the tags and either readers or servers. In other words, given a value $PRNG_i^j$, no entity (including a server) can recover K_i or any other information identifying $RFID_i$. Similarly, given two values $PRNG_i^j$ and $PRNG_j^k$, deciding whether $i = j$ must be computationally infeasible.

2 A Family of Dunces: YA-TRIP, YA-TRAP and YA-TRAP*

In this section, we introduce our main idea, based on the use of monotonically increasing time-stamps. We then present three protocols, starting with YA-TRIP which only offers efficient tag identification, continuing with YA-TRAP which also provides tag authentication, and concluding with YA-TRAP* which, in addition, incorporates DoS resistance features.

2.1 The Main Idea

The main idea of our proposal is the use of monotonically increasing time-stamps[2] to provide tracking-resistant (anonymous) tag authentication. The use of timestamps is motivated by the old result of Herzberg, et al. [6], which we briefly summarize next.

The work in [6] considered anonymous authentication of mobile users who move between domains, e.g., in a GSM [13] cellular network or a wired Kerberos-secured [12] internetwork. The technique in [6] involves a remote user identifying itself to the host domain by means of an ephemeral userid. An ephemeral userid is computed as a (collision-resistant, one-way) hash of current time and a secret permanent userid.

A trusted server in the user's "home" domain maintains a periodically updated hash table where each row corresponds to a traveling user. The length of the update interval is a system-wide parameter, e.g., one hour. The table can be either pre-computed or computed on-the-fly, as needed. Each row contains a permanent userid and a corresponding ephemeral userid. When a request from a *foreign* agent (e.g., Kerberos AS/TGS[3] in a remote domain or VLR[4] in a GSM setting) comes in, the home domain server looks up the ephemeral userid in the current table. (Since hash tables are used, the lookup cost is constant.) Assuming that timestamp used by the (authentic) traveling user to compute the ephemeral userid is reasonably recent (accurate), the hash table lookup is guaranteed to succeed. This allows a traveling user to be authenticated while avoiding any tracing by foreign agents or domains.

One of the main advantages of this approach is that the home domain server does not need to compute anything on demand, as part of each request processing. Instead, it pre-computes the current hash table and waits for requests to come in. The cost of processing a request amounts to a table lookup (constant cost) which is significantly cheaper than a similar approach using nonces or random challenges. In the latter case, the server would need to compute an entire table on-the-fly in order to identify the traveling user. As time goes by, an ephemeral userid table naturally 'expires' and gets replaced with a new one. This is the main feature we would like to *borrow* for tag authentication purposes.

[2] No other type of counters or sequence numbers will do.
[3] Authentication Server / Ticket Granting Server.
[4] Visitor Location Registry.

Although the technique from [6] works well for traveling/mobile users, it is not directly applicable to the envisaged RFID environment. First, a mobile user can be equipped with a trusted personal device that keeps accurate time. It can be as simple as a wristwatch or as sophisticated as a PDA. (Moreover, even without any trusted device, a human user can always recognize grossly incorrect time, e.g., that which is too far into the future.) Such a device can be relied upon to produce reasonably accurate current time. An RFID tag, on the other hand, cannot be expected to have a clock. Thus, it is fundamentally unable to distinguish among a legitimate and a grossly inaccurate (future) time-stamp.

However, if the tag keeps state of the last time-stamp it "saw" (assuming it was legitimate), then it can distinguish between future (valid) and past (invalid) time-stamps. We capitalize on this observation and rely on readers to offer a putatively valid timestamp to the tag at the start of the identification protocol. A tag compares the time-stamp to the stored time-stamp value. If the former is strictly greater than the latter, the tag concludes that the new time-stamp is probably valid and computes a response derived from its permanent key and the new timestamp. A tag thus never accepts a time-stamp earlier than – or equal to the one stored. However, to protect against *narrowing* attacks[5], even if the timestamp supplied by the reader pre-dates the one stored, the tag needs to reply with a value indistinguishable from a normal reply (i.e., a keyed hash over a valid timestamp). In such cases, the tag replies with a random value which is meaningless and cannot be traced to the tag even by the actual server.

2.2 YA-TRIP: Yet Another Trivial RFID Identification Protocol

We now present the first protocol (YA-TRIP) which provides only the very basic service – efficient tag identification. The protocol is illustrated in Figure 1.

Remark: We note that MACs based on keyed hashes are a fairly simple and general technique which has been used in other RFID-related contexts, e.g., [19]. Although YA-TRIP and its extensions described below use such standard MACs, it is worth noting that more efficient constructs have been proposed, e.g., Juels' light-weight MAC scheme in [19]. Also, the use of PRNGs to obfuscate the tag identity was first introduced by Weis, et al. [18].

The important part of the protocol encompasses steps 1-3. It consists of only two rounds and two messages. The size of the first message determined by T_r and the second – by H_{id}. In each case, the size is no greater than, say, 160 bits.

Note that H_{id} computed in step 2.3.2 and sent in step 3 does not actually authenticate the tag in the sense of the tag actually being present. What step 2.3.2 achieves is a weaker notion which we call "identification". It proves that *at some point*, perhaps far in the past, the tag was involved in a protocol (with a legitimate or a rogue reader) wherein it received, and replied to, the value T_r.

[5] Informally, a *narrowing attack* occurs when the adversary queries a tag with a particular timestamp and then later tries to identify the same tag by querying a candidate tag with a timestamp slightly above the previous one.

[1] Tag $\longleftarrow$ Reader: T_r
[2] Tag:[a]
- [2.1] $\delta = T_r - T_t$
- [2.2] if $(\delta \leq 0)$ or $(T_r > T_{max})$
- [2.2.1] $H_{id} = PRNG_i^j$
- [2.3] else
- [2.3.1] $T_t = T_r$
- [2.3.2] $H_{id} = HMAC_{K_i}(T_t)$

[3] Tag $\longrightarrow$ Reader: H_{id}
In real-time mode, the following steps take place immediately following Step 3. In batch mode, they are performed later.
[4] Reader $\longrightarrow$ Server: $T_r,\ H_{id}$
[5] Server:
- [5.1] $s = LOOKUP(HASH_TABLE_{T_r}, H_{id})$
- [5.2] if $(s == -1)$
- [5.2.1] MSG=TAG-ID-ERROR
- [5.3] else
- [5.3.1] MSG=TAG-VALID /* can return $G(K_s)$ instead */

[6] Server $\longrightarrow$ Reader: MSG

[a] Note that it is **imperative** for the respective times taken by steps 2.2 and 2.3 to be as close as possible. This is needed to prevent obvious timing attacks by malicious readers (aimed at distinguishing among the two cases). This is the reason for $PRNG$ to be resolved as described in Section *"Tag Requirements"* above.

Fig. 1. YA-TRIP: Tag Identification

In other words, the tag's reply in step 3 could be pre-recorded and replayed by the adversary.

Recall that we assume private and authentic channels between readers and the back-end server. Moreover, a server is assumed to "talk" only to non-compromised (non-malicious) readers. This pertains to steps 4 and 6 above. Note also that the specifics of step 5.3 depend on the application requirements. If the application allows genuine readers to identify/track valid tags, the server could return a meta-id of the tag: $G(K_s)$ where $G(.)$ is a suitable cryptographic hash with the usual features. Otherwise, it suffices to simply inform the reader that the tag in question is valid, as shown in Step 5.3.1 in Figure 1.

In batch mode, the reader interrogates a multitude of tags, collects their responses and, at a later time, off-loads the collected responses, along with the corresponding T_r value(s) to the server. (Note that, if tag responses are collected over multiple time intervals, the reader needs to group responses according to the T_r value used.) The server then needs to identify the tags. In this setting, YA-TRIP is highly advantageous. Even currently most efficient techniques

such as the MSW protocol [2], require the server to perform $O(\log\ n)$ pseudo-random function (PRF) operations to identify a single tag. This translates into $O(n * \log\ n)$ operations to identify n tags. Whereas, YA-TRIP only needs $O(n)$ operations for the same task (since the same T_r-specific hash table is used for all lookups and each lookup takes constant time).

2.3 Drawbacks

The YA-TRIP protocol, as presented above, has some potential drawbacks.

First, YA-TRIP does not provide tag authentication – it merely identifies a tag. In order to authenticate itself, a tag needs to reply to a random challenge by the reader. Obviously, T_r is not random, thus, the reply in step 3 only identifies the tag. To remedy this, we extend the protocol in section 2.4.

Second, YA-TRIP is susceptible to a trivial denial-of-service (DoS) attack: an adversary sends a wildly inaccurate (future) time-stamp to a tag and incapacitates it either fully (if the time-stamp is the maximal allowed) or temporarily. Although DoS resistance is not one of our initial goals, it is an important issue. We address it in section 2.5.

Third, the protocol does not offer reader authentication or identification. We do not consider this to be a drawback but a feature. Viewed from the application perspective, the main purpose of reader/tag interaction is to identify (and, optionally, authenticate) the tag. While a number of previously proposed protocols manage (or attempt) to let the tag authenticate the reader, we claim that this is ultimately a waste of time and resources. The reason for this claim is two-fold:

1. **MORE ROUNDS:** Authenticating a reader requires at least a three rounds and three protocol messages; whereas, YA-TRIP is a two-round two-message protocol. It is easy to see why a minimum of three rounds would be needed: the reader always initiates interaction (round 1), the tag generates a challenge and sends it to the reader (round 2), and, the reader replies to the challenge (round 3). Moreover, if the tag does not identify (and authenticate) itself to the reader until the reader first authenticates itself to the tag, a fourth round (and a fourth) message becomes necessary.
2. **TAG STATE:** To authenticate a reader, the tag MUST "remember" the challenge it sends to the reader. This challenge represents state that must be kept by the tag between rounds 2 and 3. However, this brings up the possibility of the reader's reply never arriving, i.e., what happens if the protocol does not complete? The tag winds up in a state of "tilt" and requires additional logic to recover. All this translates into needing more resources on the tag.

Finally, the protocol makes an important assumption that a given tag is never authenticated (interrogated) more than once within the same interval. This has some bearing on the choice of the interval. A relatively short interval (e.g., a second) makes the assumption realistic for many settings. However, it translates into heavy computational burden for the server, i.e., frequent computation of ephemeral tables. On the other hand, a longer interval (e.g., an hour) results

in much lower server burden, albeit, it may over-stretch our assumption, since a tag may need to be interrogated more than once per interval. One solution is to sacrifice some degree of untraceability in favor of increased functionality, i.e., allow a tag to iterate over the same time value (accept $T_r = T_t$) a fixed number of times, say k. This would entail storing an additional counter on the tag; once the counter for the same T_t reaches k, the tag refuses to accept $T_r = T_t$ and starts responding with random values as in Step 2.2 in the protocol. The resulting protocol would be k**-traceable** – an adversary would be able to track a tag over at most k sessions, with the same T_r value. (Note that the adversary can track actively, by interrogating the tag, or passively, by eavesdropping on interactions between the tag and valid readers.)

2.4 YA-TRAP: Adding Tag Authentication

Adding tag authentication to YA-TRIP is easy, requiring a few minor protocol changes. First, we amend the initial reader→tag message to include a random challenge R_r. Then, we include a MAC of both (reader and tag) challenges in the tag's reply message. Later, once the tag is identified by the server, it can be authenticated by verifying the MAC. The identification step is the same as in YA-TRIP. The resulting protocol (YA-TRAP) is shown in Figure 2.

Once the server identifies the tag (via $LOOKUP$), the extra cost of authenticating it is negligible amounting to one HMAC operation. The additional cost for the tag in YA-TRAP consists of one PRNG invocation and one HMAC to compute H_{auth}.

Introducing H_{auth} into the protocol serves another useful purpose. In the event that the tag has been previously de-synchronized (incapacitated) by a rogue reader and its T_t value has been set far into the future, H_{auth} alone can be used as a fall-back in order to identify and authenticate the tag. However, this would require the server to perform $O(n)$ operations – for each tag $0 \leq j < n$, compute $HMAC_{K_j}(R_t, R_r)$ and compare with H_{auth}. This side-benefit of H_{auth} is useful in mitigating DoS attacks. On the other hand, it puts a much heavier load on the server which is arguably unimportant in the *batch mode*. Whereas, if used in *real time* mode, an adversary who is observing (and timing) tag-reader interaction might be able to discern a tag that has been previously desynchronized. Consider the environment where a successful reader-tag interaction results in some observable event, e.g., a door or a turnstile opens. Now, the adversary can measure the delay between the tag→ reader message (step 3 in Figure 2) and the observable event (which takes place after step 6). In the context of a previously desynchronized tag, this delay would be appreciably longer than that with a normal (synchronized) tag. Short of artificially padding the delay for all tags to be the same as for a desynchronized tag (which is clearly undesirable), there does not appear to be a workable solution.

We point out that the step from YA-TRIP to YA-TRAP is identical to that in the work of Juels [17] where the same idea was used in a somewhat different context.

[1] Tag ⟵ Reader: $T_r,\ R_r$
[2] Tag:
- [2.1] $\delta = T_r - T_t$
- [2.2] if $(\delta \leq 0)$ or $(T_r > T_{max})$
- [2.2.1] $H_{id} = PRNG_i^j$
- [2.3] else
- [2.3.1] $T_t = T_r$
- [2.3.2] $H_{id} = HMAC_{K_i}(T_t)$
- [2.4] $R_t = PRNG_i^{j+1}$
- [2.5] $H_{auth} = HMAC_{K_i}(R_t, R_r)$

[3] Tag ⟶ Reader: $H_{id},\ R_t,\ H_{auth}$
- THEN, LATER:

[4] Reader ⟶ Server: $T_r,\ H_{id},\ R_r,\ R_t,\ H_{auth}$
[5] Server:
- [5.1] $s = LOOKUP(HASH_TABLE_{T_r}, H_{id})$
- [5.2] if $(s == -1)$
- [5.2.1] MSG=TAG-ID-ERROR
- [5.3] else if $(HMAC_{K_s}(R_t, R_r) \neq H_{auth})$
- [5.3.1] MSG=TAG-AUTH-ERROR
- [5.4] else MSG=TAG-VALID

[6] Server ⟶ Reader: MSG

Fig. 2. YA-TRAP: Tag Authentication

2.5 YA-TRAP*: Adding DoS Resistance

Both YA-TRIP and YA-TRAP are susceptible to DoS attacks whereby a rogue reader can easily incapacitate a tag by feeding it a "futuristic" (or even maximum) T_r value. Although it is not one of our initial goals (our emphasis is on efficient identification and authentication), we recognize that DoS resistance is an important issue in practice. Therefore, we now show how to extend YA-TRAP to mitigate Denial-of-Service (DoS) attacks aimed at incapacitating tags.

DoS attacks on YA-TRIP/YA-TRAP are possible because a tag has no means to distinguish a realistic (more-or-less current) time-stamp T_r from one that is too futuristic. Since adding a clock to a tag is not an option, we need to rely on external means of establishing timeliness.

Our approach to timeliness requires a reader to present an epoch token each time it queries a tag, as part of the initial reader→tag message. The epoch token allows a tag to ascertain that the reader-supplied T_r is not too far into the future. This token changes over time, but its frequency of change (epoch) is generally much slower than the unit of T_r or T_t time-stamps. For example, T_t and T_r are measured in minutes, whereas, the epoch token might change daily. The

main idea is that a current epoch token can be used to derive past epoch tokens but cannot be used to derive future epoch tokens. A malicious or compromised reader might possess the current token but will not obtain future tokens. Also, since the epoch token changes slower than the time-stamp, multiple genuine interactions between one or more readers and the same tag might use the same epoch token but different (increasing) T_r values. We envisage the trusted server serving as the distribution point for epoch tokens. Upon (or near) each epoch, the server delivers (or securely broadcasts) the appropriate epoch token to all genuine readers.

The choice of the epoch duration directly influences the degree of vulnerability to DoS attacks. If the epoch is too long (e.g., a month), a rogue reader would be able to put tags out of commission for at most a month. (Note that the current epoch token is not secret; see below.) In contrast, if the epoch is very short (e.g., a minute), a tag might be out of service for at most a minute, however, the frequency of update becomes problematic since each reader would need to obtain the current epoch token from the trusted server or some other trusted repository.

The protocol (YA-TRAP*) is illustrated in Figure 3. DoS resistance in YA-TRAP* is obtained by introducing a much abused and over-used cryptographic primitive – a hash chain. A hash chain of length z is generated by starting with an initial value (say, X) and repeatedly hashing it z times to produce a root $H^z(X)$. The trusted (and, in batch mode, off-line) server is assumed to have initialized and generated the hash chain.

In addition to values mentioned in Section 1.4, each tag is initialized with a root of the hash chain $ET_0 = H^z(X)$ of length $z = T_{max}/INT$ where T_{max} is as defined in Section 1.4 and INT is the epoch duration, e.g., one day.

At any given time, a tag holds its last time-stamp of use T_t and the its last epoch token of use ET_t. (Note that "last" does not mean current or even recent; a tag may rest undisturbed for any period of time). When a reader queries a tag (in step 1), it includes ET_r, the current epoch token. The tag calculates the offset of ET_r as ν in step 2.2. Assuming a genuine reader, this offset represents the number of epochs between the last time the tag was successfully queried and ET_r. If T_r is deemed to be plausible in the first two OR clauses of step 2.3, the tag computes ν successive iterations of the hash function $H()$ over its prior epoch token ET_t and checks if the result matches ET_r. In case of a match, the tag concludes that T_t is not only plausible but is at most INT time units (e.g., one day) into the future. Otherwise, the tag assumes that it is being queried by a rogue reader and replies with two random values: $PRNG_i^j$ and $PRNG_i^{j+1}$, indistinguishable from $H_{id} = HMAC_{K_i}(T_t)$ and $HMAC_{K_i}(R_t, R_r)$, respectively.

We note that, even if $H^\nu(ET_t)$ matches ET_r, the tag cannot determine whether T_r and ET_r are **current**. The tag can only conclude that T_r is strictly greater than T_t and T_r corresponds to the same epoch as ET_r. We claim that this feature has no bearing on the security of YA-TRAP*. Since the tag has no clock, it cannot possibly distinguish between current and past-but-plausible values or T_r and ET_r. It can, however, establish with certainty that it has never

replied to the same T_r before and that the epoch corresponding to ET_r is at most current (i.e., not future).

Another detail is that the purpose of Step 2.3.2 in Figure 3 is to inhibit the adversary's (whether a passive eavesdropper or a malicious reader) ability to differentiate between valid and invalid reader input, from the tag's point of view. However, it is NOT the purpose of Step 2.3.2 to obscure the value of ν (see Section 2.6 below).

[1] Tag $\longleftarrow$ Reader: $T_r,\ R_r, ET_r$
[2] Tag:
- [2.1] $\delta = T_r - T_t$
- [2.2] $\nu = \lfloor T_r/INT \rfloor - \lfloor T_t/INT \rfloor$
- [2.3] if $(\delta \leq 0)$ or $(T_r > T_{max})$ or $(H^{\nu}(ET_t) \neq ET_r)$
- [2.3.1] $H_{id} = PRNG_i^j$;
- [2.3.2] $R_t = PRNG_i^{j+1}$
- [2.3.3] $H_{auth} = PRNG_i^{j+2}$
- [2.4] else
- [2.4.1] $T_t = T_r$
- [2.4.2] $ET_t = ET_r$
- [2.4.3] $H_{id} = HMAC_{K_i}(T_t)$
- [2.4.4] $R_t = PRNG_i^{j+1}$
- [2.4.5] $H_{auth} = HMAC_{K_i}(R_t, R_r)$

Steps [3-6] are the same as in YA-TRAP

Fig. 3. YA-TRAP*: DoS Resistance

Remaining DoS Attacks: DoS resistance in YA-TRAP* is limited by the magnitude of the system-wide INT parameter. Once revealed by the server and distributed to the genuine readers, the current epoch token ET_r is not secret; it can be easily snooped on by the adversary. Therefore, the adversary can still incapacitate tags (however many it has access to) for at most the duration of INT if it queries each victim tag with the current epoch token and the maximum possible T_r value within the current epoch. We consider this kind of a limited DoS attack to be a relatively small price to pay.

2.6 Discussion and Extensions

Forward Security: None of the aforementioned protocols is forward-secure [15,14]. Forward security would require periodic key evolvement. If the tag's key (K_i) evolves in a one-way fashion (e.g., via as suitable hash function), then, an adversary who compromises a tag at time T cannot identify/link prior occurrences of the same tag. We view forward security as a feature orthogonal to our main design

goals. However, it is relatively easy to add forward security to all three protocols; we sketch this out in the context of YA-TRAP* (refer to Figure 3):

- Introduce an additional operation for the tag:
  - [2.4.6] $K_i^{\nu} = H^{\nu}(K_i)$
- Change the way the server computes ephemeral tables:
  - Recall that, for each time-stamp T_c, the server pre-computes a table, where each entry corresponds to a unique tag i. Instead of computing each table entry as: $H_{K_i}(T_c)$, the server computes it as: $H_{K_i^{\nu}}(T_c)$ where $\nu = \lfloor T_c/INT \rfloor$

As a result of this simple modification, the tag's key is evolved one per epoch determined by INT and forward security is trivially achieved. The extra cost is due to the ν hash operations on the tag in step 2.4.6. If a tag i is compromised during epoch j, its key K_i^j is j-th evolution of the original key K_i^0. Due to one-way key evolvement, knowledge of K_i^j makes The adversary is faced with the following decision problem: given K_i^j, distinguish $HMAC_{K_i^s}(R_t, R_r)$ from a random value.

Timing Attacks: We claim that YA-TRIP and YA-TRAP are immune to crude timing attacks that aim to determine the tag's state (whether it is desynchronized) or its T_t value. From the timing perspective, Steps 2.2 and 2.3 in Figures 1 and 2) are indistinguishable since $PRNG$ and $HMAC$ are assumed to execute in the same time. However, YA-TRAP* is clearly vulnerable to timing attacks. Note that the execution the last OR clause in step 2.3 in Figure 3 is dependent upon the offset of T_r which can be freely selected by the adversary. Consequently, the adversary can mount a timing attack aimed at determining the epoch corresponding to the tag's last time-stamp of use (T_t). This is because the number of repeated hash operations in step 2.3 is based on $\nu = \lfloor T_r/INT \rfloor - \lfloor T_t/INT \rfloor$. One obvious countermeasure is to artificially "pad" the number of iterated hashes or introduce a random delay in tag's reply to the reader. However, we consider such countermeasures to be counterproductive as they increase protocol execution time which is undesirable, especially in batch mode.

As pointed out by Juels and Weis [10], YA-TRAP+ proposed by Burmester, et al. [4] (as well as our YA-TRAP) is susceptible to timing attacks whereby the adversary can distinguish between these two cases: (1) normal, synchronized tag and (2) desynchronized tag, based on timing on the server side. This is possible because (1) requires the server to perform a quick table lookup, whereas, (2) requires it to perform a brute-force search. This attack is only applicable in real-time mode since server operation in batch mode is not subject to being timed by the adversary.

2.7 Efficiency Considerations

We now consider the respective costs of the three protocols described above.

YA-TRIP is very efficient for tags. When an acceptable T_r is received, the computational burden on the tag is limited to a single keyed hash computation (e.g., an HMAC). Otherwise, a tag is required to generate a pseudo-random value

(via $PRNG$), which, as discussed earlier, also amounts to a single keyed hash. Again, we stress that the two cases are indistinguishable with respect to their runtime. The reader is not involved computationally in YA-TRIP, YA-TRAP or YA-TRAP*, since it neither generates nor checks any values.

YA-TRAP requires a tag to perform two extra keyed hash operations (for a total of three): one to produce R_t in step 2.4 and the other – to compute H_{auth} in step 2.5.

In YA-TRAP*, a tag also performs three keyed hash operations (in either step 2.3 or 2.4) and, in addition, needs to compute ν hashes over ET_t. However, we stress that, in normal operation, ν is typically either zero or one. (Note that, if $\nu = 0$, the notation $H^0(ET_t)$ resolves to ET_t).

In all three protocols, although the computational load on the server is relatively heavy, most of the work is not done in real time. The real-time (on demand) computation amounts to a simple table look-up. The server can pre-compute ephemeral tables at any time. The amount of pre-computation depends on available storage, among other factors.

The efficiency with respect to server load can be illustrated by comparison. One simple and secure approach to untraceable tag identification and authentication entails the reader sending a random challenge R_r to the tag and the tag replying with keyed hash (or encryption) of the reader's challenge R_r and the tag's own nonce/challenge R_t, e.g., $H_{id-auth} = HMAC_{K_i}(R_r, R_t)$. . The reader then forwards the reply – comprised of $H_{id-auth}, R_r$ and R_t – to the server. In order to identify the tag, the server needs to perform $O(n)$ on-line keyed hashes (or encryption operations), where n is the total number of tags. Although, on the average, the server only needs to perform $n/2$ operations to identify the tag, the work is essentially wasted, i.e., it is of no use for any other protocol sessions. Whereas, in our case, one ephemeral table can be used in the context of numerous (as many as n) protocol sessions.

The same issues arise when comparing YA-TRIP with the work of Molnar, et al. [2]. Although the MSW protocol from [2] is much more efficient than the naïve scheme we compared with above, it requires the tag to store $O(logn)$ keys and perform $O(logn)$ pseudo-random function (PRF) operations (each roughly equivalent to our keyed hash). In contrast, YA-TRIP only requires a single key on the tag and a single PRF.

As far as cost considerations, our requirement for (small) non-volatile RAM on the tag elevate the cost above that of cheapest tag types, i.e., less than 10 cents per tag. In this sense, YA-TRIP is more expensive than the one of the MSW protocols which does not require any non-volatile RAM (it only needs a physical random number generator). The other protocol presented in [2] requires tags to have non-volatile storage for a counter.

2.8 Security Analysis?

No formal security analysis of YA-TRIP, YA-TRAP and YA-TRAP* is included in this paper. However, we note that the security analysis in [6] is directly applicable to YA-TRIP. With respect to YA-TRAP, its extra feature of tag

authentication is orthogonal to tag identification in YA-TRIP. In fact, if we strip tag identification from YA-TRAP, it becomes a trivial two message one-way authentication protocol whereby a tag simply replies to a challenge from the reader (R_r) with $HMAC_{K_i}(R_t, R_r)$ which is further randomized with tag's own nonce R_t. The security of this kind of authentication has been considered in the literature.

Security of YA-TRAP* is less clear. It offers limited DoS-resistance by checking the validity of the proffered epoch token (ET_r). As pointed out above (at the end of Section 2.5), YA-TRAP* still permits some limited DoS attacks and the degree to which such attacks are possible is based on INT – the duration of the authorization epoch. We consider this to be a reasonable trade-off between security and functionality.

Some recent work by Juels and Weiss [10] examined the properties of YA-TRIP as it originally appeared in [1]. They conclude that, in YA-TRIP, the adversary can easily "mark" a victim tag by feeding it an arbitrary future time-stamp (T_r) and later recognize/identify the same tag seeing whether it fails in an interaction with a genuine reader. This "attack" makes two assumptions: (1) that the success or failure of tag-reader interaction is observable by the adversary and (2) that the desynchronized tag(s) is/are unique. While the second assumption is perhaps realistic, the first one is not, at least in many practical settings. The first assumption is unrealistic in, for example, warehouse or inventory settings or wherever the interaction concludes without some publicly visible effect (such as a door opening). We also note that, even if both assumptions hold, the claimed attack has limited effect in the context of YA-TRAP* due to the DoS-resistance feature[6].

3 Related Work

There is a wealth of literature on various aspects of RFID security and privacy; see [11] for a comprehensive list. We consider only related work that seems relevant for comparison with our approach, i.e., protocols that emphasize efficient server computation, involve minimal 2-round reader-tag interaction, and aim to reduce tag requirements and computation. (Of course, this rules out some other notable and useful results.)

The first notable result is the set of MSW protocols by Molnar, et al. [2] which we use in section 2.7 above in comparing the efficiency of our protocols. The approach taken is to use hierarchical tree-based keying to allow for gradual and efficient tag identification/authentication. Tree-based keying involves a tree of degree k and depth $t = \log_k(n)$ where n is the total number of tags. Each tag holds $t = \log_k(n)$ distinct keys and, upon being challenged by a reader, responds with t MACs each based on a different key. The server can then identify/authenticate a tag with $O(\log_k(n))$ complexity. YA-TRIP and YA-TRAP are more efficient for tags in terms of computation and storage since the number of on-tag cryptographic operations (1 and 2, respectively) and tag storage

[6] We say "limited" since the attacker can still mark and recognize a tag, but, only within a single epoch.

requirements (1 key in each) are independent of the total number of tags. YA-TRAP* is less efficient: 3 cryptographic operations plus ν hashes to validate the epoch token. However, in normal operation, ν is either zero or one. (It is zero since a tag might be successfully and legitimately queried many times within one epoch.) MSW protocols appear more efficient on the server side since $t << n$. Nonetheless, if $O(n)$ sensors are queried with the same T_r value, the total server cost of MSW is $O(n \cdot \log_k(n))$. In contrast, our server cost is always $O(n)$ regardless of the number of tags queried. MSW protocols also have a security "feature" whereby an adversary who compromises one tag, is able to track/identify other tags that belong to the same families (tree branches) as the compromised tag. This vulnerability of MSW protocols has been explored by Avoine, et al. [9]. The same concern does not arise in our protocols since no two tags share any secrets. Finally, we note that MSW protocols are not easy to amend in order to support forward security.

Based in part on the early (and preliminary) version of this work [1], Burmester, et al. [4] came up with a secure universally-composable framework for RFID protocols. One of their sample protocols (called YA-TRAP+) is almost identical to YA-TRAP and, although they were developed independently and (most likely) concurrently, [4] was published earlier. As we mention above, in real-time mode, YA-TRAP+ and YA-TRAP are susceptible to some timing attacks. However, we note that, due to limited DoS-resistance, YA-TRAP* is not vulnerable to timing attacks, assuming that the server pre-computes all tables for each value of T_r within the current epoch[7].

Another proposal by Avoine and Oechslin (AO) [3] is similar in spirit, but very different in technical details, from the protocols in this paper. The AO approach involves offers built-in forward security (via one-way key evolvement for each tag query). It is based on a previously proposed OSK (Ohkubo-Suzuki-Kinoshita) protocol [5] coupled with Hellman's time/memory trade-off technique [8]. Because the tag's key evolves for each query, an active attacker can incapacitate any tag by repeatedly querying it; albeit, the number of queries might be large, as determined by the length of the hash chain, e.g., 2^{10}. Also, the AO and OSK protocols do not offer tag authentication; like YA-TRIP they offer only *tag identification*. It is simple to extend AO/OSK protocols to incorporate tag authentication and DoS-resistance along the lines of our approach in YA-TRAP and YA-TRAP*. Juels and Weis [10] show some potential vulnerabilities of the AO/OSK protocols.

Acknowledgements

Many thanks to Stephan Engberg, David Molnar, Ari Juels, Xiaowei Yang and Einar Mykletun for helpful input on early versions of this paper. We are also grateful to the PET'07 anonymous referees for their insightful comments.

[7] This would entail server maintaining up to INT distinct hash tables and looking up the H_{id} value over all of them. If hash tables are looked up in random order, the overall lookup time will be similarly randomized. Thus, timing attacks would not apply.

References

1. Tsudik, G.: Yet Another Trivial RFID Authentication Protocol, IEEE PerCom (Work-in-Progress Session) (March 2006)
2. Molnar, D., Soppera, A., Wagner, D.: A Scalable, Delegatable Pseudonym Protocol Enabling Ownership Transfer of RFID Tags. In: Workshop in Selected Areas in Cryptography (August 2005)
3. Avoine, G., Oechslin, P.: A Scalable and Provably Secure Hash-Based RFID Protocol, PerSec Workshop (March 2005)
4. Burmester, M., de Medeiros, B., Van Le, T.: Provably Secure Ubiquitous Systems: Universally Composable RFID Authentication Protocols, IEEE/Createnet Securecomm (September 2006)
5. Ohkubo, M., Suzuki, K., Kinoshita, S.: Efficient hash-chain based RFID privacy protection scheme. In: UBICOMP Workshop on Privacy: Current Status and Future Directions (2004)
6. Herzberg, A., Krawczyk, H., Tsudik, G.: On Traveling Incognito. In: IEEE Workshop on Mobile Systems and Applications (December 1994)
7. Ateniese, G., Herzberg, A., Krawczyk, H., Tsudik, G.: On Traveling Incognito. Computer Networks 31(8), 871–884 (1999)
8. Hellman, M.: A cryptanalytic time-memory tradeoff. IEEE Transactions on Information Theory 26, 401–406 (1980)
9. Avoine, G., Dysli, E., Oechslin, P.: Reducing Time Complexity in RFID Systems. In: Preneel, B., Tavares, S. (eds.) SAC 2005. LNCS, vol. 3897, Springer, Heidelberg (2006)
10. Juels, A., Weis, S.: Defining Strong Privacy for RFID, IACR eprint (April 2006)
11. Avoine, G.: Security and Privacy in RFID Systems: Bibliography (February 2007), `http://lasecwww.epfl.ch/~gavoine/rfid/`
12. Steiner, J., Neuman, B., Schiller, J.: Kerberos: An Authentication Service for Open Network Systems. In: USENIX Winter 1988 Technical Conference, pp. 191–202 (1988)
13. Redl, S., Weber, M., Oliphant, M.: GSM and Personal Communications Handbook, Artech House (May 1998) ISBN 13: 978-0890069578
14. Krawczyk, H.: Simple forward-secure signatures from any signature scheme. In: ACM Conference on Computer and Communications Security, pp. 108–115. ACM Press, New York (2000)
15. Anderson, R.: Two remarks on public-key cryptology, Invited Talk. In: ACM Conference on Computer and Communications Security, ACM Press, New York (1997)
16. Tan, C., Sheng, B., Li, Q.: Serverless Search and Authentication Protocols for RFID. In: IEEE PerCom'2007, IEEE Computer Society Press, Los Alamitos (2007)
17. Juels, A., Syverson, P., Bailey, D.: High-Power Proxies for Enhancing RFID Privacy and Utility. In: Danezis, G., Martin, D. (eds.) PET 2005. LNCS, vol. 3856, Springer, Heidelberg (2006)
18. Weis, S., Sarma, S., Rivest, R., Engels, D.: Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems. In: Security in Pervasive Computing Conference (SPC'03) (March 2003)
19. Juels, A.: Yoking-Proofs for RFID Tags. In: Workshop on Pervasive Computing and Communication Security (PerSec) (2004)