In Proceedings of Advances in Web and Network Technologies and Information Management, 2007, Springer, pp. 634-643

Coexistence Proof Using Chain of Timestamps for Multiple RFID Tags*

Chih-Chung Lin[1], Yuan-Cheng Lai[1], J.D. Tygar[2],
Chuan-Kai Yang[1], and Chi-Lung Chiang[1]

[1] Department of Information Management,
National Taiwan University of Science and Technology
D9409106@mail.ntust.edu.tw, laiyc@cs.ntust.edu.tw,
[2] University of California, Berkeley
doug.tygar@gmail.com

Abstract. How can a RFID (Radio Frequency Identification Devices) system prove that two or more RFID tags are in the same location? Previous researchers have proposed *yoking-proof* and *grouping-proof* techniques to address this problem – and when these turned out to be vulnerable to replay attacks, a new *existence-proof* technique was proposed. We critique this class of existence-proofs and show it has three problems: (a) a race condition when multiple readers are present; (b) a race condition when multiple tags are present; and (c) a problem determining the number of tags. We present two new proof techniques, *a secure timestamp proof* (secTS-proof) and a *timestamp-chaining proof* (chaining-proof) that avoid replay attacks and solve problems in previously proposed techniques.

Keywords: RFID, coexistence proof, timestamp, computer security, cryptographic protocol, race condition.

1 Introduction

Radio Frequency Identification Devices (RFID), are supported by systems comprised of wireless readers and tags, and allows objects to be identified and, in some cases, tracked. [1] The most commonly used tags, passive tags, are inexpensive devices powered by radio signals from readers. They have sharply limited memory and processing capabilities. Some argue that RFID tags allow less expensive inventory management capabilities. [2] One important issue for RFID systems is generation of coexistence proofs demonstrating two or more RFID tags are simultaneously located. The key contribution of this paper is a critique of previous RFID coexistence proofs and a set of new proofs avoiding previous shortcomings.

* This research was supported in part with funding from the iCAST project NSC95-3114-P-001-002-Y02 in Taiwan, from the US National Science Foundation, from the US Air Force Office of Scientific Research, and from the Taiwanese National Science Council. The opinions expressed in this opinion are solely those of the authors and do not necessarily reflect the opinions of any funding sponsor.

K.C. Chang et al. (Eds.): APWeb/WAIM 2007 Ws, LNCS 4537, pp. 634–643, 2007.

Here are some motivating examples:

1) *Medical care*: a medical professional can prove that a set of correct drugs, blood products, or other medical materials are brought together for patient needs. The proof can be retained in case of dispute or for insurance purposes. [3]

2) *Transportation*: a transport or logistic firm can prove that items are always stored together in a safe box, if RFID tags are on the box and contents. [3]

3) *Forensics*: if RFID tags are on phones or other personal devices, law enforcement can use it to identify witnesses to a crime.

Here is a brief summary of prior work: Juels defined *yoking-proofs* which use a random number independently generated by tags to produce a coexistence proof. [3] Saito and Sakurai observed that yoking-proofs are vulnerable to replay attacks and proposed *grouping-proofs* that request timestamps from a trusted server. [4] Piramuthu observed that grouping-proofs were still vulnerable to replay attacks and proposed *existence-proofs* that keep a random number in the tag memory and sets the inputs of one tag to information generated by a second tag. [5] We can group these techniques into those using an off-line verifier (yoking-proofs) and those using an online verifier (grouping-proofs and existence-proofs).

We show existence-proofs have three problems:

- A race condition when multiple readers are present;
- A race condition when multiple tags are present; and
- Difficulties in determining the number of tags.

We give two new proof techniques:

- a *secure timestamp proof* (secTS-proof) – a proof technique based on a secure online verifier that issues secure timestamps; and
- a *timestamp-chaining proof* (chaining-proof) – a proof technique based on a secure off-line verifier. [6]

Both schemes avoid the replay attack, and can scale up to support proofs of arbitrary number of tags in a simultaneous environment.

Notation:

- *OV*: a trusted online verifier
- *FV*: a trusted off-line verifier
- *TSD*: a trusted timestamp database, which stores timestamps and message authentication codes from readers
- *TS*: a timestamp
- x: a symmetric key
- r: a random number
- *MAC*: message authentication code.
- $MAC_x[m]$: the MAC of message m under key x
- $SK_x[m]$: the encryption of m under key x
- P_{AB}: a proof that tags A and B were scanned simultaneously
- $P_{1\sim n}$: a proof that tags 1, 2, 3, …, n were scanned simultaneously

2 Related Work

2.1 Yoking-Proof

In yoking-proofs, the reader interacts with two RFID tags, T_A and T_B, and an off-line verifier (*FV*). T_A and T_B share secret keys x_A and x_B with *FV* and generate random numbers r_A and r_B, respectively, in every session. Figure 1 gives the protocol. After receiving P_{AB}, the reader forwards P_{AB}, r_A and r_B to *FV*.

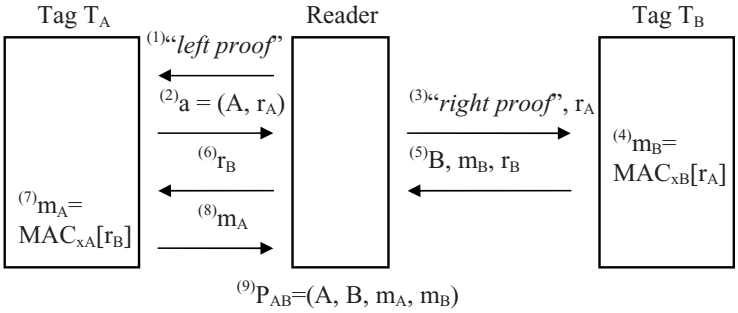

Fig. 1. Yoking-Proof

Because each tag uses a random number to compute a MAC, an adversarial reader can perform a replay attack by reusing previously generated random values. Saito and Sakurai showed a replay attack on T_A (see Figure 2 – the dashed line indicates the reader interacts with T_A and T_B at different times) [4] and Piramuthu showed a replay attack on T_B [5]. The yoking-proof technique cannot be repaired since the adversarial reader can send P_{AB}, r_A and r to *FV* for verification and ignores r_B.

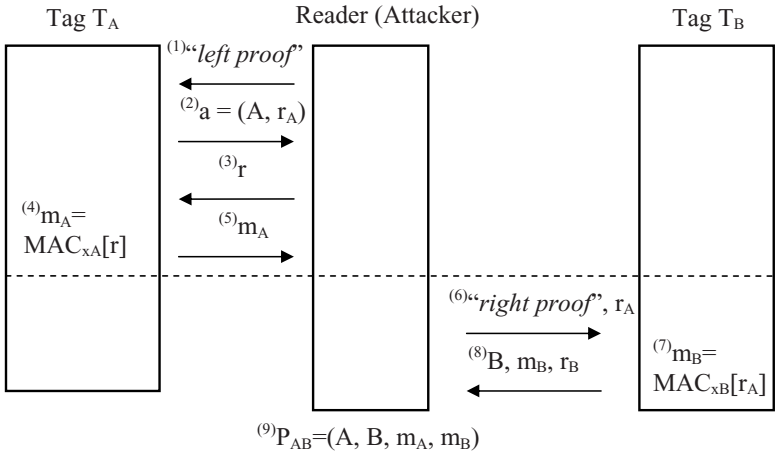

Fig. 2. Replay attack against Yoking-Proof

2.2 Grouping-Proof

Saito and Sakurai give a *grouping-proof* technique with the intention of avoiding replay attacks. [4] The reader acquires a timestamp (*TS*) from an on-line verifier (*OV*)

and sends it to T_A and T_B. T_A and T_B individually compute m_A and m_B using the secret keys x_A and x_B – see Figure 3. After receiving proof P_{AB}, the reader sends P_{AB}, A, and B to OV. Grouping-proofs rely on a timeout mechanism – if the OV receives P_{AB} at time more than $TS+\Delta$, it rejects the proof.

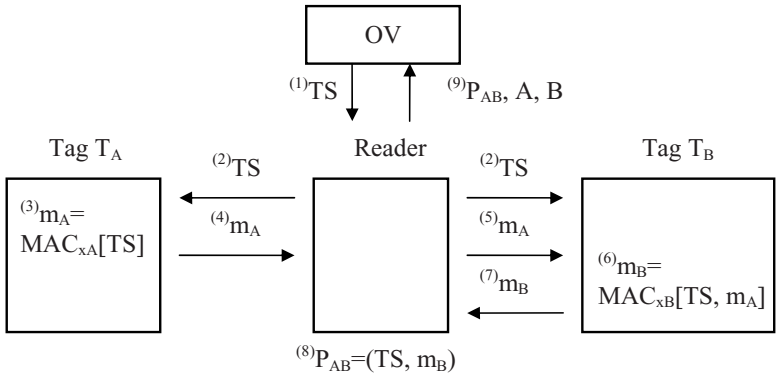

Fig. 3. Grouping-Proof

Piramuthu shows grouping-proofs are vulnerable to replay attacks (see Figure 4.). [5] An adversarial reader repeatedly transmits different future timestamps to tag T_A, generating different (TS, m_A) pairs. At the future time, the adversarial reader transmits these combinations to the tag T_B and then sends P_{AB} to OV. Note that the adversarial reader formally acquires TS from OV.

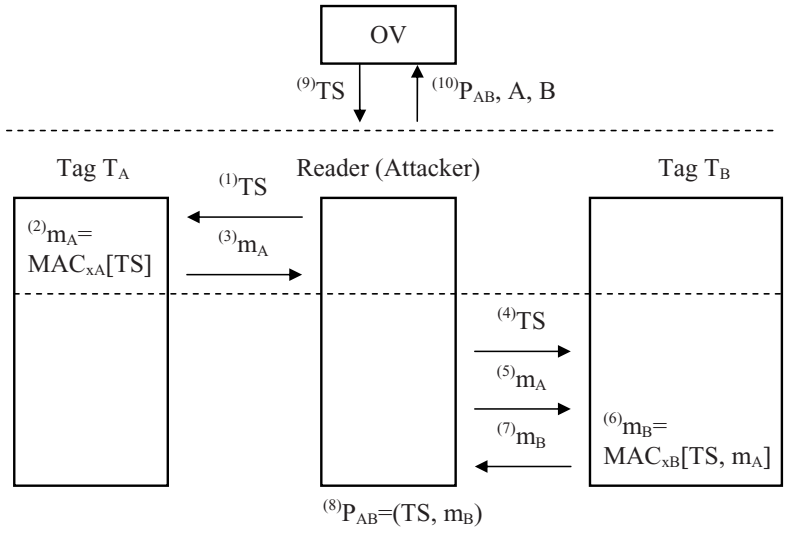

Fig. 4. Replay attack against Grouping-Proof

Saito and Sakurai extend their scheme to prove coexistence of multiple tags (Figure 5). Their scheme needs two types of tags, *product tags* and *pallet tags*. Product tags function similarly to the T_A or T_B discussed above. Pallet tags can compute symmetric key encryption and have larger memory stores than product tags.

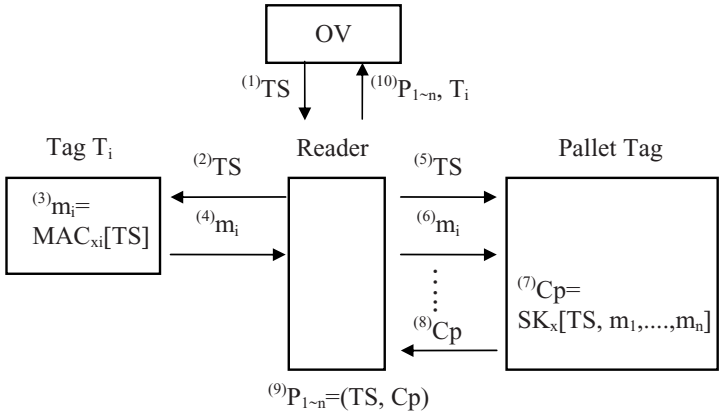

Fig. 5. Grouping-Proof for multiple tags

Product tags and a pallet tag share their secret keys with *OV*. The reader gathers n MACs from the product tag T_i ($1 \le i \le n$) and sends them to the pallet tag. The pallet tag encrypts n MACs m_i and *TS* to generate the ciphertext *Cp*. After the reader receives *Cp* from the pallet tag, it sends $P_{1\sim n}$ and all T_i ($1 \le i \le n$) to *OV*. *OV* first checks whether $P_{1\sim n}$ are within the timeout range. *OV* decrypts *Cp* using x to get m_i. *OV* verifies m_i using x_i. Note that this approach is also vulnerable to replay attacks.

2.3 Existence-Proof

Piramuthu proposes *existence-proofs* with the intention of avoiding replay attacks. [5] His idea is to ensure that inputs to a tag depend on information generated by other tags. Figure 6 shows his approach:

- The reader requests random number r from *OV*, which in term is a seed for generating r_A and r_B by tags T_A and T_B.
- T_B generates m_B which depends on both r and r_A.
- T_A uses m_B and r_A to generate m_A.

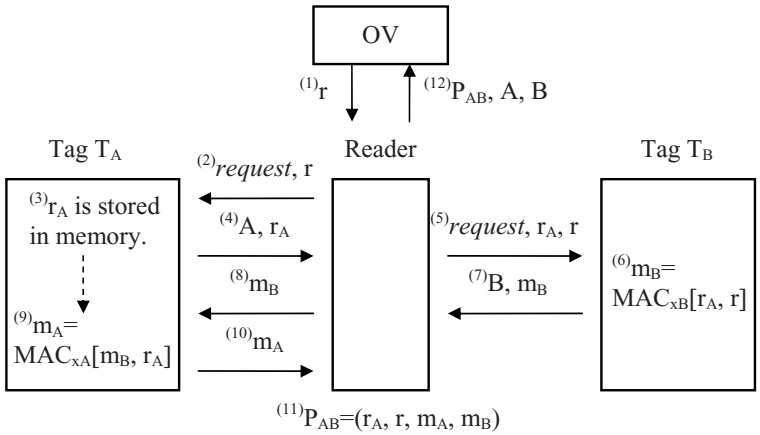

Fig. 6. Existence-Proof

Since both T_A and T_B rely on values generated by the other, it is robust against replay attacks. This scheme also uses a timeout mechanism to ensure the freshness of proofs.

3 Problems of Existence-Proofs

While existence-proofs avoid replay attacks, they have other problems. First, when tag T_A interacts with multiple readers, a race condition can occur (Figure 7). Reader1 sends r_1 to T_A and Reader2 sends r_2 to T_A almost simultaneously. T_A generates r_{1A} and r_{2A}, stores them in the memory, and transmits them to Reader1 and Reader2. After Reader1 and Reader2 individually interact with their other tags, i.e. T_{1B} or T_{2B}, they send m_{1B} and m_{2B} to T_A almost simultaneously. T_A does not know which r_A (r_{1A} or r_{2A}) should be used with m_{1B} or m_{2B} to generate m_{1A} or m_{2A}, causing a race condition.

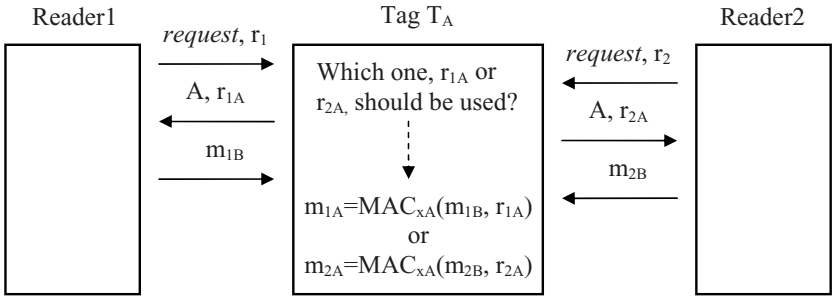

Fig. 7. Race condition for multiple readers

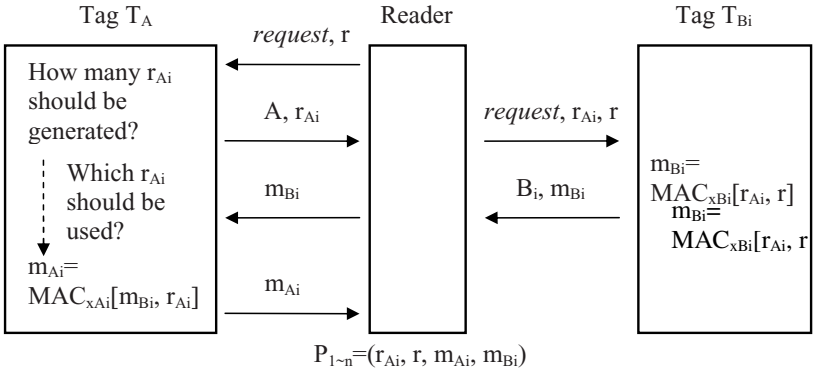

Fig. 8. Race condition for multiple tags and determining the number of tags

Piramuthu designed existence-proofs to scale to handle multiple tags. Tag T_A generates r_{Ai} (i=1, … , n-1, where n is the number of tags) by partitioning r into n-1 parts and the reader generates m_{Bi} according to each r_{Ai}. In the end, $P_{1\sim n}$ is composed of r_{A1}, r_{A2}, …, $r_{A(n-1)}$, r, m_{A1}, m_{A2}, …, $m_{A(n-1)}$ and m_{B1}, m_{B2}, …, $m_{B(n-1)}$. However, because T_A does not know how many r_{Ai} should be generated, it cannot determine the number of tags (see Figure 8.) Even if the reader can give the number of T_{Bi} to T_A, T_A

does not know which r_{Ai} is used with m_{Bi} to generate m_{Ai}, causing a different type of race condition, which we call the race condition for multiple tags.

4 Two Proposed Coexistence-Proofs

Above, we distinguished between systems with online verifiers and off-line verifiers. Below we give two proposed proof types, *secTS-proof* (with an online verifier) and *chaining-proof* (with an off-line verifier.)

Figure 9 shows the secTS-proof. To prevent adversarial readers from generating bogus timestamps, when the reader requests *OV*, *OV* generates a random number *r* and uses its secret key *x* to encrypt *TS* and *r* to create a unique *S*. *OV* also checks the freshness of proofs – if a proof is submitted after *TS*+Δ, it is rejected.

Figure 10 shows the chaining-proof. In this scheme, the reader can issue the timestamp by itself. Because there is not an on-line verifier to monitor the reader's behavior, an attacker may issue a bogus timestamp. We use Haber-Stornetta timestamps [6] to avoid attack; each new timestamp is formed by taking a hash and using the hash value and MAC from previous timestamps. Because the reader does not have the tag's secret key, the timestamp can be generated until the last timestamp has been obtained. To complete verification, the reader must report the last timestamp, tag id, and timestamp MAC computed by the tag to an offline trusted third party (timestamp database *TSD*). When *TSD* receives the timestamp, it marks the timestamp information with a trusted time value.

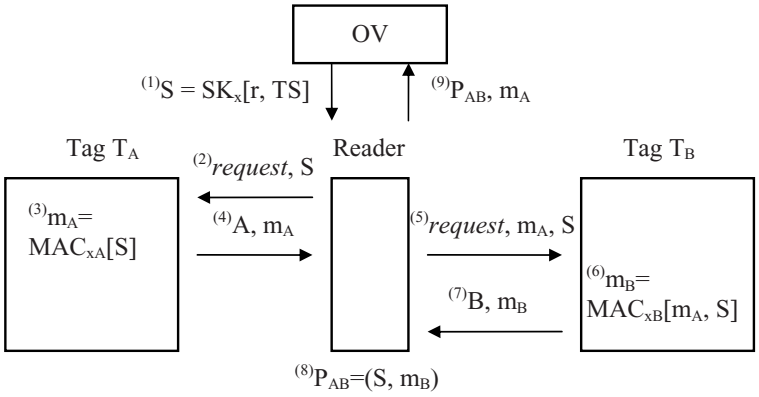

Fig. 9. SecTS-Proof

In the chaining-proof, each tag shares its secret key x_i with *FV*. Furthermore, the reader reports timestamps $ms_i=(T_i, TS_i(h(ms_{i-1})), m_i)$ to *TSD*, where *i* indicates the sequence that the reader scans the tags. ms_0 is a random number *r* acquired from *TSD*. Once *TSD* receives one ms_i, *TSD* stores and combines ms_i with a time (RT_i) which means when *TSD* receives this ms_i. The procedure of chaining-proof for multiple tags is described as follows:

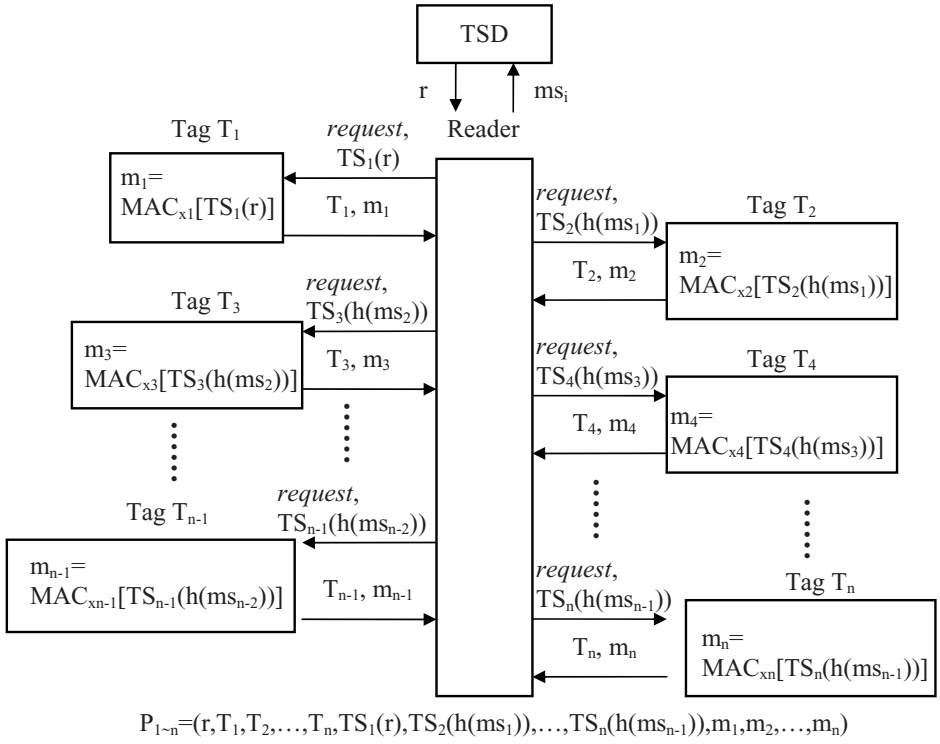

Fig. 10. Chaining-Proof

1. *TSD* gives a random number *r* to the reader and combines *r* with a time, RT_0.
2. The reader issues TS_1 including a random number *r* to the tag T_1.
3. T_1 generates m_1 by applying x_1 to TS_1, and sends its id T_1 and m_1 to the reader.
4. The reader reports $ms_1=(T_1 \| TS_1(r) \| m_1)$ to *TSD*, and *TSD* stores and combines it with a time, i.e. RT_1.
5. The reader sends TS_2 including $h(ms_1)$ to the tag T_2. $h(\ )$ is a one-way hash function.
6. T_2 use x_2 on TS_2 to generate m_2, and submits its id T_2 and m_2 to the reader.
7. The reader reports $ms_2=(T_2 \| TS_2(h(ms_1)) \| m_2)$ to *TSD*, and *TSD* stores and combines it with a time, i.e. RT_2.
8. The remainder tags and the reader repeat until the reader gets all MACs of nearby tags.
9. When verifying, *FV* receives the proof $P_{1\sim n}$ from the reader.
10. *FV* extracts ms_i from $P_{1\sim n}$ and sequentially acquires RT_i from *TSD* according to ms_i.
11. *FV* checks that the duration between RT_i and RT_{i-1} is less than Δ, a predefined time threshold.
12. After *FV* checks all time durations, if no any problem exists, *FV* verifies each MAC m_i by using the corresponding secret key x_i and $TS_i(h(ms_{i-1}))$.
13. Finally, *FV* verifies whether each TS_i is located between RT_0 and RT_{n-1}. If so, TS_i are accepted. If no problem exists, the proof is.

5 Sketch of Security Proof

Space does not permit a full proof of security; so here we just provide a sketch of security for the chaining-proof. (SecTS-proofs are substantially simpler to show security for.) We assume an adversary can control one or more reader, but not a verifier or *TSD*. We assume that our cryptographic functions observe standard requirements (see [6] for a fuller discussion of security of the timestamp mechanism.). Now the sketch of the proof is straightforward. Refer to Figure 10. If the tag T_2 is not in range of an adversarial reader the attacker cannot get m_2 from T_2. Without m_2, the adversary cannot compute $ms_2=(T_2||TS_2(h(ms_1))||m_2)$ and cannot submit ms_2 to the on-line timestamp database (*TSD*). Alternatively, if the adversary waits for the tag T_2, *TSD* will mark ms_2 with the later time timestamp (RT_2) – and if this exceeds timeout range Δ, the key will be discarded. Replay attacks against T_1, T_3, T_4, …, T_n are not possible for parallel reasons.

Note that because the reader uses a random number r from *TSD* and this value is combined with a trusted time in *TSD*, *FV* can discover if an adversarial tries to collect information by interacting individually with each tag using bogus timestamps at some later time.

Note further, if a proof $P_{1234}=(r, T_1, T_2, T_3, T_4, TS_1(r), TS_2(h(ms_1)), TS_3(h(ms_2)), TS_4(h(ms_3)))$ is already verified and is valid. If an adversarial reader attempts to duplicate the same proof by using a valid r and insert existence evidence of T_5 , i.e. $(T_5, TS_5(h(ms_4))$, in this proof, the attack will be detected, since each timestamp inside the proof chains is reported to *TSD*.

6 Conclusion

We showed three RFID co-existence proof types (yoking-proofs, grouping-proofs, and existence-proofs) suffer from a number of problems: replay attacks, race conditions, and ambiguity in the number of tags. We proposed two novel proof types: secTS-proofs and chaining-proofs. SecTS-proof is applied on the environment having an online verifier while chaining-proof is used on the environment having an off-line verifier. Our two schemes successfully avoid all known attacks, including replay attacks.

References

1. Shepard, S.: RFID: Radio Frequency Identification. McGraw-Hill, New York (2005)
2. Juels, A.: Strengthening EPC tags against cloning. In: Proceedings of the 4th ACM Workshop on Wireless Wecurity, pp. 67–76 (2005)
3. Juels, A.: Yoking-proofs for RFID tags. In: Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops. 2004, pp. 138–143 (2004)

4. Saito, J., Sakurai, K.: Grouping proof for RFID tags. In: Proceedings of the 19th IEEE International Conference on Advanced Information Networking and Applications, pp. 621–624 (2005)
5. Piramuthu, S.: On existence proofs for multiple RFID tags. In: Proceedings of the ACS/IEEE International Conference on Pervasive Services, pp.317–320 (2006)
6. Haber, S., Stornetta, W.: How to time-stamp a digital document. Journal of Cryptology 3(2), 99–111 (1991)