O-FRAP+: Enhancing Security of a Forward secure RFID Authentication Protocol

Dang Nguyen Duc* and Kwangjo Kim*

*Department of Information and Communication Engineering, KAIST

Abstract

In this paper, we point out two security weaknesses of a forward-secure RFID authentication protocol proposed by Tri Van Le *et al* called O-FRAP. The first security weakness of O-FRAP is its vulnerability against denial-of-service attack. The second one is a flaw in a key updating procedure that can break forward security of O-FRAP. We then propose our improved protocol which we call O-FRAP+. Our proposed protocol eliminates the denial-of-service threat and achieves forward security.

*I.*Introduction

Tri Van Le *et al.* proposed a forward secure RFID authentication protocol called O-FRAP [1] which stands for *Optimistic Forward secure RFID Authentication Protocol.* The following security properties of O-FRAP are claimed by the authors:

- Mutual authentication between RFID tag and RFID reader: a RFID tag and a RFID reader is mutually authenticated after a successful execution of O-FRAP.
- Anonymity of RFID tag: RFID tag is secure against tracking by malicious parties.
- Forward security: even if a RFID tag is corrupted, its history remains anonymous.

O-FRAP is proven to be secure in a security model called *Universal Composable Security Framework* (UC framework for short) which was suggested by Ran Canetti [2]. The UC framework not only guarantees the security of a protocol running in isolation, but also the same protocol running as a component of a larger system.

In this paper, we first analyze the security of O-FRAP. Our analysis of O-FRAP shows that it is vulnerable to denial of service attack. Furthermore, we demonstrate that O-FRAP does not strictly achieve forward security. We then propose an improved protocol called O-FRAP+ to cope with our attacks. Comparing to O-FRAP, O-FRAP+ introduces insignificant computational overhead but requires much less memory storage on the server side.

II. O-FRAP

Throughout this paper, we will use notations summarized in Table 1.

Notation	Description
D	Database of tags kept by server
$D.query(.)$	Database querying procedure
$D.update(.)$	Key updating procedure at database
n	Number of tags in D
l	Security parameter, *e.g.*, bit length of secret
k_{tag}	Shared secret between Tag and Server
k_{tag}^{prev}	Shared secret in one of old session
k_{tag}^{cur}	Current shared secret
k_{tag}^{new}	Shared secret for the next session
r_{tag}	Tag pseudonym
$Prev_i$	(k_{tag}, r_{tag}) of tag i stored in D which was used in one of previous sessions
Cur_i	(k_{tag}, r_{tag}) of tag i stored in D which is currently used
$instance(i)$	A pair of the form (k_{tag}, r_{tag}) of tag i
k_S	Fixed key shared between Tag and Server
$F(.)$	Pseudorandom function

Table 1. Notations

O-FRAP [1] is an authentication protocol between a server and many RFID tags. In O-FRAP, each tag is numbered from 1 to n

and all of tag information are stored in a database D of the back-end server. Note that the RFID reader is omitted in the description of O-FRAP as it essentially just plays the role of a proxy relaying messages exchanged between the tag and the server. A detail discription of O-FRAP is given in Fig. 1.

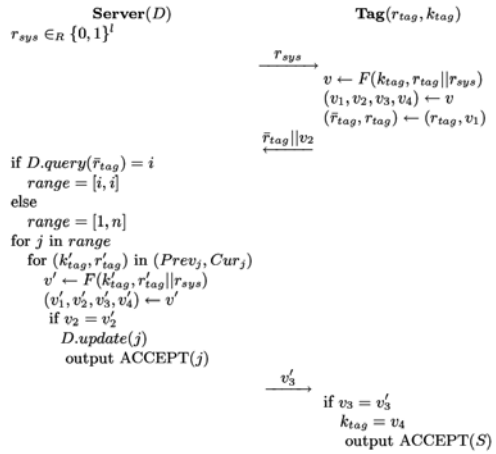

Fig. 1. O-FRAP

We now discuss how O-FRAP achieves anonymity and forward security. Each tag shares a secret key, denoted as k_{tag}, with the server. To protect a tag against tracing attack, for each authentication session, a tag uses a randomly chosen number r_{tag} as its pseudonym. The pseudonym is also stored in the memory of the tag and D simultaneously. The goal is to use the pseudonym to index D and quickly look up information on a tag given its pseudonym (the *D.query*(.) procedure). To achieve forward-security, k_{tag} is updated after every successful authentication session both at the tag and the server sides. To prevent desynchronization attack, the sever keeps two versions of secret key for each tag in its database. a previously-used key (say k^{prev}) and a currently-used one (say k^{cur}). The server updates the two keys in the *D.update*(.) procedure as follows:

- If the tag is authenticated with k^{cur}, the server does: $k^{prev} = k^{cur}$ and $k^{cur} = k^{new}$ where knew is a newly generated key.
- If the tag is authenticated with k^{prev}, the server preserves k^{prev} and lets $k^{cur} = k^{new}$. The server does not update k^{prev} because an active attacker can cause desynchronization of the secret by modifying the authentication token of the server in two consecutive authentication sessions.

The server also maintains two versions of pseudonym for each tag and each entry in D is indexed by two pseudonyms. The two pseudonyms are also updated in the same fashion as the secret keys are.

III. Security Weaknesses of O-FRAP

Denial of service attack on O-FRAP. As we can see, in O-FRAP, the server will scan through the whole tag population in D if it fails to locate one single tag in D given a pseudonym of a tag. An attacker can exploit this property by sending an invalid pseudonym to the server and thus abuse the computational resource of the server. A widespread presence of illegitimate tags can make the problem even more serious.

Forward insecurity of O-FRAP. Assuming that the pseudorandom function F is secure, the forward security of O-FRAP relies entirely on the secret key being updated after every authentication session by both the tag and the server. Consequently, if the server successfully authenticates the tag and updates the shared secret, it is required that the tag will also correctly verify the server and update the secret key. However, this is impossible to achieve as an active attacker can always modify the v'_3

value sent by the server resulting in the tag not to update its secret key. In other words, forward security is lost.

IV. Our Proposed Protocol – O-FRAP+

We now present our improved protocol named O-FRAP+ which aims to solve the security issues with O-FRAP we have showed in Section 2. We now discuss how to address those security issues and then describe the construction of O-FRAP+.

Main Idea. In order to prevent a RFID tag from updating its pseudonym accidentally, the server needs to be authenticated first. This cannot be done in O-FRAP as the server has to look up the secret key before it can compute the authentication token v'_3. However, we note that the privacy of the server (including forward security for the server) is not required. Therefore we can use another fixed key to authenticate the server. This key is common for all tags or group of tags in the database of the server so that the server does not have to look up this key first. Let's call this key k_S. Authenticating the server first has another benefit as the tag can now update its secret key before the server. The desynchronization attack can still be possible but in this case the problem seems to be much easier to handle without the need for storing double keys for each tag. To detect whether a pseudonym has been tampered with before reaching the server, a tag can attach to its pseudonym an integrity-checking message with the secret key k_S. In other words, the tag is authenticated by the server in two steps: first, the server verifies that the tag is in its database with k_S; then the tag is identified with its own secret key k_{tag}. Doing this has some benefits in practice. For example, the tag database may be partitioned and distributed and a unique k_S is assigned for each partition of the database. Then the key k_S can be used to identify which partition of D that the tag belongs to.

Construction. In O-FRAP+, each tag shares with the server two key, a common key k_S and its private secret key k_{tag}. The database D is indexed with only currently-used pseudonym. The protocol consists of 4 rounds roughly described as follows:

- Round 1: The server broadcasts its querying request r_{sys}.
- Round 2: The tag then challenges the server with t_{sys}.
- Round 3: The server communicates its response to be authenticated by the tag.
- Round 4: After authenticating the server, the tag updates its pseudonym and secret key. Then it sends its old pseudonym and its authentication token.

A detail description of O-FRAP+ is depicted in Fig. 2.

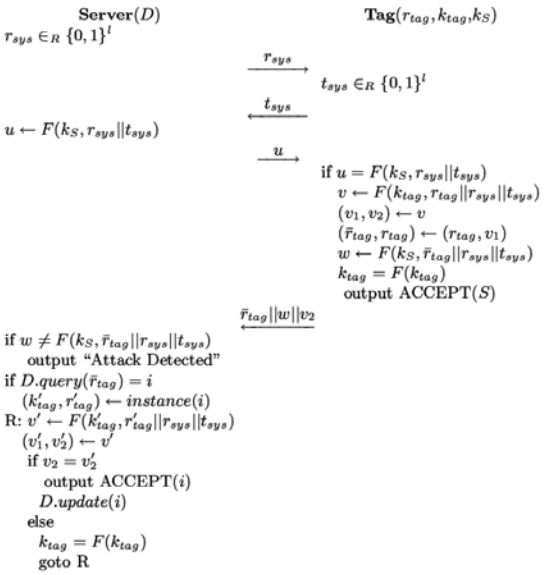

Fig. 2. O-FRAP+

Note that, O-FRAP+ is a 4-round protocol for a purely practical reason. In real life, the server (or the RFID reader) is usually the one to initiate an authentication session. From a security point of view, r_{sys} can be moved to the round 2 without any effect on the security of the protocol.

We now discuss the key updating procedure at the server side. After successfully verify that the tag is in D by using k_S, it is likely the *D.query*(.) will succeed and return one entry in D. Let *instance*(i) be a (Secret Key, Pseudonym) = (k'_{tag}, r'_{tag}) pair of the tag i stored in D. Then, the server can authenticate the tag with the key k'_{tag}. However, it is still possible to cause desynchronization of k_{tag} in O-FRAP+. An active attacker can modify v_2 resulting in the tag authentication with k'_{tag} to fail and the server not to update the shared secret. Note that, the desynchronization problem in O-FRAP+ is very different from the one in O-FRAP. In O-FRAP, the desynchronization of k_{tag} means the tag still keeps a key used in one of previous authentication sessions while the server keeps the currently-used key. Meanwhile, in O-FRAP+, if desynchronization of k_{tag} occurs then the server keeps one of old keys while the tag has the latest key. Furthermore, even though k_{tag} is inconsistent, the server can still single out the candidate tag in its database. It is not the case in O-FRAP. This is why we do not need to store two versions of key for each tag in D. To accomodate a tag whose k_{tag} has been desynchronized, we update k_{tag} in a chaining fashion, *i.e.*, k_{tag} = $F(k_{tag})$ where F is a pseudorandom function. This allows the server to try a new k'_{tag} = $F(k'_{tag})$ to authenticate the tag once it fails to authenticate the tag with its current version of k'_{tag}. The number of times the server tries in this scenario is up to a specific deployment of O-FRAP+.

In [3], the authors mentioned a method to cause desynchronization of secret by completely blocking an important message, *e.g.*, v'_3 in O-FRAP and v_2 in O-FRAP+. We argue that the blocking attack is easier to be dealt with in O-FRAP+ than in O-FRAP. It is because in O-FRAP+, the server, not the tag, is at risk not to update the secret. As the server has a far more computational resource that the tag, it can use some mechanisms like timer to detect blocking.

Comparison. We compare O-FRAP and O-FRAP+ in terms of computational resource and security features in Table 2.

	O-FRAP	O-FRAP$^+$
Number of $F(.)$ Evaluation	Tag: 1 Server: 1	Tag: 4 Server: 4
Key Length	Tag: $2l$ bits Server: $4ln$ bits	Tag: $3l$ bits Server: $(2n+1)l$ bits
Authentication	Mutual	Mutual
Anonimyty	Yes	Yes
Forward Security	Weak	Strong
Resistant against DoS	No	Yes

Table. 2. Comparison of O-FRAP+ and O-FRAP

V.Concluding Remarks

In this paper, we have presented two security weaknesses of a provably secure RFID authentication protocol called O-FRAP. In particular, we pointed out that it is trivial to abuse the server computational resource in O-FRAP. We also showed that O-FRAP is not forward secure due to the way the shared secret key is updated. We also presented our improved protocol of O-FRAP which we call O-FRAP+. The most important point in the design of O-FRAP+ is that the server should be authenticated first. By doing so, we can not only defend O-FRAP+ against denial of service attack but also remove the need for storing two

versions of secret key in the server's database. In addition, O-FRAP+ is now fully forward secure. We conclude that our improved protocol has much better practical value than O-FRAP.

References

[1] Tri Van Le, Mike Burnmester and Breno de Medeiros, ''Universally Composable and Forward Secure RFID Authentication and Authenticated Key Exchange'', In the Proceedings of the 2nd ACM Symposium on Information, Computer and Communications Security, pp. 242--252, March 2007..

[2] Ran Canetti, ''Obtaining Universally Composable Security: Towards the Bare Bones of Trust'', Available at http://eprint.iacr.org/2007/475.

[3] T. van Deursen and S. Radomirovic, ''Attacks on RFID Protocols'', Available at http://eprint.iacr.org/2008/310.

[4] Ari Juels, ''RFID Security and Privacy: A Research Survey'', In the Journal of Selected Areas in Communication (J-SAC), 24(2):381-395, February 2006.

[5] Ari Juels, ''RFID Security and Privacy: A Research Survey'', In the Journal of Selected Areas in Communication (J-SAC), 24(2):381-395, February 2006.

[6] Stephen Weis, ''Security and Privacy in Radio Frequency Identification Devices'', Master Thesis, Available at http://theory.lcs.mit.edu/~sweis/masters.pdf, May 2003.

[7] Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita, ''Efficient Hash-Chain Based RFID Privacy Protection Scheme'', In the Proceedings of International Conference on Ubiquitous Computing, Workshop Privacy, September 2004.

[8] Ari Juels and Stephen Weis, ''Authenticating Pervasive Devices with Human Protocols'', In the Proceedings of CRYPTO'05, Victor Shoup (Eds.), Springer-Verlag, LNCS 3261, pp. 293 - -308, 2005.

[9] Ari Juels, ''Strenthening EPC Tag against Cloning'', In the Proceedings of ACM Workshop on Wireless Security (WiSe), M. Jakobsson and R. Poovendran (Ed.), pp.67-76. 2005.