YA-TRAP: Yet Another Trivial RFID Authentication Protocol

Gene Tsudik
CS Department, University of California, Irvine
Email: gts_AT_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 work describes a simple technique for inexpensive untraceable identification of RFID tags. The proposed protocol (called YA-TRAP) involves minimal interaction between a tag and a reader and places low computational burden on the tag (a single keyed hash). It also imposes low computational load on the back-end server.

1. Introduction

RFID tags are rapidly becoming ubiquitous. They are expected to replace barcodes as the means of product or item identification. RFIDs, unlike barcodes, do not require a line-of-sight channel and their smaller form factor takes up less valuable packaging "real estate". However, their proliferation into many spheres of everyday life raises numerous privacy-related concerns. One of the main issues is tracking of RFID-equipped items. While tracking RFID tags is typically one of the key features and goals of a legitimate RFID system, *unauthorized* tracking of RFID tags by rogue readers is viewed as a major privacy threat.

This paper describes a simple technique (called YA-TRAP) for inexpensive untraceable identification of RFID tags. *Untraceable* means that it is computationally difficult to infer – from interactions with a tag – information about the identity of the tag or link multiple manifestations of the same tag. YA-TRAP is inexpensive, requiring one light-weight cryptographic operation on the tag and storage for one key. It is particularly well-suited for the **batch mode** of tag identification (see below for details). Furthermore, real-time computational load on the back-end sever is minimal due to the pre-computation technique described below.

2. Operating Environment

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, etc. A server is assumed to be physically secure and not attackable. Multiple readers are assigned to a single server which 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 server and 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 a tag has no clock and small amounts of ROM and non-volatile RAM With power supplied by a reader a tag is can perform a modest amount of computation and commit any necessary state – of small constant length – to non-volatile storage. The *adversary*, in our context, can be either passive or active: it can corrupt or, attempt to impersonate, any entity and we assume that its primary goal is to **track** RFID tags.

3. Non-Security Goals

As usual, our goals are to *minimize everything*, including: (1) non-volatile RAM on the tag, (2) ROM on the tag (both code and data), (3) tag computation, (4) # of messages & rounds in reader-tag interaction, (5) message size in reader-tag interaction, (6) server real-time computation, and (7) server storage

While all cost factors matter, the first three directly influence tag cost and are thus more important than the rest. We also need to avoid features currently impractical for most RFID tags, such as: public key cryptography, tamper-resistant shielding or an on-board clock.

4. Modes of Operation

We consider two modes of tag identification: *real-time* and *batch*. The former is the mode typically considered in the literature: it involves on-line contact between the reader and the server, in order to quickly authenticate (or identify) the tag in question. If immediate feedback about a tag is needed (as, for example, in popular retail or library check-out scenarios), the server must be contacted in real time. In other applications, such as inventory control, where readers are mobile while items equipped with tags are stationary, *batch* mode can be used, whereby a reader scans numerous tags, collects replies and sometime later performs their identification in bulk.

Clearly, batch mode is only relevant in settings where there is no concern about fraudulent tags, whereas, emphasis is on security against potentially fraudulent readers. However, we consider it to be an important point on the overall design space of RFID security techniques. In particular, if the server is simply not available (e.g., unreachable or overloaded) or if mobile/wireless readers need to conserve battery power[1], batch mode becomes quite attractive.

5. Tag Requirements

Each tag $RFID_i$ is initialized with at least the following values: K_i, T_0, and T_{max}.

K_i is a tag-specific value that serves as both: (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., based on the total number of tags) and that required to serve as a sufficiently strong cryptographic key for the purposes of Message Authentication Code (MAC) computation. In practice, a 160-bit K_i will likely suffice.

T_0 is the initial timestamp assigned to the tag. In a most likely scenario, T_0 can be the timestamp 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 top value for the timestamp. 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 $i = j$ is computationally hard for any entity.

[1] Contacting a server for each scanned tag can be very communication-intensive, much more so than storing all tag replies and batching them later.

6. Overview

The main idea of our proposal is the use of monotonically increasing timestamps to provide tracking-resistant (anonymous) tag authentication. The use of timestamps is motivated by the old result of Herzberg, et al. [1], which we briefly summarize next.

The work in [1] considered anonymous authentication of mobile users who move between domains, e.g., in a GSM cellular network or a wired Kerberos-secured internetwork. Their technique 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[2] updated hash table where each row corresponds to a traveling user. (The table can be either pre-computed or computed 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 in a remote domain or VLR in 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'

[2] The length of the update interval is a system-wide parameter, e.g., one hour.

and gets replaced with a new one. This is the main feature we would like to *borrow* for tag authentication purposes.

Although the technique from [1] 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 timing device which 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 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 unable to distinguish among a legitimate and a grossly inaccurate timestamp.

However, if the tag keeps state of the last timestamp it "saw" (assuming it was legitimate), then it can distinguish between future (valid) and past (invalid) timestamps. 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 timestamp to the stored timestamp value. If the former is strictly greater than the latter, the tag concludes that the new timestamp is probably valid and computes a response derived from its permanent key and the new timestamp. A tag thus never accepts a timestamp earlier than the one stored. However, to protect against *narrowing* attacks[3] , 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.

7. YA-TRAP Protocol

The protocol is illustrated below.

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

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 returns a meta-id of the tag: $G(K_s)$ where $G(.)$ is a suitable cryptographic hash with the usual features. Otherwise, it suffices to inform the reader that the tag in question is valid.

1 Tag $\longleftarrow$ Reader: T_r

2 Tag:

[2.2] if ($(T_r - T_t) \leq 0$) OR $(T_r > T_{max})$

$H_r = PRNG_i^j$

[2.3] else $T_t = T_r$, $H_r = HMAC_{K_i}(T_t)$

3 Tag $\longrightarrow$ Reader: H_r

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_r

5 Server:

[5.1] Let $s = LOOKUP(HASH\_TABLE_{T_r}, H_r)$

[5.2] If $(s == -1)$ MSG=TAG-ERROR

[5.3] Else $MSG = G(K_s)$ (or $MSG =$ *"VALID"*)

6 Server $\longrightarrow$ Reader: MSG

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)[4] to the server. The server then needs to identify the tags. In this situations, YA-TRAP is highly advantageous. Even currently most efficient techniques such as the MSW protocol [2], require the server to perform $O(logn)$ pseudo-random function (PRF) operations to identify a single tag. This translates into $O(n * logn)$ operations to identify n tags. Whereas, YA-TRAP would only need $O(n)$ operations for the same task (since the same T_r-specific hash table is used for all lookups and each lookup takes constant time).

7.1 Drawbacks and Extensions

YA-TRAP has two potential drawbacks.

First, it is susceptible to a trivial denial-of-service (DoS) attack: the adversary can send a wildly inaccurate timestamp (T_r) and incapacitate a tag either fully (if the timestamp is the maximal allowed) or temporarily. Although DoS resistance is not among our key goals, it is still an important issue. Unfortunately, there does not seem to be an easy way of addressing this issue (without additional computation on the tag and extra bandwidth).

[3]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.

[4]If tag responses are collected over multiple time intervals, the reader needs to group responses according to the T_r value used.

Second, the protocol makes an implicit assumption that a 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 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.)

7.2 Efficiency/Cost Considerations

The proposed protocol is fairly efficient. 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 HMAC. Again, we stress that the two cases are indistinguishable with respect to their runtime. The reader is not involved computationally in YA-TRAP, since it neither generates nor checks any values. The computational load on the server is not light, by any means. However, it does not require any on-demand (real-time) computation other than a simple table look-up. The server has the *freedom* to (pre-)compute ephemeral tables at any time. The amount of pre-computation would likely depend on available storage, among other factors.

The efficiency of our protocol as far as server load can be illustrated by comparison. One simple and secure approach to untraceable tag identification involves 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 H_t and the tag's own random confounder/nonce R_t. The reader forwards the reply – comprised of H_t, 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 encryptions), 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 has no use for any other protocol sessions. Whereas, in our case, an ephemeral table is can be used for multiple (as many as n) protocol sessions.

The same issues arise when comparing YA-TRAP with the recent work in [2], which represents the state-of-the-art. Although the MSW protocol [2] is much more efficient than the naïve scheme above, it requires the tag to store $O(logn)$ keys and perform $O(logn)$ pseudo-random function (PRF) operations. YA-TRAP requires a single key on the tag and a single PRF.

As far as cost, our requirement for non-volatile RAM elevate the cost above that of cheapest tags, i.e., less than \$0.10 per tag. In this sense, YA-TRAP is more expensive than the one of the MSW protocols which makes do without non-volatile RAM (it only needs a physical random number generator).[5]

8. What is not covered in this paper?

Due to space limitations, security properties of YA-TRAP are not addressed. However, most of the security analysis in [1] applies to YA-TRAP. The full version of this paper will include a detailed discussion. Also overview of related work is deferred to the full version of this paper. While we use [2] as one point of comparison, a recent protocol by Avoine and Oechslin [3] is similar in spirit (but very different in technical details) from YA-TRAP.

Acknowledgements

Many thanks to David Molnar, Ari Juels, Einar Mykletun and Joao Girao for their helpful comments.

References

[1] A. Herzberg, H. Krawczyk and G. Tsudik, *On Traveling Incognito,* IEEE Workshop on Mobile Systems and Applications, December 1994.

[2] D. Molnar, A. Soppera and D. Wagner, *A Scalable, Delegatable Pseudonym Protocol Enabling Ownership Transfer of RFID Tags,* Workshop in Selected Areas in Cryptography, August 2005.

[3] G. Avoine and P. Oechslin, *A Scalable and Provably Secure Hash-Based RFID Protocol,* IEEE PerSec Workshop, March 2005.

[5] The other protocol presented in [2] requires tags to have non-volatile storage for a a counter.