Hash-based Mutual Authentication Protocol for Low-Cost RFID Systems

Győző Gódor and Sándor Imre

Department of Telecommunications,
Budapest University of Technology and Economics
Magyar Tudósok körútja 2., Budapest, Hungary H-1117
{godorgy,imre}@hit.bme.hu
http://www.hit.bme.hu

Abstract. In the last decade RFID technology has become widespread. It can be found in various fields of our daily life. Due to the rapid development more and more security problems were raised. Since tags have limited memory and very low computational capacity a so-called lightweight authentication is needed. Several protocols have been proposed to resolve security and privacy issues in RFID systems. However, the earlier suggested algorithms do not satisfy all of the security requirements.
In this paper we introduce our hash-based mutual authentication protocol which meets all the security requirements. Our solution provides an efficient mutual authentication method. Our protocol can defy the well-known attacks and does not demand high computational capacity.

Keywords: RFID, lightweight, authentication

1 Introduction

In the last few years the RFID (Radio Frequency Identification) technology has developed rapidly. Slather of applications were designed which steal into most of most fields of our daily life, e.g., personal identification in passports, medical identification of patients, access control, thief prevention, payment systems, tickets for transportation, supply-chain management, asset tracking, etc.

As in case of other radio frequency bases systems, e.g., wireless lans, bluetooth, cellular networks, the communication operates via the air-interface which raises many security issues. Since this kind of medium can be easily accessed by everyone the messages between the communication parties can be eavesdropped, observed or forged and further on fake messages can be composed and it can be replayed. Moreover, the communication parties can be impersonated, customers (possessing a tag) can be tracked or their transactions can be recorded.

In order to prevent the unauthorized access to protected databases or systems, or any sensitive data, authentication and encryption have to be applied. Since the tags have very limited computational capacity and low memory, new solutions are needed. Due to these conditions of RFID tags in the very beginning of the development of RFID authentication protocols only mathematical

and logical operations could be used. After the computational capacity of tags increased, simple cryptographic functions, e.g., one-way hash functions could be used and many lightweight authentication protocols were developed.

In the last years numerous RFID authentication protocols have been proposed in order to resolve these privacy problems. For example, the hash-lock protocol [1],[2], the randomized hash-lock protocol [1],[3], tree-based protocol [4], the hash-chain protocol [5] and the hash-based ID variation protocol [6],[7],[8] are representative. However, the previous protocols do not resolve privacy problems since the previous protocols can not reuse tag or the previous protocols are vulnerable to replay and spoofing attacks, and tag can be tracked.

In this paper we introduce our hash-based mutual authentication protocol which meets all the security requirements. The protocol can defy the well-known attacks and does not demand high computational capacity.

2 Related Works

In the last decade the security issues of RFID systems got into the observed of all observers. Numerous protocols were proposed. Most of them are based on cryptographic hash functions, random number generation, mathematical and logical operations. These are the so-called lightweight protocols. In this section few significant protocols are introduced.

2.1 Hash-Based Access Control (HBAC)

The protocol, which can be implemented in low-cost RFID tags, was proposed by Weis [1] in 2003. The scheme is illustrated in Fig. 1.

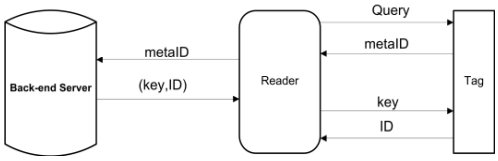

Fig. 1. Hash-Locking: A reader unlocks a hash-locked tag

In case of Hash-Locking scheme, only the key k of each tag is stored in the database of the back-end server. In the followings the authentication process of the HBAC is presented.

1. When a given tag arrives near a reader's coverage the reader sends a Query message to the tag.
2. The tag computes the $metaID = hash(ID)$ and transmits it to the reader. After that the reader forwards the received $metaID$ to the Back-end Server.
3. The Back-end Server looks up the appropriate key in the back-end database and transmits it to the tag.

4. The tag calculates the hash value of the key which is sent by the reader and compares it to the stored $metaID$. If the values match, the tag sends its own ID to the reader.

Security issues of HBAC protocol The hash-lock scheme is very elementary authentication protocol and very vulnerable. Since the key k is sent in plain-text over the air, thus an attacker can eavesdrop the key and it can be spoofed later or a given tag could be impersonated. Furthermore, since $metaID$ is always constant each tag can be identify, and trace them, in order to avoid this the $metaID$ should be changed frequently.

2.2 Randomized Access Control (RAC)

The RAC protocol [1] is an enhancement of HBAC. The aim was preventing the tracking of a given tag. While in this mode, a tag must not respond predictably to queries by unauthorized users, but must still be identifiable by legitimate readers. Tags are equipped with a one-way hash function, additionally also have a random number generator to make its constant variable randomized. Fig. 2 shows the authentication process of RAC.

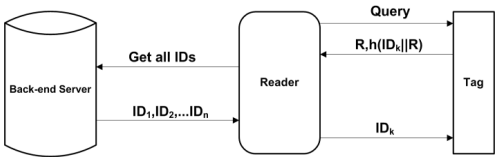

Fig. 2. Randomized Hash-Locking

1. The tag responds to the reader's Query messages by generating a random number R and then computes $h(ID \parallel R)(= hash(ID \parallel R))$ as the tag's unique identification for each session. The tag transmits $h(ID \parallel R)$ and R to the back-end server via the reader.
2. The server finds the unique identifier of the tag by comparing $h(ID \parallel R)$ with the construction of R and each ID_s which is stored in database, then authenticates itself by sending the unique identifier ID_k back to the tag.

Security issues of RAC protocol This protocol can protect against tracking and replay attack effectively because the tag's response varies on each session. However, the tags' IDs are always constant so the tag can be traced if the tag's ID is leaked out. In addition, an adversary can easily obtain a valid message pair $(h(ID \parallel R); R)$ from the tag and later the attacker can impersonate that tag to a legitimate reader. In order to identify a given tag back-end server needs to perform a brute-force search to verify the signature $h(ID \parallel R)$. If there is a big amount of tags in the database, the computation load would be very high.

2.3 Tree-based RFID protocol

Molnar proposed a mutual authentication protocol for back-end server and tags in a private way which is based on key-tree [4]. In Fig. 3 the tree-based authentication protocol can be seen.

Beginning of the authentication process, back-end server sends a random number r_r to tag. Tag also generates a random number r_t and computes $\sigma = ID \oplus f_k(0, r_r, r_t)$, where f_k is a pseudo random function. Then tag sends r_t and σ to the server. Based on the information received from the tag, the server could find the ID, k of the tag on the database.

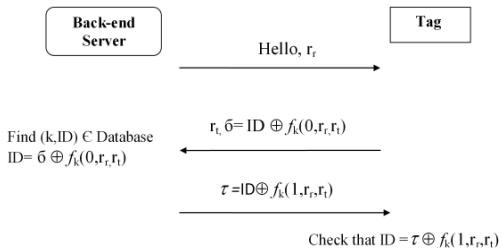

Fig. 3. Privacy and Security in Library RFID

The search on the database server is the process of searching in key-trees. A key-tree is a tree where a unique key is assigned to each edge. Each key is generated uniformly and independently. The server is assumed to know all keys. Each tag possesses the keys assigned to the edges of the path starting from the root and ending in the leaf that corresponds to the given tag.

For authenticating a reader it starts at the root and uses r_r to check whether the tag uses the "left" key or the "right" key. If the reader and the tag successfully authenticate using one of these two keys, the reader and tag continue to the next level of the tree. If the reader fails to convince the tag on any level, the tag rejects the reader. If the reader passes all secrets in the path, the tag accepts the reader. If the server finds the ID of the tag, then the tag is authenticated. Server sends back $\tau = ID \oplus f_k(1, r_r, r_t)$ to tag. Tag can thus verify the identity of back-end server by checking whether $ID = \tau \oplus f_k(1, r_r, r_t)$.

Security issues of the tree-based protocol The implementation of a tree-key authentication protocol is very easy, moreover, this kind of protocol is very scalable, enlarging the system with additional tags is simply feasible. Since the total scheme requires $O(logn)$ rounds of communication, $O(logn)$ works for the reader, and $O(logn)$ storage at the tag, where n denotes the number of tags. However, in this tree-based approach upper level keys in the tree are used by many members. This will cause problem if a member is compromised and its keys become known to the adversary, because in this case the adversary gains partial knowledge of the key of other members, too. This obviously reduces the privacy

provided by the system to its members, since by observing the authentication of an uncompromised member, the adversary can recognize the usage of some compromised keys, and therefore its uncertainty regarding the identity of the authenticating member is reduced.

This scheme cannot provide backward security. Once a tag is compromised, the attacker can trace its past communication transactions, because a tag's identifier and secret key are static.

2.4 Hash-based Untraceable Protocol (Ohkubo protocol)

The main idea of this protocol is to modify the tag's identifier after each successful authentication process in order to prevent traceability and disclosure of the identifier and key. As it can be seen in Fig. 4 two hash functions are applied, one to refresh the secret of the tag, and the other to make responses of the tag untraceable by eavesdroppers [5].

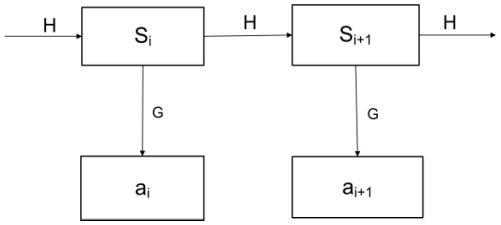

Fig. 4. Ohkubo protocol

Each tag has initial information s_1. In the i^{th} transaction when a reader queries tag, it sends an identification request to the tag and $a_i = G(s_i)$ is sent back, where s_i is the current identifier of the tag. Tag refreshes its secret $s_{i+1} = H(s_i)$ as determined from previous secret s_i, where H and G are the two hash functions. The reader sends a_i to the back-end database. From a_i, the back-end database has to identify the corresponding tag.

In order to do this, the back-end database constructs a hash chain from each n initial value s_1 until it finds the expected a_i, or until it reaches a given maximum limit m on the chain length. So the back-end database received tag's output a_i from the reader and calculates $a'_i = G(H_i(s_1))$ for each s_1 in the list, and checks if $a_i = a'_i$. If this holds, it returns the ID which is a pair of a'_i.

Security issues of the Ohkubo protocol Ohkubo protocol guarantees untraceability and forward secrecy, G is a one-way function, so if the adversary obtains tag output a_i, it cannot know s_i from a_i. G outputs random values, so if the adversary watches the tag output, it cannot link a_i and a_{i+1}. H is also a one-way function, so if the adversary tampers with a tag and obtains the secret information in the tag, it cannot know s_i from s_{i+1}, but Ohkubo requires

exhaustive search in the back-end server. The time complexity of this protocol is $O(2mn)$ in terms of hash computations. The time complexity $O(2mn)$ of Ohkubo is less scalable in comparison with Hash-Lock Scheme $O(n)$.

Ohkubo protocol is very weak in the resistance attacks by DoS. If the attacker impersonates the real tag and floods the server with fake a_i', the server should have to calculate thousands, even millions of hash function, thus it will occupy a lot of server resources and legitimate tags could not authenticate themselves.

3 The Proposed Protocol

3.1 Main idea

In this section, we propose a robust privacy preserving mutual authentication protocol which utilizes hash function and synchronized secret information. The proposed authentication protocol, which can be realized in a low-cost tag, guarantees the privacy protection for a tag holder, which requires confidentiality, untraceability, anti-cloning, and integrity.

Usually, in RFID systems, synchronization of tag's identification information between tags and back-end servers can be violated by asymmetric communication channel of air interface between tags and readers. The replay attack is also enabled for tags and back-end servers, respectively. Moreover, as the characteristic of RFID systems, a tag can emit its data to everyone including adversaries. Adversaries thus can trace location and behaviors of tag holders without detection and impersonate a legitimate tag or a legitimate reader. To remove these security and privacy problems, our protocol is based on mutual authentication between tags and back-end servers guaranteeing freshness of tag's identification information. In addition, it is assumed that the communication channel between a reader and a back-end server is secure.

Our protocol works with the natural assumption that tag has a hash function, XOR gate, and the capability to keep state during a single session and the cryptographic hash function used in our protocol has the desirable security like preimage resistance, second preimage resistance, and collision avoidance. It is designed in order that tags do not have any real data and all messages would be random, hashed, and encrypted, so confidentiality is guaranteed even though the authentication messages are eavesdropped by adversaries.

In our protocol, the server stores the keys k_1, k_2, and $h(k_1)$ of all the tags. When a tag sends its $h(k_1)$ to the server, it tries to looking for this $h(k_1)$ value in the database. At first sight, storing both k_1 and $h(k_1)$ in the database seems to be wasteful because it is redundant, but in exchange for storing $h(k_1)$ values in the database, the searching for a certain tag takes place extremely quickly. Generally, in real RFID systems there could be big amount of tags which can take part in the authentication process. The bigger the system is, the longer it takes to search for a certain tag, because more database entries need to be investigated. To speed up this process, it is reasonable to store the $h(k_1)$ in the database because in this case we only have to read this value from the database

instead of computing it every time a database entry needs to be investigated. It increases the effectiveness of the protocol.

After receiving $h(k_1)$ from a tag, the server tries to search for this $h(k_1)$ value in the database. Finding the $h(k_1)$ value in the database only means that the message sender might be a tag in the system. The remaining part of the protocol is the mutual authentication between the tags and the server, as well as updating the keys. Notice that after the server has found $h(k_1)$ value in its database, it generates such a random r_1 that $h(k_1 \oplus r_1)$ become unique in the database. This is required for the sake of the latter key update, because when the process of exchanging data has finished, the k_1 key of the tag would be updated to $k_{new} = h(k_1 \oplus r_1)$.

3.2 The proposed protocol

Our proposed protocol can be seen in Fig. 5. In the followings our hash-based authentication protocol is described in detail.

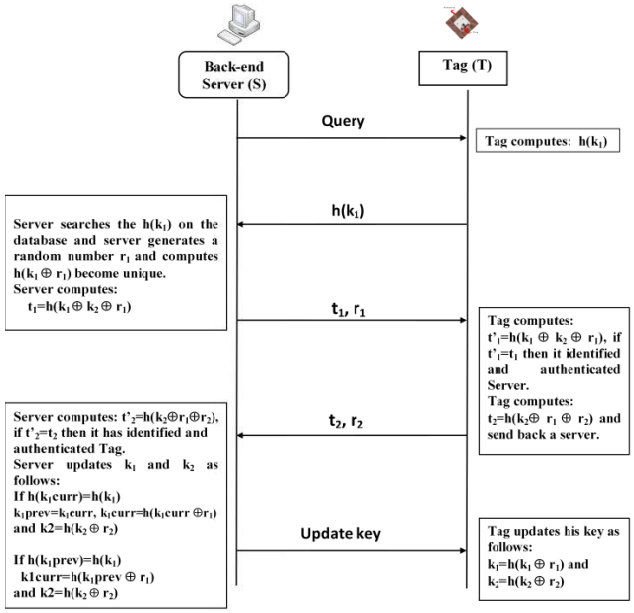

Fig. 5. Proposed Authentication Protocol

1. Back-end server broadcasts Query messages.
2. Receiving the Query message the tag computes $h(k_1)$ and sends it back to the back-end server.
3. The back-end server tries to looking for the received $h(k_1)$ among the stored ones in the database (in that case when $h(k_1)$ is not found in the current

values' table, it should be searched in the previous values' table, too). In case it is found, the back-end server generates such a random number r_1 that $h(k_1 \oplus r_1)$ become unique among other $h(k_1)$ values in the database. The back-end server then computes $t_1 = h(k_1 \oplus k_2 \oplus r_1)$, and sends the r_1, t_1 message to the tag.

4. The tag computes $t_1' = h(k_1 \oplus k_2 \oplus r_1)$, if $t_1' = t_1$ then it has an identified and authenticated server. Tag computes: $t_2 = h(k_2 \oplus r_1 \oplus r_2)$ and sends r_2, t_2 message back to the back-end server.
5. The back-end server computes $t_2' = h(k_2 \oplus r_1 \oplus r_2)$, if $t_2' = t_2$ then it has an identified and authenticated tag. After the successful authentication back-end server updates k_1 and k_2 as follows:
   - If $h(k_{1curr}) = h(k_1)$, in other words the received $h(k_1)$ value corresponds to the stored ones in the current values table of database, then:
     - $k_{1prev} = k_{1curr}$, $k_{1curr} = h(k_{1curr} \oplus r_1)$
     - $k_2 = h(k_2 \oplus r_2)$
   - If $h(k_{1prev}) = h(k_1)$, which means the received $h(k_1)$ value corresponds to the stored ones as in the previous values table of database, then:
     - $k_{1curr} = h(k_{1prev} \oplus r_1)$
     - $k_2 = h(k_2 \oplus r_2)$

   This process seems a bit complicated because our goal is to prevent the loss of data when any kind of attacks happens. It protects the tags even in case of DoS attack in the last step of the protocol, while the server sends the "Update key" message to the tag. In this case the tag does not receive the "Update key" message, thus it does not update its keys, meanwhile the server has updated keys related to the tag, but the old information about the tag is still stored in the database as previous values. So the next time when tag want to authenticate with the server, the server would not be able to find the $h(k_1)$ of the tag among the $h(k_1)$ stored as current value in its database, but it could find it among the values stored as previous ones. Hereby this tag would be legal in the system.
6. After the server has updated the keys, sends "Update key" message to the tag. Tag updates his key as follows:
   - $k_1 = h(k_1 \oplus r_1)$
   - $k_2 = h(k_2 \oplus r_2)$

3.3 Security Analysis

In this section the security analysis of our scheme is presented. We will show that our scheme is protected against the well-known attacks in an RFID environment. Moreover, attacks based on tampering with tags are also limited by updating secure information.

Tag information privacy When the server broadcasts Query messages a given tag sends back only its $h(k_1)$ to the server, then waits for the server to replies the t_1, r_1 message. After it is received by the tag, it could check whether t_1

is the right value, or not. In case that it is suitable the tag sends t_2 which contains k_2 to server. So, the server only has access to the tag information after it authenticated itself. A counterfeit server can only access to $h(k_1)$ values which are freely accessible from anywhere.

Tag location privacy The responses from the tags are anonymous, the tag only sends $h(k_1)$, t_2, r_2, which cannot be linked to any particular tag. In other words, the eavesdropper can neither link tag responses to previous responses from the same tag, nor distinguish one tag's responses from other's. It can be said that tracking the location of a tag is very difficult.

Eavesdropping Eavesdropper can collect information about $h(k_1)$, t_1, r_1, and t_2, r_2 during one authentication session. Since r_1 and r_2 are random values, they are useless to trace tags. After each successful authentication process, tags update their k_1 keys, therefore $h(k_1)$ values can also be considered as random values. The $t_2 = h(k_2 \oplus r_1 \oplus r_2)$ requires random numbers generated by two parties – the server and the tag. An impersonated server or impersonated tag cannot control the random number generated by the other party. Consequently, it is difficult for an eavesdropper to attack with t_2. The $t_1 = h(k_1 \oplus k_2 \oplus r_1)$ value is computed from the two stored key values and a random value generated by server, the probability that the generated values among sessions are identical is negligible. Thus any useful information cannot be gained by eavesdropping.

Untraceability The server authenticates itself by sending the t_1, r_1 message, while a tag authenticates itself by sending the t_2, r_2 message. In each step, tag and server authenticate counterpart to remove traceability. In addition, despite the attacker knows k_1, k_2, a particular tag cannot be traced since tag responses in each session is different.

Cloning The secret information stored in a given tag is pertinent to that tag. Even if some tags are compromised, the obtained information is irrelevant for the others. Therefore, attacker cannot produce any other fake tag except the compromised tags.

Security against the spoofing attack Since each message is encrypted during the authentication process the spoofing attack is prevented. The attacker can only perform the following attack.

1. The attacker impersonate a real server and sends Query to the tag.
2. The attacker can acquire $h(k_1)$ from the tag as a response.

In the subsequent session, when the real server transmits Query, the attacker responds with $h(k_1)$ in order to disguise as a right tag and a server responses with $t_1 = h(k_1 \oplus k_2 \oplus r_1)$, r_1, but the attacker does not know the k_2 and it is

not be able to computed, the $t_2 = h(k_2 \oplus r_1 \oplus r_2)$ for replying to the server. Thus it can authenticate itself as a tag to the server. In our protocol, an attacker can only spoof a tag if the attacker knows both k_1 and k_2, which happens very rarely. Consequently, the proposed protocol is secure against spoofing attack.

Security against the replay attack An attacker can eavesdrop the response message from a tag and may perform an attack as the followings.

1. After the server transmits $t_1 = h(k_1 \oplus k_2 \oplus r_{1old})$, r_{1old} to the tag, the attacker eavesdrops the response of the tag $t_2 = h(k_2 \oplus r_1 \oplus r_2)$, r_2 and the attacker block "Update key" message which is sent by the server, consequently the tag could not update its keys.
2. In the next authentication session, the attacker disguise as a legitimate tag and sends $h(k_1)$ to the server. When the server transmits $t_1' = h(k_{1old} \oplus k_{2old} \oplus r_{1new})$, r_{1new} , the attacker responds with $t_2 = h(k_2 \oplus r_{1old} \oplus r_2)$, r_2. But since r_{1new} is a random number, r_{1new} is different from r_{1old}, the $t_2' = h(k_2 \oplus r_{1new} \oplus r_2)$ value computed by the server wolud be different from t_2 value which is possessed by the attacker, therefore the server could stop this authentication session. So the proposed protocol is secure against replay attack. Furthermore, if the one-way hash function such as MD5, SHA-1 is secure, the attacker does not know the keys of the tag due to the one-way property of hash function. So it is impossible to produce a right response $h(k_2 \oplus r_1 \oplus r_2)$.

Forward and backward security Such an attack might happen that after an attacker obtains k_1, k_2 by tampering with tag and has eavesdropped plenty of interactions between server and various tags, the attacker tries to infer the past or future secret information, but server and tag always update their keys using hash-function each time of the protocol, so it will never succeed.

Denial of Service Attack If the server keeps only the current key per tag, the protocol is under the denial of service attack. The attack is possible as follows:

Attacker may try to do attack in order to desynchronize k_1 and k_2 between server and tag. We assume that attacker can relay the data between these two parties. When server starts the authentication process, attacker transfers t_1, r_1 from server to tag and then transfers t_2, r_2 to server. After that the server updates k_1 and k_2 in its database it sends the "Update key" message in order to update k_1 and k_2 in tag. At this time, if an attacker could block the delivery of "Update key" message, k_1, k_2 values become different between server and tag.

If server uses only the latest k_1, k_2 to authenticate a tag, this situation makes tag useless. However, in our protocol not only the current k_1, k_2 values, but the previous k_1 and k_2 values are stored in the server database. In case a DoS attack happens server can still identify tag in a near future session by checking both the current and previous values in the database.

Scalability In our protocol, the server can find the keys of a tag by searching for the $h(k_1)$ received from the tag. In the database the k_1, $h(k_1)$, and k_2 values of each tag are stored. When a tag sends $h(k_1)$ to the server the server can quickly find $h(k_1)$ in the database, and so the server can find the appropriate k_1, k_2 quickly and efficiently. So the system is scalable.

4 Comparison with other protocols

In Section 2 a brief overview about the security issues of previously proposed protocols were given, additionally in Section 3.3 our protocol's security analysis was presented. In this section the comparison from the point of view of security is given. In Table 1 we summarize the security and efficiency issues about the analyzed protocols.

The notations are the followings:

- The notation O means that the given protocol satisfies a given security requirement.
- The notation $\triangle$ means the given protocol just only partially satisfy a given security requirement.
- The notation X means that the given protocol does not satisfy a given security requirement.

Table 1. *Security comparison with other protocols*

Protocols	HBAC	RAC	Tree-based protocol	Ohkubo protocol	Our protocol
Replay attack	X	X	O	O	O
Prevent Spoofing R	X	X	O	O	O
Prevent Spoofing T	X	X	O	O	O
Forward Secrecy	X	X	X	O	O
Untraceability	X	△	O	O	O
Anti-Cloning	O	O	O	O	O
DoS attack	X	X	X	X	△
Eavesdropping	X	△	O	O	O
De-synchronization	△	△	O	△	O
Scalability	△	△	O	X	O

As it can be seen in Table 1 all schemes satisfy anti-cloning. Since all values emitted by tags are randomized and there is no data which reveals information of an object tag is attached in memory of tag, all schemes including our scheme satisfy data confidentiality. Because attacker cannot generate a legitimate response without any knowledge of secure information of tag, attacker cannot cloning tag without tampering it. HBAC is such a simple protocol that cannot resist any types of attack. In the tree-based protocol, the keys are not updated after each sessions, thus an attacker manages to obtain key of a tag, the attacker would

know the behavior of the tag in the past. The Ohkubo protocol is very strong against tracking attack and replay attack, but the drawback of it is a lack of scalability. All protocols is very weak against desynchronization attack except our proposed protocol, where the previous keys are also stored.

5 Conclusion

In this contribution we pointed out the weaknesses of RFID systems which have to be solved. A brief overview about some well-known authentication protocols were given. It can be said that the major disadvantages are the decrease of anonymity level and the necessity of very high computational capacity.

We proposed our hash-based mutual authentication protocol for low-cost RFID environment. Our solution provides an efficient authentication method for the back-end and tags, furthermore, it resists against common RFID attacks, i.e., replay attack, impersonation, and forward security. Since in our solution every messages are randomized the location privacy is guaranteed, too.

Our future consideration is to implement our authentication protocol in order to make a comparison between our protocol and other authentication protocols by simulations in terms of security and performance.

References

1. Weis, S., Sarma, S., Rivest, R., and Engels, D., Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems, *International Conference on Security in Pervasive Computing – SPC 2003*, pp. 454–469, Boppard, Germany, March 2003.
2. Lee, S., Hwang, Y., Lee, D., and LimWeis, J., Efficient authentication for low-cost RFID systems, *International Conference on Computational Science and its Applications – ICCSA 2005*, pp. 619–627, May 2005.
3. Zhaoyu Liu and Dichao Peng, True random number generator in RFID systems against traceability, *IEEE Consumer Communications and Networking Conference – CCNS 2006*, pp. 620–624, Las Vegas, Nevada, USA, May 2005.
4. Molnar, D., and Wagner, D., Privacy and Security in Library RFID: Issues, Practices, and Architectures, *Conference on Computer and Communications Security – ACM CCS*, pp. 210–219, Washington, DC, USA, October 2004.
5. Ohkubo, M., Suzuki, K., and Kinoshita, S., Cryptographic Approach to "Privacy-Friendly" Tags, *RFID Privacy Workshop*, MIT, MA, USA, November 2003.
6. Gildas Avoine and Philippe Oechslin, A Scalable and Provably Secure Hash based RFID Protocol, *International Workshop on Pervasive Computing and Communication Security*, 2005.
7. Gene Tsudik, YA-TRAP: Yet Another Trivial RFID Authentication Protocol, *International Conference on Pervasive Computing and Communications – PerCom 2006*, Pisa, Italy, March 2006.
8. Henrici, D. and Muller, P., Hash-based Enhancement of Location Privacy for Radio-Frequency Identification Devices using Varying Identifiers, *International Conference on Pervasive Computing and Communications Workshops*, March 2004.