Efficient RFID authentication protocols based on pseudorandom sequence generators

Jooyoung Lee and Yongjin Yeom

The Attached Institute of Electronics and Telecommunications Research Institute
Yuseong-gu, Daejeon, Korea 305-390
jlee05@ensec.re.kr

Abstract. In this paper, we introduce a new class of PRSGs, called *partitioned pseudorandom sequence generators*(PPRSGs), and propose an RFID authentication protocol using a PPRSG, called *S-protocol*. Since most existing stream ciphers can be regarded as secure PPRSGs, and stream ciphers outperform other types of symmetric key primitives such as block ciphers and hash functions in terms of power, performance and gate size, S-protocol is expected to be suitable for use in highly constrained environments such as RFID systems. We present a formal proof that guarantees resistance of S-protocol to desynchronization and tag-impersonation attacks. Specifically, we reduce the availability of S-protocol to pseudorandomness of the underlying PPRSG, and the security of the protocol to the availability. Finally, we give a modification of S-protocol, called S^*-protocol, that provides mutual authentication of tag and reader.

Keywords: authentication protocol, pseudorandom sequence generator, stream cipher, RFID

1 Introduction

Low-cost RFID tags are rapidly becoming pervasive in our daily life. Well known applications include electronic passports, contactless payments, product tracking and building access control, to name a few. However, the small programmable chips that passively respond to every reader have raised concerns among researchers about privacy and security breaches. A considerable body of research has been focused on providing RFID tags with cryptographic functionality, while scarce computational and storage capabilities of low-cost RFID tags make the problem challenging. Typically, RFID tags can only store hundreds of bits and have 1000-10000 logic gates, with 200-2000 budgeted specifically for security. In such an environment, cryptographic primitives should be implemented with low clock frequency since a tag derives its power from a reader in a short period of time.

In this paper, we focus on the issue of authentication. Our work begins with the observation that stream ciphers are more efficient cryptographic primitives than block ciphers and hash functions. In general, stream ciphers require the fewest computational resources amongst traditional cryptographic primitives, including block ciphers and hash functions, in terms of power, performance, and gate size [9]. One approach to the construction of an authentication protocol based on a stream cipher is to transform a stream cipher into a pseudorandom function via a well-known construction of Goldreich et. al. [8], and use existing protocols based on pseudorandom functions. In this way, we can obtain various authentication protocols that guarantee both security and unlinkability. However, the construction based on the GGM construction might not be suitable for RFID systems due to its inefficiency.

In this paper, we consider a direct construction of an authentication protocol from a stream cipher. In order to capture the properties of existing stream ciphers useful in the construction of authentication protocols, we introduce a new class of pseudorandom sequence generators, called *partitioned pseudorandom sequence generators*(PPRSGs). Informally speaking, a PPRSG is a pseudorandom sequence generator that consists of two functions, respectively called an updating function and a filtering function. Most existing stream ciphers can be regarded as PPRSGs as seen in the next section. Using a secure PPRSG, we construct an authentication protocol, called S-protocol, that provides resistance to desynchronization and tag-impersonation attacks. We also present a modification of S-protocol, called S^*-protocol, that provides mutual authentication of tag and reader.

Contribution. The advantages of S-protocol are summarized as follows.

- S-protocol is mainly targeted at the use of stream ciphers. Based on a secure stream cipher, S-protocol outperforms most existing authentication protocols using hash functions or block ciphers, in terms of power, performance, and gate size. Furthermore, S-protocol does not require key initialization process of the underlying stream cipher, since each tag's secret information can be initialized as the internal state of the stream cipher, obtained after key initialization process with a random secret key.
- Using a same stream cipher, we can construct S-protocols of various security levels against online attacks. If a stream cipher outputs m' bits in one clock, then we can define PPRSGs associated with the stream cipher in a flexible way, so that the filtering function outputs $m = lm'$ bits for any positive integer l. As seen later, the parameter m determines the security level of the S-protocol against online attacks. On the other hand, block ciphers or hash functions always output fixed-size blocks. The block size might be unnecessarily large as compared to the security requirement of the target protocol. This would result in computational overload on the tag-side.
- Availability and security of S-protocol is established by a formal proof. In the proof, we assume a prevention-based model where the adversary has unfettered access to oracles for a tag and a reader. The proof is not based on the use of truly random numbers on the tag-side. Thus S-protocol does

not require any physical random number generator or that an independent PRSG be equipped with a tag devcie.

We also point out some drawbacks of S-protocol.

– S-protocol does not provide untraceable identification. General-purpose authentication protocols may or may not have support for untraceability, while untraceability is considered as a core requirement of authentication protocols for certain RFID applications.
– Since each tag's secret information is updated every session, the back-end databases of the system, if multiple, should be connected in real-time so as to maintain synchronized with the tags.

Related work. The majority of authentication protocols for RFID are still based on hash functions or block ciphers [2, 5, 6, 12, 17, 19–21]. Those works are mainly focused on providing untraceable authentication protocols for RFID. Lightweight implementation of existing block ciphers as well as new constructions are widely studied [7, 13, 18]. As another direction, there have been a number of protocols proposed based on new cryptographic primitives, among which a series of HB-protocols are drawing a lot of attention due to their efficiency and provable security [10, 14, 15]. In [16], the authors proposed an RFID protocol based on a PRSG. We note that, if a stream cipher is used as a PRSG for their protocol, an independent random number is required at each generation of a message. However, a physical random number generator equipped in RFID tags might not guarantee a sufficient level of randomness to support the security of the protocol.

Organization. In the next section, we define a partitioned pseudorandom sequence generator, and present some examples of PPRSGs. In Section 3, we describe S-protocol in two steps. First, we define a tag and a reader as message-driven deterministic algorithms. Based on the algorithms, we illustrate how they exchange messages in each session. In Section 4, we prove the availability and the security of S-protocol. In Section 5, we slightly modify S-protocol to present S^*-protocol that provides mutual authentication of tag and reader. Section 6 concludes.

2 Partitioned pseudorandom sequence generator

In this section, we define a new class of pseudorandom sequence generators called *partitioned pseudorandom sequence generators*. We first begin with the definition of a pseudorandom sequence generator (in a concrete model).

Definition 2.1. *Let d and L be positive integers such that $d < L$. A function $G : \{0,1\}^d \rightarrow \{0,1\}^L$ is called a (t, ϵ)-*pseudorandom sequence generator *(PRSG) if for every probabilistic Turing machine D of runtime $\leq t$ we have*

$$|\,\mathsf{Pr}_{R \overset{\$}{\leftarrow} \{0,1\}^L}[D(R) \Rightarrow 1] - \mathsf{Pr}_{K \overset{\$}{\leftarrow} \{0,1\}^d}[D(G(K)) \Rightarrow 1]| \leq \epsilon.$$

Definition 2.2. *Let d, m and L be positive integers such that $m|L$ and $d < L$. A (t, ϵ, L)*-partitioned pseudorandom sequence generator*(PPRSG) is a pair $\mathcal{S} = (\mathsf{Up}, \mathsf{F})$ of functions $\mathsf{Up} : \{0,1\}^d \to \{0,1\}^d$ and $\mathsf{F} : \{0,1\}^d \to \{0,1\}^m$ such that*

$$\begin{aligned} G_{\mathcal{S}} : \{0,1\}^d &\longrightarrow \{0,1\}^L \\ K &\longmapsto \Big(\mathsf{F}(K) || \mathsf{F} \circ \mathsf{Up}(K) || \dots || \mathsf{F} \circ \mathsf{Up}^{\left(\frac{L}{m}-1\right)}(K)\Big) \end{aligned} \tag{1}$$

is a (t, ϵ)-pseudorandom sequence generator. Functions Up and F are called an update function and a filtering function, respectively.

Example 2.1. Let $h : \{0,1\}^* \to \{0,1\}^m$ be a random oracle. Then $\mathcal{S} = (\mathsf{Up}, \mathsf{F})$ such that

$$\begin{aligned} \mathsf{Up} : \{0,1\}^d &\longrightarrow \{0,1\}^d \\ K &\longmapsto (K + 1 \mod 2^d), \end{aligned}$$

and

$$\begin{aligned} \mathsf{F} : \{0,1\}^d &\longrightarrow \{0,1\}^m \\ K &\longmapsto h(K), \end{aligned}$$

is a PPRSG, since $\mathcal{S}$ generates a truly random sequence.

Example 2.2. Let $E : \{0,1\}^k \times \{0,1\}^m \to \{0,1\}^m$ be a ideal block cipher. Then $\mathcal{S} = (\mathsf{Up}, \mathsf{F})$ such that

$$\begin{aligned} \mathsf{Up} : \{0,1\}^k \times \{0,1\}^m &\longrightarrow \{0,1\}^k \times \{0,1\}^m \\ (K, M) &\longmapsto (K, M + 1 \mod 2^m), \end{aligned}$$

and

$$\begin{aligned} \mathsf{F} : \{0,1\}^k \times \{0,1\}^m &\longrightarrow \{0,1\}^m \\ (K, M) &\longmapsto E(K, M), \end{aligned}$$

is a PPRSG if $L \ll m2^{m/2}$(the birthday bound). Output feedback mode of a block cipher is also a PPRSG.

Example 2.3. Let $f : \{0,1\}^d \to \{0,1\}^d \times \{0,1\}$ be a secure PRSG, denoted $f(K) = (f_1(K), f_2(K))$ for $K \in \{0,1\}^d$. Then the stretching theorem implies that $\mathcal{S} = (\mathsf{Up}, \mathsf{F})$ such that

$$\begin{aligned} \mathsf{Up} : \{0,1\}^d &\longrightarrow \{0,1\}^d \\ K &\longmapsto f_1(K), \end{aligned}$$

and

$$\begin{aligned} \mathsf{F} : \{0,1\}^d &\longrightarrow \{0,1\} \\ K &\longmapsto f_2(K), \end{aligned}$$

is a PPRSG if L is polynomial in d.

Example 2.4. Stream ciphers F-FCSR-H [1], Mickey [3], Trivium [4] and Grain [11], finalized as the eSTREAM portfolio, can be regarded as PPRSGs, if an update function and a filtering function are appropriately defined. For example, Grain-1 consists of a function that updates 160 bits of an internal state and a filtering function that xors certain bits selected from the state. Here we assume that a key initialization process fills the internal state of a stream cipher with uniform random bits.

3 Description of S-protocol

3.1 Main idea

Let $\mathcal{S} = (\mathsf{Up}, \mathsf{F})$ be a PPRSG and let a tag $\mathcal{T}$ and a reader $\mathcal{R}$ store variables $S_{\mathcal{T}}$ and $S_{\mathcal{R}}$, respectively, both initialized as a secret key $K \in \{0,1\}^d$. We can consider a naive approach to the construction of an authentication protocol as follows.

1. $\mathcal{R}$ sends "*query*" to $\mathcal{T}$.
2. $\mathcal{T}$ sends $M_1 \leftarrow \mathsf{F}(S_{\mathcal{T}})$ to $\mathcal{R}$.
3. If $M_1 = \mathsf{F}(S_{\mathcal{R}})$, then $\mathcal{R}$ sends $M_2 \leftarrow \mathsf{F}(\mathsf{Up}(S_{\mathcal{R}}))$ to $\mathcal{T}$.
4. If $M_2 = \mathsf{F}(\mathsf{Up}(S_{\mathcal{T}}))$, then $\mathcal{T}$ updates $S_{\mathcal{T}} \leftarrow \mathsf{Up}^2(S_{\mathcal{T}})$ and sends $M_3 \leftarrow \mathsf{F}(S_{\mathcal{T}})$ to $\mathcal{R}$.
5. If $M_3 = \mathsf{F}(\mathsf{Up}^2(S_{\mathcal{R}}))$, then $\mathcal{R}$ updates $S_{\mathcal{R}} \leftarrow \mathsf{Up}^2(S_{\mathcal{R}})$ and accepts $\mathcal{T}$.

In the above protocol, $\mathcal{T}$ and $\mathcal{R}$ alternately transmit $\mathsf{F}(\mathsf{Up}^i(K))$, $i = 0, 1, 2 \ldots,$ over sessions, and ends up with a state of $S_{\mathcal{T}} = S_{\mathcal{R}}$ for a successful session. Note that $\mathcal{T}$ and $\mathcal{R}$ should exchange at least four messages in a session, in order to prevent trivial replay attacks. However, the above protocol is still vulnerable to a replay attack. An adversary might block message M_3 transmitted from $\mathcal{T}$ to $\mathcal{R}$, and use messages M_1 and M_3 to impersonate $\mathcal{T}$. If the adversary does not mount an impersonation attack on $\mathcal{R}$, then the blocking of message M_3 would lead to desynchronization of $\mathcal{T}$ and $\mathcal{R}$. Therefore, we introduce two additional flags *rev* and *add* on the reader-side to indicate the possibility of replay and desynchronization attacks and control communications. $\mathcal{R}$ sets flag *rev* to one once $\mathcal{R}$ has transmitted $\mathsf{F}(\mathsf{Up}(S_{\mathcal{R}}))$ for the current value of $S_{\mathcal{R}}$. If $\mathcal{R}$ fails to authenticate $\mathcal{T}$ and a new session is initiated, then $\mathcal{R}$ sets flag *add* to one and makes one more round in the session, in order to prevent replay attacks. Now we are ready to present a formal description of S-protocol.

3.2 Formal description

For simplicity, we treat a reader and a back-end database as a single entity. Thus we consider an RFID system that consists of one reader $\mathcal{R}$ and a multiple number of tags, say $\mathcal{T}_i$, $i = 1, \ldots, n$. We model tag and reader functionalities as deterministic Turing machines that store and update variables as follows.

– Each tag $\mathcal{T}_i$, $i = 1, \ldots, n$, stores a d-bit variable $S_{\mathcal{T}_i}$, which is initialized with a random secret key $K_i \in \{0,1\}^d$.
– A reader $\mathcal{R}$ stores a variable $S_{\mathcal{R},i}$ and two auxiliary flags rev_i and add_i for each tag $\mathcal{T}_i$. $S_{\mathcal{R},i}$ is initialized with the same key K_i as tag $\mathcal{T}_i$, while rev_i and add_i are both initialized with "0" for every $i = 1, \ldots, n$.

Focusing on a 1-1 communication between R and $\mathcal{T}_i$, we simplify the notations as $\mathcal{T} = \mathcal{T}_i$, $S_{\mathcal{R}} = S_{\mathcal{R},i}$, $S_{\mathcal{T}} = S_{\mathcal{T}_i}$, $rev = rev_i$ and $add = add_i$. We now present a specific description of tag and reader algorithms in Figure 1. The "*init*" message can be regarded as an external signal or ID claim from a tag that initiates a new session within a reader $\mathcal{R}$. "$rev = 1$" indicates that the reader $\mathcal{R}$ has revealed $F(S_{\mathcal{R}})$ for the current value of $S_{\mathcal{R}}$. "$add = 1$" requires that the reader $\mathcal{R}$ authenticate the tag with an additional round of message exchange. We assume that the special types of messages "*init*", "*query*" and "*accept*" are distinguished from any m bit binary sequence. We call the set of variables $\mathbf{S} = (S_{\mathcal{T}}, S_{\mathcal{R}}, rev, add)$ the *state* of $\mathcal{T}$ and $\mathcal{R}$, and define synchronization states as follows.

Definition 3.1. *Tag $\mathcal{T}$ and reader $\mathcal{R}$ are said to be in a* synchronization state *if their state satisfies $\mathsf{F}(S_{\mathcal{R}}) \neq \mathsf{F}(\mathsf{Up}^2(S_{\mathcal{R}}))$, $\mathsf{F}(\mathsf{Up}^2(S_{\mathcal{R}})) \neq \mathsf{F}(\mathsf{Up}^4(S_{\mathcal{R}}))$, and one of the following three conditions:*

- *Type 1: $S_{\mathcal{T}} = S_{\mathcal{R}}$ and $rev = 0$*
- *Type 2: $S_{\mathcal{T}} = S_{\mathcal{R}}$ and $rev = 1$*
- *Type 3: $S_{\mathcal{T}} = \mathsf{Up}^2(S_{\mathcal{R}})$ and $rev = 1$*

In the next section, we prove that a tag and a reader remain in a synchronization state except with a negligible probability even though a certain number of oracle queries are made to the tag and the reader. This property guarantees the completeness of S-protocol, together with the following theorem.

Theorem 3.1. *Suppose that tag $\mathcal{T}$ and reader $\mathcal{R}$ are in a synchronization state and $\mathcal{R}$ opens a new session on the message "init". Then the session is completed with the signal "accept" within 4 or 6 passes as seen in Figure 2. At the end of the session, $\mathcal{R}$ and $\mathcal{T}$ reach a synchronization state of type 1.*

Proof. First, suppose that $\mathcal{T}$ and $\mathcal{R}$ are in a synchronization state of type 1 and a new session is initiated with the message "*init*". $\mathcal{R}$ sends a message "*query*" to $\mathcal{T}$. On the message "*query*", $\mathcal{T}$ responds with $M_1 \leftarrow F(S_{\mathcal{T}})$. Since $M_1 = F(S_{\mathcal{R}})$, $\mathcal{R}$ sets $rev \leftarrow 1$ and sends $M_2 \leftarrow F(\mathsf{Up}(S_{\mathcal{R}}))$ to $\mathcal{T}$, where the state transits into a synchronization state of type 2. Since $M_2 = F(\mathsf{Up}(S_{\mathcal{T}}))$, $\mathcal{T}$ updates $S_{\mathcal{T}} \leftarrow \mathsf{Up}^2(S_{\mathcal{T}})$ and sends $M_3 \leftarrow F(S_{\mathcal{T}})$ to $\mathcal{R}$, where the state transits into a synchronization state of type 3. Finally, $\mathcal{R}$ checks that $M_3 = F(\mathsf{Up}^2(S_{\mathcal{R}}))$, updates $S_{\mathcal{R}} \leftarrow \mathsf{Up}^2(S_{\mathcal{R}})$ and $rev \leftarrow 0$, and accepts the session. Note that $add = 0$ whenever $rev = 0$. Now the state of $\mathcal{T}$ and $\mathcal{R}$ transits into a synchronization state of type 1.

Next, suppose that $\mathcal{T}$ and $\mathcal{R}$ are in a synchronization state of type 3 and a new session is initiated with the message "*init*". Since $rev = 1$, $\mathcal{R}$ sets $add \leftarrow 1$ and sends a message "*query*" to $\mathcal{T}$. On the message "*query*", $\mathcal{T}$ responds with $M_1 \leftarrow F(S_{\mathcal{T}})$. Since $M_1 = F(\mathsf{Up}^2(S_{\mathcal{R}}))$ and $add = 1$, $\mathcal{R}$ updates $S_{\mathcal{R}} \leftarrow \mathsf{Up}^2(S_{\mathcal{R}})$, $add \leftarrow 0$ and $rev \leftarrow 1$. $\mathcal{R}$ also sends $M_2 \leftarrow F(\mathsf{Up}(S_{\mathcal{R}}))$ to $\mathcal{T}$, where the state transits into a synchronization state of type 2. Since $M_2 = F(\mathsf{Up}(S_{\mathcal{T}}))$, $\mathcal{T}$ updates $S_{\mathcal{T}} \leftarrow \mathsf{Up}^2(S_{\mathcal{T}})$ and sends $M_3 \leftarrow F(S_{\mathcal{T}})$ to $\mathcal{R}$, where the state transits into a synchronization state of type 3. Finally, $\mathcal{R}$ checks that $M_3 = F(\mathsf{Up}^2(S_{\mathcal{R}}))$, updates $S_{\mathcal{R}} \leftarrow \mathsf{Up}^2(S_{\mathcal{R}})$ and $rev \leftarrow 0$, and accepts the session. The state of $\mathcal{T}$ and $\mathcal{R}$ transits into a synchronization state of type 1. The proof is similar for the synchronization state of type 2.

```
1: if $A =$ "query" then
2:     output $\mathsf{F}(S_{\mathcal{T}})$
3: else if $A = \mathsf{F}(\mathsf{Up}(S_{\mathcal{T}}))$ then
4:     $S_{\mathcal{T}} \leftarrow \mathsf{Up}^2(S_{\mathcal{T}})$
5:     output $\mathsf{F}(S_{\mathcal{T}})$
```

(a) $\mathcal{T}$ on an incoming message A

```
1: if $B =$ "init" then
2:     if $rev = 1$ then $add \longleftarrow 1$
3:     output "query"
4: else if $B = \mathsf{F}(S_{\mathcal{R}})$ then
5:     $rev \longleftarrow 1$
6:     output $\mathsf{F}(\mathsf{Up}(S_{\mathcal{R}}))$
7: else if $B = \mathsf{F}(\mathsf{Up}^2(S_{\mathcal{R}}))$ then
8:     $S_{\mathcal{R}} \longleftarrow \mathsf{Up}^2(S_{\mathcal{R}})$
9:     $rev \longleftarrow 0$
10:    if $add = 0$ then
11:        output "accept"
12:    else
13:        $add \longleftarrow 0$
14:        $rev \longleftarrow 1$
15:        output $\mathsf{F}(\mathsf{Up}(S_{\mathcal{R}}))$
```

(b) $\mathcal{R}$ on an incoming message B

Fig. 1. Tag algorithm $\mathcal{T}$ and reader algorithm $\mathcal{R}$.

4 Availability and security of *S*-protocol

4.1 Security against a desynchronization attack

We model a desynchronization-adversary $\mathcal{A} = \mathcal{A}(q, t)$ as a probabilistic Turing machine of run time t that makes total q queries to $\mathcal{T}$ and $\mathcal{R}$. The goal of desynchronization-adversary is to make the reader and tag's state satisfy one of the following three conditions.

- Type 1: $S_{\mathcal{R}} = \mathsf{Up}^2(S_{\mathcal{T}})$
- Type 2: $S_{\mathcal{T}} = \mathsf{Up}^4(S_{\mathcal{R}})$ or $(S_{\mathcal{T}} = \mathsf{Up}^2(S_{\mathcal{R}})$ and $rev = 0)$
- Type 3: $\mathsf{F}(S_{\mathcal{R}}) = \mathsf{F}(\mathsf{Up}^2(S_{\mathcal{R}}))$ or $\mathsf{F}(\mathsf{Up}^2(S_{\mathcal{R}})) = \mathsf{F}(\mathsf{Up}^4(S_{\mathcal{R}}))$

We call the above states *desynchronization states*. Informally speaking, the desynchronization states are the three possible states that may occur when the previous state is a synchronization state, but the current state is not. For simplicity of analysis, we assume that the reader $\mathcal{R}$ reveals at each query whether or not the variable $S_{\mathcal{R}}$ and rev are updated. It allows us to assume that $\mathcal{A}$ stops updating the tag and reader's state once the state comes into a desynchronization state of type 1 or 2. In this way, the desynchronization states cover all the situations such that $\mathcal{T}$ and $\mathcal{R}$ are not in a synchronization state (i.e., we do not need to consider the case $S_{\mathcal{R}} = \mathsf{Up}^i(S_{\mathcal{T}})$ for $i > 2$ or $S_{\mathcal{T}} = \mathsf{Up}^i(S_{\mathcal{R}})$ for $i > 4$ as desynchronization states). When $\mathcal{A}^{\mathcal{T},\mathcal{R}}$ ends up with a desynchronication state, we say that $\mathcal{A}$ *succeeds in a desynchronization attack against* $\mathcal{T}$ *and* $\mathcal{R}$.

Now we would like to use a desynchronization-adversary $\mathcal{A}$ with a success probability

$$\delta = \mathsf{Pr}_{K \xleftarrow{\$} \{0,1\}^d}[\mathcal{A} \text{ succeeds in a desynchronization attack against } \mathcal{T} \text{ and } \mathcal{R}], \tag{2}$$

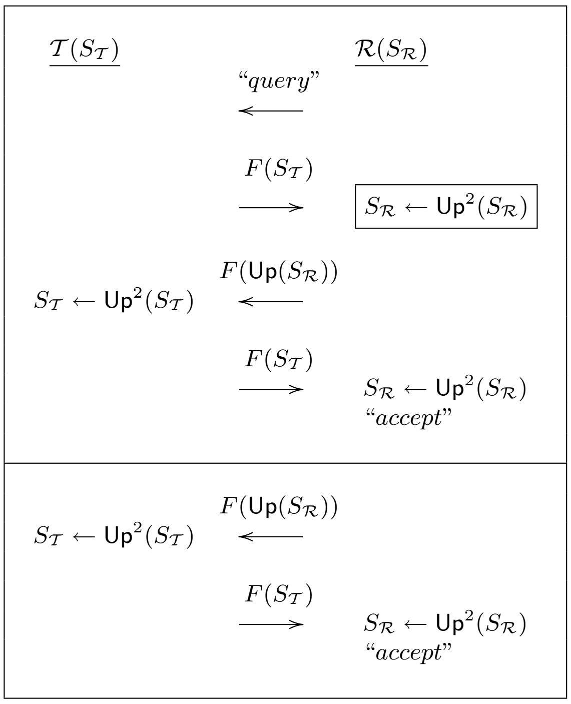
$\underline{\mathcal{T}(S_\mathcal{T})}$
$\underline{\mathcal{R}(S_\mathcal{R})}$
"*query*"
$F(S_\mathcal{T})$
$\boxed{S_\mathcal{R} \leftarrow \mathsf{Up}^2(S_\mathcal{R})}$
$F(\mathsf{Up}(S_\mathcal{R}))$
$S_\mathcal{T} \leftarrow \mathsf{Up}^2(S_\mathcal{T})$
$F(S_\mathcal{T})$
$S_\mathcal{R} \leftarrow \mathsf{Up}^2(S_\mathcal{R})$
"*accept*"
$F(\mathsf{Up}(S_\mathcal{R}))$
$S_\mathcal{T} \leftarrow \mathsf{Up}^2(S_\mathcal{T})$
$F(S_\mathcal{T})$
$S_\mathcal{R} \leftarrow \mathsf{Up}^2(S_\mathcal{R})$
"*accept*"

Fig. 2. Message flow of S-protocol. The boxed statement is executed only if $\mathcal{T}$ and $\mathcal{R}$ are in a synchronization state of type 3. The last two flows are executed only if $\mathcal{T}$ and $\mathcal{R}$ are in a synchronization state of type 2, in which case the reader does not send "accept" message as a response of the fourth flow.

for the construction of a distinguisher that determines whether a given $(2q+3)m$ bit sequence $\mathcal{M} = (M_0, \ldots, M_{2q+2})$ is generated by a PPRSG $\mathcal{S}$ or truly at random. We describe the distinguisher D using a game $\mathcal{G}(\mathcal{M})$ defined in Figure 3. The game $\mathcal{G}(\mathcal{M})$ is parameterized by the sequence $\mathcal{M}$ and includes procedures that simulate tag and reader interfaces up to q queries by referring to $\mathcal{M}$. The variables c_T and c_R of the game $\mathcal{G}(\mathcal{M})$ record, respectively, the numbers of updates of tag-side and reader-side internal state variables. (i.e., c_T and c_R, respectively, correspond to $S_\mathcal{T}$ and $S_\mathcal{R}$.) In procedure Finalize, $\mathcal{G}(\mathcal{M})$ returns the value 1 if the current state, represented by c_T and c_R, is in a desynchronization state.

The distinguisher D consists of $\mathcal{A}$ and $\mathcal{G}(\cdot)$ as illustrated in Figure 4. On the input sequence $\mathcal{M}$, D runs $\mathcal{A}$ and responds to $\mathcal{A}$'s queries by using procedures $\mathcal{T}$ and $\mathcal{R}$ of the game $\mathcal{G}(\mathcal{M})$. At the end of execution, D outputs the value returned in procedure Finalize of $\mathcal{G}(\mathcal{M})$. From the construction, the following estimate is obvious.

$$\mathsf{Pr}_{K \overset{\$}{\leftarrow} \{0,1\}^d}[\mathcal{G}(G_\mathcal{S}(K))^\mathcal{A} \Rightarrow 1] = \mathsf{Pr}_{K \overset{\$}{\leftarrow} \{0,1\}^d}[D(G_\mathcal{S}(K)) \Rightarrow 1] = \delta. \qquad (3)$$

Game $\mathcal{G}(\mathcal{M})$

```
procedure Initialize
1: $c_T, c_R, rev, add, X \leftarrow 0$

procedure $\mathcal{T}(A)$
1: if $A =$ "query" then
2:     output $M_{c_T}$
3: else if $A = M_{c_T+1}$ then
4:     $c_T \leftarrow c_T + 2$
5:     output $M_{c_T}$

procedure Finalize
1: if $M_{c_R} = M_{c_R+2}$ or $M_{c_R+2} = M_{c_R+4}$ then
2:     $X \longleftarrow 1$
3: else if $c_R = c_T + 2$ then
4:     $X \longleftarrow 1$
5: else if $c_T = c_R + 4$ then
6:     $X \longleftarrow 1$
7: else if $c_T = c_R + 2$ and $rev = 0$ then
8:     $X \longleftarrow 1$
9: return $X$

procedure $\mathcal{R}(B)$
1: if $B =$ "init" then
2:     if $rev = 1$ then $add \longleftarrow 1$
3:     output "query"
4: else if $B = M_{c_R}$ then
5:     $rev \longleftarrow 1$
6:     output $M_{c_R+1}$
7: else if $B = M_{c_R+2}$ then
8:     $c_R \longleftarrow c_R + 2$
9:     $rev \longleftarrow 0$
10:    if $add = 0$ then
11:        output "accept"
12:    else
13:        $add \longleftarrow 0$
14:        $rev \longleftarrow 1$
15:        output $M_{c_R+1}$
```

Fig. 3. Parameterized game $\mathcal{G}(\mathcal{M})$.

On the other hand, let us assume that $\mathcal{M}$ is a truly random L bit sequence for $L = (2q+3)m$. Let

$$P_1 = \mathsf{Pr}_{\mathcal{M} \overset{\$}{\leftarrow} \{0,1\}^L}[\mathcal{G}(\mathcal{M})^{\mathcal{A}} \text{ sets } c_R = c_T + 2], \tag{4}$$

$$P_2 = \mathsf{Pr}_{\mathcal{M} \overset{\$}{\leftarrow} \{0,1\}^L}[\mathcal{G}(\mathcal{M})^{\mathcal{A}} \text{ sets } (c_T = c_R + 4) \text{ or } (c_T = c_R + 2 \wedge rev = 0)], \tag{5}$$

and

$$P_3 = \mathsf{Pr}_{\mathcal{M} \overset{\$}{\leftarrow} \{0,1\}^L}[\mathcal{G}(\mathcal{M})^{\mathcal{A}} \text{ sets } M_{c_R} = M_{c_R+2} \text{ or } M_{c_R+2} = M_{c_R+4}]. \tag{6}$$

Then we have

$$\mathsf{Pr}_{\mathcal{M} \overset{\$}{\leftarrow} \{0,1\}^L}[D(\mathcal{M}) \Rightarrow 1] = \mathsf{Pr}_{\mathcal{M} \overset{\$}{\leftarrow} \{0,1\}^L}[\mathcal{G}(\mathcal{M})^{\mathcal{A}} \Rightarrow 1] \leq P_1 + P_2 + P_3. \tag{7}$$

Now the following lemma provides the estimation of P_1, P_2 and P_3.

Lemma 4.1. *Let P_1, P_2 and P_3 be the probabilities, respectively defined by* (4), (5) *and* (6). *Then we have $P_1 \leq q/2^m$, $P_2 \leq q/2^m$ and $P_3 \leq 2q/2^m$.*

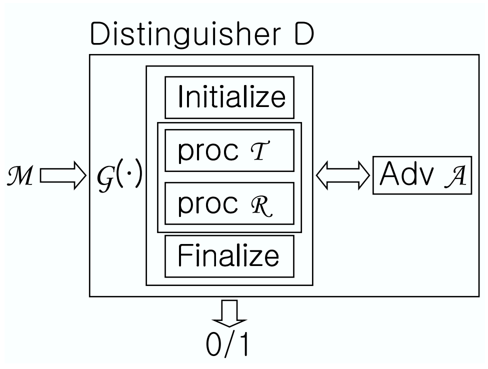

Fig. 4. Distinguisher with game $\mathcal{G}(\cdot)$.

Proof. As for the probability P_3, we have

$$P_3 = \mathsf{Pr}_{\mathcal{M}\xleftarrow{\$}\{0,1\}^L}[\mathcal{G}(\mathcal{M})^{\mathcal{A}} \text{ sets } M_{c_R} = M_{c_R+2} \text{ or } M_{c_R+2} = M_{c_R+4}]$$
$$\leq \sum_{i=0}^{q-1}\Big(\mathsf{Pr}_{\mathcal{M}\xleftarrow{\$}\{0,1\}^L}[M_{2i} = M_{2i+2}] + \mathsf{Pr}_{\mathcal{M}\xleftarrow{\$}\{0,1\}^L}[M_{2i+2} = M_{2i+4}]\Big) = \frac{2q}{2^m}.$$

In order to estimate the probability P_1, we modify the game $\mathcal{G}(\mathcal{M})$ to define $\mathcal{G}_2$ as shown in Figure 5. In $\mathcal{G}_2$, the parameter $\mathcal{M}$ is randomized in procedure Initialize. Each index $i \in I$ is associated with a set $\mathbf{Z}_i$ that is used to record every attempt of setting the counters c_T and c_R to the desynchronization state of $c_R = c_T + 2 = i$. Note that procedures $\mathcal{T}(\cdot)$ and $\mathcal{R}(\cdot)$ of $\mathcal{G}_2$ compute exactly the same responses as those of game $\mathcal{G}(\mathcal{M})$ for a random sequence $\mathcal{M} \xleftarrow{\$} \{0,1\}^L$, with the only exception being that procedure of $\mathcal{R}(\cdot)$ of $\mathcal{G}_2$ stores any message it receives in the set $\mathbf{Z}_{c_R}$ (without any response) if the condition $c_T = c_R$ holds. Suppose that a desynchronization-adversary $\mathcal{A}$ sets $c_R^j = c_T^j + 2$ on the j-th query for the first time. Then the previous state should be such that $c_R^{j-1} = c_T^{j-1} = c_T^j$ and the j-th query should be the message $M_{c_R^{j-1}+2}$ transmitted to $\mathcal{R}(\cdot)$. When $\mathcal{A}$ interacts with game $\mathcal{G}_2$ in the same way (in terms of the random coins), the j-th query $M_{c_R^{j-1}+2}$ would be stored in $\mathbf{Z}_{c_R^{j-1}+2}$ and procedure Finalize return "1". With this observation, we obtain the equality

$$\mathsf{Pr}_{\mathcal{M}\xleftarrow{\$}\{0,1\}^L}[\mathcal{G}(\mathcal{M})^{\mathcal{A}} \text{ sets } c_R = c_T + 2] = \mathsf{Pr}[\mathcal{G}_2^{\mathcal{A}} \Rightarrow 1].$$

From now on, we fix every random coin of $\mathcal{A}$. When a message $B \in \{0,1\}^m$ is added to $\mathbf{Z}_i$ for $i \in I$, it holds that $c_T = c_R = i - 2$, which means procedures $\mathcal{T}(\cdot)$ and $\mathcal{R}(\cdot)$ have not referred to the message block M_i. Note that the reference of M_i in the line 9 of $\mathcal{R}(\cdot)$ with $c_T < c_R$ never happens in game $\mathcal{G}_2$. Therefore it follows that each set Z_i, $i \in I$, is determined by $M_0, \ldots, M_{i-1}$, denoted $Z_i(M_0, \ldots, M_{i-1})$. Let

$$u = |\{(M_i)_{i\in[0,2q+2]} \in \{0,1\}^{(2q+3)m} : \exists\ i \in I \text{ such that } M_i \in Z_i\}|$$

```
procedure Initialize
1: $c_T,\ c_R,\ rev,\ add \leftarrow 0$
2: $\mathbf{Z}_i \leftarrow \emptyset$ for $i \in I = \{2, 4, \ldots, 2q\}$
3: $\mathcal{M} \xleftarrow{\$} \{0,1\}^L$

procedure $\mathcal{T}(A)$
1: if $A =$ "query" then
2:     output $M_{c_T}$
3: else if $A = M_{c_T+1}$ then
4:     $c_T \leftarrow c_T + 2$
5:     output $M_{c_T}$

procedure Finalize
1: if $\exists\ i \in I$ such that $M_i \in \mathbf{Z}_i$ then
2:     return 1
3: else
4:     return 0

procedure $\mathcal{R}(B)$
1: if $B =$ "init" then
2:     if $rev = 1$ then $add \longleftarrow 1$
3:     output "query"
4: else if $c_T = c_R$ then
5:     $\boxed{\mathbf{Z}_{c_R+2} \longleftarrow \mathbf{Z}_{c_R+2} \cup \{B\}}$
6: else if $B = M_{c_R}$ then
7:     $rev \longleftarrow 1$
8:     output $M_{c_R+1}$
9: else if $B = M_{c_R+2}$ then
10:    $c_R \longleftarrow c_R + 2$
11:    $rev \longleftarrow 0$
12:    if $add = 0$ then
13:        output "accept"
14:    else
15:        $add \longleftarrow 0$
16:        $rev \longleftarrow 1$
17:        output $M_{c_R+1}$
```

Fig. 5. Game $\mathcal{G}_2$. The boxed statement records every attempt of setting the counters c_T and c_R to the desynchronization state of ($c_R = c_T + 2$).

be the number of sequences $(M_i)_{i\in[0,2q+2]}$ that result in an desynchronization state of $c_R = c_T + 2$. Then we have the estimate

$$\begin{aligned}
u &\le 2^{2qm} \sum_{(M_0,M_1)\in\{0,1\}^{2m}} |Z_2(M_0, M_1)| + 2^{(2q-2)m} \sum_{(M_0,\ldots,M_3)\in\{0,1\}^{4m}} |Z_4(M_0, \ldots, M_3)| \\
&\quad + \ldots + 2^{2m} \sum_{(M_0,\ldots,M_{2q-1})\in\{0,1\}^{2qm}} |Z_{2q}(M_0, \ldots, M_{2q-1})| \\
&= 2^{2m} \sum_{(M_0,\ldots,M_{2q-1})\in\{0,1\}^{2qm}} (|Z_2(M_0, M_1)| + \ldots + |Z_{2q}(M_0, \ldots, M_{2q-1})|) \\
&\le q2^{(2q+2)m}, \qquad (8)
\end{aligned}$$

since the summand "$|Z_2(M_0, M_1)| + \ldots + |Z_{2q}(M_0, \ldots, M_{2q-1})|$" is not greater than the number of queries. Therefore, we have

$$\mathsf{Pr}[\mathcal{G}_2^{\mathcal{A}} \Rightarrow 1] \le \max_{\mathcal{C}} \frac{u}{2^{(2q+3)m}} \le \frac{q2^{(2q+2)m}}{2^{(2q+3)m}} = \frac{q}{2^m},$$

where $\mathcal{C}$ denotes the set of the random coins of $\mathcal{A}$. The inequality for the probability P_2 can be proved in a similar way, by using the modified game $\mathcal{G}_3$ defined in Figure 6, where $J = \{1, 3, 5, \ldots, 2q - 1\}$. □

Now we can prove the following theorem.

```
procedure Initialize
1: $c_T,\ c_R,\ rev,\ add \leftarrow 0$
2: $\mathbf{Z}_i \leftarrow \emptyset$ for $i \in J = \{1, 3, 5, \ldots, 2q-1\}$
3: $\mathcal{M} \xleftarrow{\$} \{0,1\}^L$

procedure $\mathcal{T}(A)$
1: if $A =$ "query" then
2:     output $M_{c_T}$
3: else if $c_T > c_R$ then
4:     $\boxed{\mathbf{Z}_{c_T+1} \leftarrow \mathbf{Z}_{c_T+1} \cup \{A\}}$
5: else if $c_T = c_R$ and $rev = 0$ then
6:     $\boxed{\mathbf{Z}_{c_T+1} \leftarrow \mathbf{Z}_{c_T+1} \cup \{A\}}$
7: else if $A = M_{c_T+1}$ then
8:     $c_T \leftarrow c_T + 2$
9:     output $M_{c_T}$

procedure Finalize
1: if $\exists\ i \in J$ such that $M_i \in \mathbf{Z}_i$ then
2:     return 1
3: else
4:     return 0

procedure $\mathcal{R}(B)$
1: if $B =$ "init" then
2:     if $rev = 1$ then $add \longleftarrow 1$
3:     output "query"
4: else if $B = M_{c_R}$ then
5:     $rev \longleftarrow 1$
6:     output $M_{c_R+1}$
7: else if $B = M_{c_R+2}$ then
8:     $c_R \longleftarrow c_R + 2$
9:     $rev \longleftarrow 0$
10:    if $add = 0$ then
11:        output "accept"
12:    else
13:        $add \longleftarrow 0$
14:        $rev \longleftarrow 1$
15:        output $M_{c_R+1}$
```

Fig. 6. Game $\mathcal{G}_3$. The boxed statements record every attempt of setting the counters c_T and c_R and the flag *rev* to the desynchronization state of ($c_T = c_R + 4$) or ($c_T = c_R + 2$ and $rev = 0$).

Theorem 4.1. *Let $\mathcal{S} = (\mathsf{Up}, \mathsf{F})$ be a (t, ϵ, L)-PPRSG for $L = (2q+3)m$. For a desynchronization-adversary $\mathcal{A}(t, q)$, we have*

$$\mathsf{Pr}_{K \xleftarrow{\$} \{0,1\}^d}[\mathcal{A} \textit{ succeeds in a desynchronization attack}\,] \leq \frac{4q}{2^m} + \epsilon.$$

Proof. From the inequalities (3), (7) and Lemma 4.1, we can construct a distinguisher D such that

$$\mathsf{Pr}_{K \xleftarrow{\$} \{0,1\}^d}[D(G_{\mathcal{S}}(K)) \Rightarrow 1] = \delta,$$

and

$$\mathsf{Pr}_{\mathcal{M} \xleftarrow{\$} \{0,1\}^L}[D(\mathcal{M}) \Rightarrow 1] \leq \frac{4q}{2^m}.$$

Since the advantage of the distinguisher D is not greater that ϵ, we have

$$\delta \leq \frac{4q}{2^m} + \epsilon,$$

which completes the proof. □

Remark 4.1. Our security proof is easily extended to a multi-tag setting, where an adversary makes oracle queries to a multiple number of tags as well as a reader.

4.2 Security against a tag-impersonation attack

We model a tag-impersonation-adversary $\mathcal{A} = \mathcal{A}(q,t)$ as a probabilistic Turing machine of run time t that makes total q queries to $\mathcal{T}$ and $\mathcal{R}$. The execution of $\mathcal{A}$ with $\mathcal{T}$ and $\mathcal{R}$ yields a transcript

$$\mathrm{T} = (\mathbf{S}^0, \mathbf{T}^1, \mathbf{S}^1, \ldots, \mathbf{S}^{q-1}, \mathbf{T}^q, \mathbf{S}^q)$$

of query-response pairs and states, where

$$\mathbf{S}^i = (S^i_T, S^i_R, rev^i, add^i)$$

represents the state of $\mathcal{T}$ and $\mathcal{R}$ determined by the i-th query of $\mathcal{A}$ for $0 \le i \le q$, and

$$\mathbf{T}^i = (M^i_Q, M^i_R, x^i)$$

represents the query-response pair of the i-th query for $1 \le i \le q$. Here M^i_Q, M^i_R and $x^i \in \{\mathcal{T}, \mathcal{R}\}$, respectively, represent a query message, a response message and the interface that the adversary made the query to.

Now suppose that the transcript T satisfies the following condition.

– There exist $1 \le i < j \le q$ such that
  1. $\mathbf{T}^i = (\text{“}init\text{”}, *, \mathcal{R})$, $\mathbf{T}^j = (*, \text{“}accept\text{”}, \mathcal{R})$, and
  2. $M^h_R \ne M^{h'}_Q$ for every $i \le h < h' \le j$ such that $x^h = \mathcal{T}$ and $x^{h'} = \mathcal{R}$.

Then we say that $\mathcal{A}$ *succeeds in a tag-impersonation attack against* $\mathcal{T}$ *and* $\mathcal{R}$. The above condition implies that the adversary $\mathcal{A}$ opened a new session with “*init*” message, and derived “*accept*” message from the reader $\mathcal{R}$ without using any response from the tag $\mathcal{T}$. Now we show that a tag-impersonation-adversary succeeding in a tag-impersonation attack results in a desynchronization state between $\mathcal{T}$ and $\mathcal{R}$. Suppose that the transcript T satisfies the above condition, and $\mathbf{S}^{j-1}$ is a synchronization state. Then we have three possible cases as follows.

Case 1. We assume that $S^{j-1}_{\mathcal{T}} = S^{j-1}_{\mathcal{R}}$. The “*accept*” message of $\mathbf{T}^j$ implies that the variable $S_{\mathcal{R}}$ is updated on the j-th query. The update results in a desynchronization state with

$$S^j_{\mathcal{R}} = \mathsf{Up}^2(S^{j-1}_{\mathcal{R}}) = \mathsf{Up}^2(S^{j-1}_{\mathcal{T}}) = \mathsf{Up}^2(S^j_{\mathcal{T}}).$$

Case 2. We assume that $S^{j-1}_{\mathcal{T}} = \mathsf{Up}^2(S^{j-1}_{\mathcal{R}})$ and there is no update in the variable $S_{\mathcal{T}}$ between the i-th query and the j-th query. Then we have $S^{i-1}_{\mathcal{T}} = S^{j-1}_{\mathcal{T}}$. Suppose that the state $\mathbf{S}^{i-1}$ is a synchronization state of type 1 or 2, i.e., $S^{i-1}_{\mathcal{T}} = S^{i-1}_{\mathcal{R}}$. If $S^{j-1}_{\mathcal{R}} = \mathsf{Up}^2(S^{j'}_{\mathcal{R}})$ for some $i-1 \le j' < j-1$, then we have

$$S^{j'}_{\mathcal{T}} = S^{j-1}_{\mathcal{T}} = \mathsf{Up}^2(S^{j-1}_{\mathcal{R}}) = \mathsf{Up}^4(S^{j'}_{\mathcal{R}}),$$

which is a desynchronization state of type 2 for $\mathbf{S}^{j'}$. If $S_{\mathcal{R}}^{j-1} = S_{\mathcal{R}}^{i-1}$, then we have

$$\mathsf{Up}^2(S_{\mathcal{R}}^{j-1}) = S_{\mathcal{T}}^{j-1} = S_{\mathcal{T}}^{i-1} = S_{\mathcal{R}}^{i-1} = S_{\mathcal{R}}^{j-1},$$

which is a desynchronization state of type 3 for $\mathbf{S}^{j-1}$. Finally, suppose that $\mathbf{S}^{i-1}$ is a synchronization state of type 3, i.e., $S_{\mathcal{T}}^{i-1} = \mathsf{Up}^2(S_{\mathcal{R}}^{i-1})$ and $rev^{i-1} = 1$. Since $rev^{i-1} = 1$, we have $add^i = 1$. Since the "*accept*" message is returned only if $add = 0$, we have $add^{j-1} = 0$, which implies that the variable $S_{\mathcal{R}}$ has been updated before the $(j-1)$-th query. Since the j-th query results in another update of $S_{\mathcal{R}}$, we can find $j' \in [i+1, j]$ such that

$$S_{\mathcal{R}}^{j'} = \mathsf{Up}^4(S_{\mathcal{R}}^{i-1}) = \mathsf{Up}^2(S_{\mathcal{T}}^{i-1}) = \mathsf{Up}^2(S_{\mathcal{T}}^{j'}).$$

Case 3. We assume that $S_{\mathcal{T}}^{j-1} = \mathsf{Up}^2(S_{\mathcal{R}}^{j-1})$ and there is an update in the variable $S_{\mathcal{T}}$ between the i-th query and the j-th query. In order for the reader to return the message "*accept*" for the j-th query, it should be the case that $M_Q^j = \mathsf{F}(\mathsf{Up}^2(S_{\mathcal{R}}^{j-1})) = \mathsf{F}(S_{\mathcal{T}}^{j-1})$. Suppose that the h-th query, $i \le h < j$, updates the variable $S_{\mathcal{T}}^{h-1} (\neq S_{\mathcal{T}}^{j-1})$ as $S_{\mathcal{T}}^{j-1}$. Then it follows that $M_R^h = \mathsf{F}(S_{\mathcal{T}}^{j-1})$, which contradicts the second condition for the transcript to satisfy.

To summarize, we can construct a desynchronization adversary of success probability at least δ from a tag-impersonation-adversary of success probability δ. By Theorem 4.1, we obtain the following theorem.

Theorem 4.2. *Let $\mathcal{S} = (\mathsf{Up}, \mathsf{F})$ be a (t, ϵ, L)-PPRSG for $L = (2q+3)m$. For a tag-impersonation-adversary $\mathcal{A}(t, q)$, we have*

$$\mathsf{Pr}_{K \xleftarrow{\$} \{0,1\}^d}[\mathcal{A} \textit{ succeeds in a tag-impersonation attack }] \le \frac{4q}{2^m} + \epsilon.$$

5 S^*-protocol: modification for mutual authentication

Certain RFID systems might require reader authentication. In this section, we slightly modify our protocol so that the resulting protocol, called S^*-protocol, provides mutual authentication of tag and reader. As seen in Figure 7, there are only two differences in tag and reader algorithms as compared to S-protocol. One is that each tag stores and updates an auxiliary flag $add_{\mathcal{T}}$, and the other is that a reader always set flags rev and add to zero on an "*init*" message. Then a session of S^*-protocol is completed within 6 passes of messages as seen in Figure 8. It is easy to see that any desynchronization-adversary or tag-impersonation adversary on S^*-protocol can be transformed, respectively, into a desynchronization adversary or a tag-impersonation adversary on S-protocol. Now we model a reader-impersonation-adversary $\mathcal{A} = \mathcal{A}(q, t)$ as a probabilistic Turing machine of run time t that makes total q queries to $\mathcal{T}$ and $\mathcal{R}$. Let

$$\mathrm{T} = (\mathbf{S}^0, \mathbf{T}^1, \mathbf{S}^1, \ldots, \mathbf{S}^{q-1}, \mathbf{T}^q, \mathbf{S}^q)$$

```
1: if $A =$ "query" then
2:     $add_{\mathcal{T}} \longleftarrow 1$
3:     output $\mathsf{F}(S_{\mathcal{T}})$
4: else if $A = \mathsf{F}(\mathsf{Up}(S_{\mathcal{T}}))$ then
5:     $S_{\mathcal{T}} \leftarrow \mathsf{Up}^2(S_{\mathcal{T}})$
6:     if $add_{\mathcal{T}} = 0$ then
7:         output ("accept", $\mathsf{F}(S_{\mathcal{T}})$)
8:     else
9:         $add_{\mathcal{T}} \longleftarrow 0$
10:        output $\mathsf{F}(S_{\mathcal{T}})$
```

(a) $\mathcal{T}$ on an incoming message A

```
1: if $B =$ "init" then
2:     $rev \longleftarrow 1$
3:     $add \longleftarrow 1$
4:     output "query"
5: else if $B = \mathsf{F}(S_{\mathcal{R}})$ then
6:     $rev \longleftarrow 1$
7:     output $\mathsf{F}(\mathsf{Up}(S_{\mathcal{R}}))$
8: else if $B = \mathsf{F}(\mathsf{Up}^2(S_{\mathcal{R}}))$ then
9:     $S_{\mathcal{R}} \longleftarrow \mathsf{Up}^2(S_{\mathcal{R}})$
10:    $rev \longleftarrow 0$
11:    if $add = 0$ then
12:        output "accept"
13:    else
14:        $add \longleftarrow 0$
15:        $rev \longleftarrow 1$
16:        output $\mathsf{F}(\mathsf{Up}(S_{\mathcal{R}}))$
```

(b) $\mathcal{R}$ on an incoming message B

Fig. 7. Tag algorithm $\mathcal{T}$ and reader algorithm $\mathcal{R}$ for mutual authentication. $\mathcal{R}$ interprets a message of the form ("*accept*", B) as B.

be a transcript obtained from the execution of $\mathcal{A}$ with $\mathcal{T}$ and $\mathcal{R}$. Here

$$\mathbf{S}^i = (S^i_T, add^i_{\mathcal{T}}, S^i_R, rev^i, add^i)$$

represents the state of $\mathcal{T}$ and $\mathcal{R}$ determined by the i-th query of $\mathcal{A}$ for $0 \leq i \leq q$, and

$$\mathbf{T}^i = (M^i_Q, M^i_R, x^i)$$

represents the query-response pair of the i-th query for $1 \leq i \leq q$, as in the previous section. If the transcript T satisfies the following condition, then we say that $\mathcal{A}$ *succeeds in a reader-impersonation attack against* $\mathcal{T}$ *and* $\mathcal{R}$.

– There exist $1 \leq i < j \leq q$ such that
  1. $\mathbf{T}^i = (\text{“}query\text{”}, *, \mathcal{T})$, $\mathbf{T}^j = (*, (\text{“}accept\text{”}, *), \mathcal{T})$, and
  2. $M^h_R \neq M^{h'}_Q$ for every $i \leq h < h' \leq j$ such that $x^h = \mathcal{R}$ and $x^{h'} = \mathcal{T}$.

The above condition implies that the adversary $\mathcal{A}$ has transmitted "*query*" message, and derived "*accept*" message from the tag $\mathcal{T}$ without using any response from the reader $\mathcal{R}$. Now we show that an adversary succeeding in a reader-impersonation attack results in a desynchronization state between $\mathcal{T}$ and $\mathcal{R}$.

Case 1. We assume that there is no update in the variable $S_{\mathcal{R}}$ between the i-th query and the j-th query. Let $\mathbf{S}^i$ be a synchronization state such that $S^i_{\mathcal{T}} = \mathsf{Up}^v(S^i_{\mathcal{R}})$ for $v = 0$ or 2. Since $add^i_{\mathcal{T}} = 1$ and $add^{j-1}_{\mathcal{T}} = 0$, we have $S^{j-1}_{\mathcal{T}} = \mathsf{Up}^u(S^i_{\mathcal{T}})$ for some $u \geq 2$. From the message $M^j_R = (\text{“}accept\text{”}, *)$, we see that

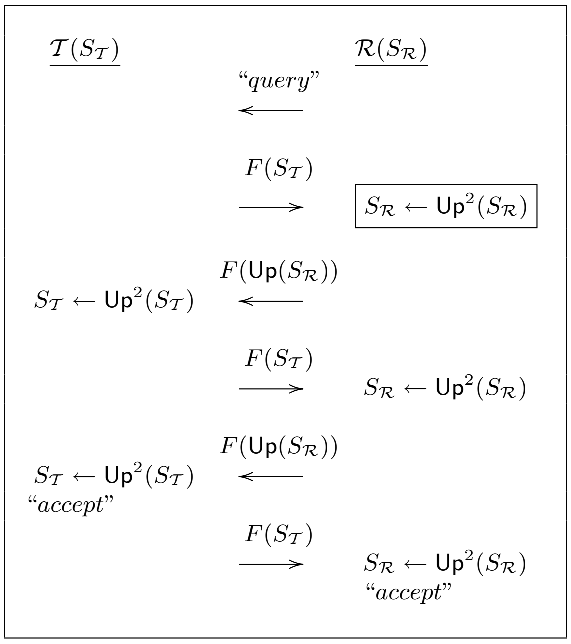
$\underline{\mathcal{T}(S_{\mathcal{T}})}$
$\underline{\mathcal{R}(S_{\mathcal{R}})}$
"*query*"
$F(S_{\mathcal{T}})$
$\boxed{S_{\mathcal{R}} \leftarrow \mathsf{Up}^2(S_{\mathcal{R}})}$
$F(\mathsf{Up}(S_{\mathcal{R}}))$
$S_{\mathcal{T}} \leftarrow \mathsf{Up}^2(S_{\mathcal{T}})$
$F(S_{\mathcal{T}})$
$S_{\mathcal{R}} \leftarrow \mathsf{Up}^2(S_{\mathcal{R}})$
$F(\mathsf{Up}(S_{\mathcal{R}}))$
$S_{\mathcal{T}} \leftarrow \mathsf{Up}^2(S_{\mathcal{T}})$
"*accept*"
$F(S_{\mathcal{T}})$
$S_{\mathcal{R}} \leftarrow \mathsf{Up}^2(S_{\mathcal{R}})$
"*accept*"

Fig. 8. Message flow of S^*-protocol. The boxed statement is executed only if $\mathcal{T}$ and $\mathcal{R}$ are in a synchronization state of type 3.

the variable $S_{\mathcal{T}}^{j-1}$ is updated on the j-th query, i.e., $S_{\mathcal{T}}^{j} = \mathsf{Up}^2(S_{\mathcal{T}}^{j-1})$. Then we obtain a desynchronization state

$$S_{\mathcal{T}}^{j} = \mathsf{Up}^{u+2}(S_{\mathcal{T}}^{i}) = \mathsf{Up}^{v+u+2}(S_{\mathcal{R}}^{i}) = \mathsf{Up}^{v+u+2}(S_{\mathcal{R}}^{j}).$$

Case 2. We assume that there is an update in the variable $S_{\mathcal{R}}$ between the i-th query and the j-th query. If the h-th query, $i \leq h < j$, is the last query that makes an update of $S_{\mathcal{R}}$, then it should be the case $S_{\mathcal{R}}^{h} = S_{\mathcal{R}}^{j-1} = S_{\mathcal{T}}^{j-1}$, since otherwise we could obtain a desynchronization state in either $\mathbf{S}^{j-1}$ or $\mathbf{S}^{j}$. If $M_R^h =$ "*accept*", then rev^h is set to 0. Then the j-th query results in a desynchronization state such that $S_{\mathcal{T}}^{j} = \mathsf{Up}^2(S_{\mathcal{R}}^{j})$ and $rev^j = 0$. Otherwise we have

$$M_R^h = \mathsf{F}(\mathsf{Up}(S_{\mathcal{R}}^{h})) = \mathsf{F}(\mathsf{Up}(S_{\mathcal{T}}^{j-1})) = M_Q^j,$$

which contradicts the second condition for the transcript to satisfy.

To summarize, we can construct a desynchronization adversary of success probability at least δ from a reader-impersonation-adversary of success probability δ. By Theorem 4.1, we obtain the following theorem.

Theorem 5.1. *Let $\mathcal{S} = (\mathsf{Up}, \mathsf{F})$ be a (t, ϵ, L)-PPRSG for $L = (2q+3)m$. For a reader-impersonation-adversary $\mathcal{A}(t, q)$, we have*

$$\mathsf{Pr}_{K \overset{\$}{\leftarrow} \{0,1\}^d}[\mathcal{A} \textit{ succeeds in a reader-impersonation attack }] \leq \frac{4q}{2^m} + \epsilon.$$

Remark 5.1. The flag *rev* can be removed in the reader algorithm 7(b) since it does not affect any response or update of $\mathcal{R}$. The flag *rev* is only used for security proof.

6 Conclusion

In this paper, we have proposed S-protocol, an authentication protocol based on a special class of PRSGs. We also have presented a formal proof of availability and security of our protocol. Since most existing stream ciphers can be used as a building block of S-protocol, S-protocol is expected to be suitable for use in highly constrained environments such as RFID systems. We now pose two open problems. One is to provide S-protocol with untraceability, as required in many RFID applications. The other is to reduce the number of rounds of S-protocol for more efficient communication. As a partial answer to the first question, we might consider an approach where a tag does not claim its ID in an explicit way, but a reader identifies the tag in a back-end data base by using the keystream block transmitted in response to *query* message.

References

1. F. Arnault, T. P. Berger and C. Lauradoux, Update on F-FCSR stream cipher, State of the Art of Stream Ciphers 2006(SASC 2006), Workshop Record, pp. 267–277, February, 2006.
2. G. Avoine, P. Oechslin, A scalable and provably secure hash based RFID protocol, Workshop on Pervasive Computing and Communication Security(PerSec) 2005, March 2005.
3. S. Babbage and M. Dodd, The stream cipher MICKEY-128 2.0. eSTREAM, ECRYPT Stream Cipher Project, 2006.
4. C. De. Canniere and B. Preneel, TRIVIUM-a stream cipher construction inspired by block cipher design principles. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/030, 2005.
5. T. Dimitriou, A lightweight RFID protocol to protect against traceability and cloning attacks, Conference on Security and Privacy for Emerging Areas in Communication Networks(SecureComm) 2005, September 2005.
6. M. Feldhofer, S. Dominicus and J. Wolkerstorfer, Strong authentication for RFID systems using the AES algorithm, Proceedings of CHES '04, LNCS 3156, pp. 357–370, 2004.
7. M. Feldhofer, J. Wolkerstorfer and V. Rijmen, AES implementation on a grain of sand, Information Security, IEE Proceedings, vol. 152, no. 1, pp. 13–20, 2005.
8. O. Goldreish, S. Goldwasser and S. Micali, How to construct pseudorandom functions, Journal of the ACM, 33(4), 1986.
9. T. Good and M. Benaissa, Hardware results for selected stream cipher candidates, State of the Art of Stream Ciphers 2007(SASC 2007), Workshop Record, pp. 191–204, February, 2007.
10. H. Gilbert, M. J. B. Robshaw and Y. Seurin, HB#: increasing the security and efficiency of HB+, Eurocrypt 2008, LNCS 4965, pp. 361–378, Springer, 2008.

11. M. Hell, T. Johansson and W. Meier, Grain - a stream cipher for constrained environments. eSTREAM, ECRYPT Stream Cipher Project, 2006.
12. D. Henrici and P. M*ü*ller, Hash-based enhancement of location privacy for radio-frequency identification devices using varying identifiers, Workshop on Pervasive Computing and Communication Security(PerSec) 2004, March 2004.
13. D. Hong, J. Sung, S. Hong, J. Lim, S. Lee, B.-S. Koo, C. Lee, D. Chang, J. Lee, K. Jeong, H. Kim, J. Kim and S. Chee, HIGHT: a new block cipher suitable for low-resource device, Proceedings of CHES '06, LNCS 4249, pp. 46–59, 2006.
14. A. Juels and S. A. Weis, Authenticating pervasive devices with human protocols, Crypto 2005, LNCS 3126, pp. 293–308, Springer, 2005.
15. A. Kats and J. S. Shin, Parallel and concurrent security of the HB and HB+ protocols, Eurocrypt 2006, LNCS 4004, pp. 73–87, Springer, 2006.
16. T. van Le, M. Burmester and B. de Medeiros, Univerally composable and forward-secure RFID authentication and authenticated key exchange, Proceedings of the ACM Symposium on Information, Computer and Communications Security(ASIACCS 2007), 2007.
17. M. Ohkubo, K. Suzuki and S. Kinoshita, Efficient hash-chain based RFID privacy protection scheme, International Conference on Ubiquitous Computing - Ubicomp, Workshop Privacy: Current Status ans Future Directions, September 2004.
18. A. Poschmann, G. Leander, K. Schramm and C. Paar, New light-weight crypto algorithms for RFID, Proceedings of the 2007 IEEE International Symposium on Circuits and Systems, 2007.
19. K. Rhee, J. Kwak, S. Kim and D. Won, Challenge-response based RFID authentication protocol for distributed database environment, International Conference on Security in pervasive Computing(SPC) 2005, April 2005.
20. G. Tsudik, YA-TRAP: Yet another trivial RFID authentication protocol, International Conference on Pervasive Computing and Communications(PerCom 2006), 2006.
21. S.A. Weis, S.E. Sarma, R.L. Rivest and D.W. Engels, Security and Privacy Aspects of Low-Cost Radio Frequency Identification Systems, Security in Pervasive Computing, International Conference on Security in pervasive Computing(SPC) 2003, March 2003.