A Study on Computational Formal Verification for Practical Cryptographic Protocol: The Case of Synchronous RFID Authentication

Yoshikazu HanataniI[1,2], Miyako Ohkubo[3], Sin'ichiro Matsuo[3], Kazuo Sakiyama[2], and Kazuo Ohta[2]

[1] Toshiba Cor oration,Chofugaoka Komukai Toshiba-cho 1, Saiwai-ku, Kawasaki-shi, Kanagawa 212-8582, Ja an yoshikazu.hanatani@toshiba.co.jp

[2] The University of Electro-Communications,Chofugaoka 1-5-1, Chofu-shi, Tokyo 182-8585, Ja an {hana,saki,ota}@ice.uec.ac.jp

[3] The National Institute of Information and Communications Technology, 4-2-1 Nukui-Kitamachi, Koganei, Tokyo 184-8795, Ja an {m.ohkubo, smatsuo}@nict.go.jp

Abstract. Formal verification of cry togra hic rotocols has a long history with a great number of successful verification tools created. Recent rogress in formal verification theory has brought more owerful tools ca able of handling com utational assum tion, which leads to more reliable verification results for information systems.

In this a er, we introduce an effective scheme and studies on a lying com utational formal verification toward a ractical cry togra hic rotocol. As a target rotocol, we reconsider a security model for RFID authentication with a man-in-the-middle adversary and communication fault. We define three model and security roofs via a game-based a roach that, in a com utational sense, makes our security models com atible with formal security analysis tools. Then we show the combination of using a com utational formal verification tool and handwritten verification to overcome the com utational tool's limitations. We show that the target RFID authentication rotocol is robust against the above-mentioned attacks, and then rovide game-based (handwritten) roofs and their verification via Cry toVerif.

Keywords: RFID, authentication, rivacy, formal roofs

1 Introduction

1.1 Background

Cryptographic protocols are widely used in real-life information systems, and serve as significant components in fulfilling systems' complicated security requirements. To put a system into use, it must be demonstrated to have security sufficient to satisfy the fundamental security requirements. Formal verification theory and tools provide a good method to meet this challenge, and many attempts have been made in their development over the last 30 years. As the use

of such theory and tools becomes more common, the framework for reliably using them is being standardized in ISO/IEC 29128. In the standardizing process of ISO/IEC 29128, the use and limitations of computational verification tools, which offer superior power and reliability, becomes a major issue [15]. Thus, we need much experiences of applying computational verification tool to real-life cryptographic protocols.

For example real-life protocol, a great number of low-power devices called RFID tags, which communicate over wireless channels, have entered into everyday use by executing cryptographic protocols. In most cases, RFID tags are used for identifying goods, authenticating parties' legitimacy, detecting fakes, and billing for services; applications that demand secure authentication of each tag. If a tag's output is fixed or related to a different type of authentication, privacy issues also arise in which an adversary can trace the tag and the activity of the owner. Therefore, most research on RFID authentication protocols realizes the importance of tag-unforgeability and forward privacy. To verify the security and privacy of an RFID-authentication protocol, we must consider two aspects: security on wireless communication channels and computational security.

Though a large number of proposed secure protocols assume wired networks, the next consideration is how to deal with issues caused via wireless networks. In wireless network environments where RFID is used, the adversary has opportunities to conduct, for instance, man-in-the-middle or relay attacks. Connections are less stable than in a wired setting, so we have to consider robustness against communication errors. Formal verification is a good approach for dealing with such communication-related security issues, and its recent progress in cryptographic protocols helps achieve rigorous verification, even for the computational security of cryptographic building blocks. This rigor is needed to foster a high level of trust in actual RFID authentication protocols. However, for this application we do not have results that cover both communication-related issues and computational security. We also need to construct a security model and definition, secure protocol, and security proofs for such situations in order to reveal the security strength in actual use.

1.2 Our Contribution

In this paper, we show how we can apply computational verification theory and tool to real-life cryptographic protocol. The target protocol of this work is robust RFID-authentication protocol, which is tolerant against communication errors. The paper's main contributions are in two areas. (1) We provide a formal security model and definitions that deal with man-in-the-middle adversaries and communication faults. This model is similar to key exchange protocol security models and suitable for rigorously estimating the success probability of attacks. (2) We prove the security and privacy of a robust RFID authentication protocol that satisfies the above model. We choose a hash-based scheme due to the extremely high computation cost in a public key-based scheme. The protocol is based on the OSK protocol [17] , which can provide forward privacy. Because OSK protocol is not free from desynchronization problem caused from communication errors, we

combine a mechanism that synchronizes the internal status of the tag and reader with OSK protocol to overcome it. We prove the security of our proposed scheme using CryptoVerif formal verification tool. There are limitations in formalizing of security notion of cryptographic protocols which are practical and useful in real-life setting. For example, though desynchronization and forward privacy are needed for practical sense for cryptographic protocols, we cannot easily formalize this environment. To solve this limitation, we combine the CryptoVerif and handwritten proof. We give formalizations of security notions without forward privacy and a simplified proposed protocol for CryptoVerif, and by using CryptoVerif show the protocol satisfies them. Next, By handwritten proof, we show the proposed protocol satisfies security notions with forward privacy if the simplified proposed protocol satisfies the security notions with forward privacy. This is evidently the first work in the RFID arena that defines the security notion and in a computational sense shows the security via a formal verification tool. This is one practical direction for application of computational formal verification.

1.3 Related Works

Many schemes exist for secure RFID authentication that protects privacy; and these are summarized in [1]. For security models for RFID authentication, Juels and Weis first proposed the privacy model [13]. Vaudenay then proposed a classification of security concepts for privacy regarding tag-authentication [19]. Paise and Vaudenay presented a classification of security concepts for mutual-authentication with privacy [18]. Hancke and Kuhn introduced a type of RFID authentication scheme that is robust against replay attacks and wireless settings using the distance-bounding protocol [11]. Because this protocol needs many rounds of communication we chose another construction.

A major contribution of this paper is proving the security of our scheme by using a formal verification tool. Security verification using formal methods has a long history dating back to the 1980s. Brusò et al. showed formal verification on privacy issues of the original OSK protocol by using ProVerif, which can conduct formal verification by assuming cryptographic building blocks as "ideal" [10]. Recently, combining "computational difficulty", a major concept in cryptography, and "automated verification", a prime benefit of the formal method, has become the subject of mainstream research in this area. Adabi and Rogaway pioneered work on the gap [3], and many works have subsequently been proposed. Practical tools such as CryptoVerif [8] [9], which we use in this paper, have also been proposed. Our result uses a game-based approach for representing the proof of a cryptographic protocol, and then uses a formal method to verify the (handwritten) proof.

2 Security Model and Definitions

2.1 RFID System

First we show informal descriptions for a general RFID system.

Communication: Communication between servers and clients is provided via a wireless network, upon which third parties can easily eavesdrop, and which can be cut or disturbed.
Client: We assume small devices like passive RFID tags as clients, called a set of "tags". Clients are only provided a poor level of electronic power by servers and can only perform light calculations. Memory in the client is not resilient against tamper attacks.
Server: We assume PCs and devices readers as servers. Generally, an RFID system tag communicates with readers over wireless channels, and the readers then communicate with servers over secure channels. We assume that the communication between reader and server is secure by using ordinal cryptographic techniques such as secure socket layer (SSL) and virtual private network (VPN). We therefore describe the communications in an RFID system using two players: client and server.

2.2 Adversary Model

An adversary can acquire information by eavesdropping or accessing a tag. Let such information be given to the adversary by the oracles.

The following oracles (i.e., server oracle, tag oracle, and random oracle) are used for modeling an adversary against a mutual authentication algorithm. All information that a tag output are described in output of tag oracle $\mathcal{T}$, and all information that a server output are described in output of tag oracle $\mathcal{S}$. All calculation results of functions are described in output of random oracle $\mathcal{R}$. So all information, that adversary can obtain by attacking, can be described by using oracle $\mathcal{T}$, $\mathcal{S}$, and $\mathcal{R}$. I.e., information that adversary can obtain by attacking are given by oracles.
Server oracle $\mathcal{S}$: This gives the adversary the same outputs as an honest server's output in the regular process of a mutual authentication algorithm. The adversary is allowed to request *Send* queries from the server oracle $\mathcal{S}$. If $\mathcal{S}$ receives a *Send* query, $Send(*)$, the same processes as the regular processes of an honest server are performed upon receiving a request $(*)$, and it then returns the output as an answer to the adversary. In the database $\mathcal{S}$-List of $\mathcal{S}$, all pairs of sid and I/O of $\mathcal{S}$, $(sid, Inputdata, Outputdata)$, are recorded.
Tag oracle $\mathcal{T}$: This gives the adversary the same outputs as an honest tag's output in the regular process of a mutual authentication algorithm. The adversary is allowed to request *Send* queries from and *Reveal* queries to the tag oracle $\mathcal{T}$. If $\mathcal{T}$ receives a *Send* query, $Send(*)$, the same processes as the regular processes of an honest tag are performed when it receives the request $(*)$, and it then returns the output as an answer to the adversary. If $\mathcal{T}$ receives a *Reveal* query, it returns the session's key, $sk_{ID,sid}$, and then in all sessions after the session the status is set as *Revealed*. In the database $\mathcal{T}$-List of $\mathcal{T}$, all pairs of sid and I/O of $\mathcal{T}$, $(sid, Inputdata, Outputdata)$, are recorded.
Random Oracle: Ideal Hash Functions (our proposed mutual authentication algorithm needs three hash functions: $\mathcal{H}_0$, $\mathcal{H}_1$, and $\mathcal{H}_2$)

We add information regarding electricity to the input of a tag in order to describe a situation in which a device uses external power to perform processes, such as with an RFID passive tag. In the model tag processes depend on the information on the external electricity. If a tag is given enough electricity, it processes completely and outputs a result. Otherwise, it processes as much as possible depending on the amount of external electricity and then cuts off.

2.3 Required Properties

There are some security requirements for secure mutual authentication using lightweight devices like RFID tags.

First, the basic properties are identification, authentication, and privacy, and then forward security and synchronization are extended properties. There are two perspectives on privacy issues. One is if a party identifies the ID there are risks of breaching the privacy of products or people connected to tags. Another is that if the output of a tag can be identified, the tag can be used as a tracing tool. For instance, a tagged person (or an item such as a books glasses, or a bag) can be traced by tracing the tag output. From these two standpoints, indistinguishability is required; i.e., a tag's output must be indistinguishable from random values. For authentication requirements, there are two sides: client authentication and server authentication. Mutual authentication should satisfy both requirements. Since low cost is a requirement of small devices such as RFIDs many are not able to satisfy the further requirement of tamper resistance. This gives an adversary the chance to acquire the secret key in these devices by tampering, which poses the risk of the tag's past output being traced, identified, and/or forged (i.e., client privacy and/or authenticity are breached). To protect the history in tampered devices, the property of forward security is required. Synchronization is another important requirement since small devices such as RFIDs communicate wirelessly and wireless communication is easily lost. Therefore when desynchronization occurs, a (state-full) protocol requires the property of self-synchronization.

2.4 Security Definitions

In this section we show security requirements for mutual authentication using lightweight devices. Note that κ : is a security parameter: i.e., a key length of hash functions.

Definition 1. (*Forward-secure* [*Client*] *Indistinguishability*) : *The simulator selects* $b \in \{0,1\}$ *randomly, exceeds the Key Generation Algorithm, and then gives the generated security parameter and secret key to* $\mathcal{S}$ *and* $\mathcal{T}$. *Adversary* $\mathcal{A}_{FI}$ *is allowed to access* $\mathcal{S}$ *and* $\mathcal{T}$ *in no particular order. (In the case of a random oracle model, the adversary is also allowed to access the random oracle in no particular order.)* $\mathcal{A}_{FI}$ *can at any time send a* Test *query with* sid^* *to* $\mathcal{T}^{sid}$. *A coin is flipped. If* $b = 0$, $\mathcal{T}$ *performs regular processes of the algorithm and returns the result to* $\mathcal{A}_{FI}$. *Otherwise, i.e.,* $b = 1$, $\mathcal{T}$ *selects a random value in the output*

space, and returns that random value to $\mathcal{A}_{FI}$. After $\mathcal{A}_{FI}$ sends a Test *query, $\mathcal{A}_{FI}$ is again allowed to access $\mathcal{S}$ and $\mathcal{T}$ in no particular order. At the end, $\mathcal{A}_{FI}$ outputs $\tilde{b} \in \{0,1\}$, and then stops. $\mathcal{A}_{FI}$ wins if $\tilde{b} = b$ and the status of $\mathcal{T}$ at sid^*, $\mathtt{T}_{\mathtt{sid}^*}$, is not Revealed.*

$$\mathsf{AdvCIND}_{\mathcal{A}} = \Pr\left[\textit{tildeb} = b \wedge \mathtt{T}_{\mathtt{sid}} \textit{is not revealed} \middle| \mathtt{b} \xleftarrow{R} \{0,1\}, \tilde{\mathtt{b}} \leftarrow \mathcal{A}_{\mathtt{FI}}^{\mathcal{S},\mathcal{T}} \right]$$

The mutual authentication algorithm satisfies Forward-secure (Client) Indistinguishability if $|\mathsf{AdvCIND}_{\mathcal{A}} - \frac{1}{2}|$ is negligible.

The above requirement provides the property of the tag's untraceability. Note that if the adversary cuts all responses from server to tag, the secret key in the tag cannot be updated. In such a case, if the tag is tampered, the tag can be traced. The above tag's tracing can be avoided by updating secret key EACH session whether previous session is finished or not. However, the maximum number of key updating must be set for verification and calculation cost for verification should be increased since there are several participants as session keys for a tag. Additionally, Dos-like attack presented in [14] can be applied. Therefore, in the paper traceability when key updating is obstructed is out of scope. As a practical matter, obstructing key updating of a target tag is not so easy, if tag's output seems random value and is changed every session like the proposed protocol. While, we care about "untraceability" for secret key updating. Roughly speaking, untraceability defined above is satisfied, if it is indistinguishable between (Tag/Server) outputs before key updating and after key updating. We can construct a protocol that satisfy this property with no limitation of the maximum number of key updating and small cost for verification. Moreover, Dos-like attack can not be applied.

Definition 2. (*Forward-secure* [*Client*] *Unforgeability*) : *The simulator selects $b \in \{0,1\}$ randomly, exceeds the Key Generation Algorithm, and then gives the generated security parameter and secret key to $\mathcal{S}$ and $\mathcal{T}$. Adversary $\mathcal{A}_{FU}$ is allowed to access $\mathcal{S}$ and $\mathcal{T}$ in no particular order. (For a random oracle model, the adversary is also allowed to access the random oracle in no particular order.) $\mathcal{A}_{FU}$ can at any time send a* Test *query with sid^* to $\mathcal{S}^{sid}$. $\mathcal{S}$ normally processes the algorithm and communicates with $\mathcal{A}_{FU}$. $\mathcal{A}_{FU}$ wins if the status of $\mathcal{S}$ at sid^*, $\mathtt{S}_{\mathtt{sid}^*}$, is Accepted and the status of $\mathcal{T}$ at sid^*, $\mathtt{T}_{\mathtt{sid}^*}$, is not Revealed, and the output of $\mathcal{S}^{sid}$ has not been requested from $\mathcal{T}^{sid}$.*

$$\mathsf{AdvCUF}_{\mathcal{A}} = \Pr\left[\begin{array}{l} \mathtt{S}_{\mathtt{sid}} \textit{is accepted} \wedge \mathtt{T}_{\mathtt{sid}} \textit{ is not revealed} \\ \wedge \textit{Output of } \mathcal{S}^{sid} \notin \mathcal{T}\textit{-List} \end{array} \middle| \textit{test}(sid) \leftarrow \mathcal{A}_{CU}^{\mathcal{S},\mathcal{T}} \right]$$

The mutual authentication algorithm satisfies Forward-secure (Client) Unforgeability if $\mathsf{AdvCUF}_{\mathcal{A}}$ is negligible.

The above requirement provides the property of impossibility of the tag's impersonation.

Definition 3. (*Forward-secure [Server] Unforgeability*) : *The simulator selects* $b \in \{0,1\}$ *randomly, exceeds the Key Generation Algorithm, and then gives the generated security parameter and secret key to* $\mathcal{S}$ *and* $\mathcal{T}$. *Adversary* $\mathcal{A}_{FU}$ *is allowed to access* $\mathcal{S}$ *and* $\mathcal{T}$ *in no particular order. (In the case of a random oracle model, the adversary is also allowed to access the random oracle in no particular order.)* $\mathcal{A}_{FU}$ *can at any time send a* Test *query with* sid^* *to* $\mathcal{T}^{sid}$. $\mathcal{T}$ *normally processes the algorithm and communicates with* $\mathcal{A}_{FU}$. $\mathcal{A}_{FU}$ *wins if the status of* $\mathcal{T}$ *at* sid^*, $\mathtt{T}_{\mathtt{sid}^*}$, *is Accepted and the status of* $\mathcal{T}$ *at* sid^*, $\mathtt{T}_{\mathtt{sid}^*}$, *is not Revealed, and the output of* $\mathcal{T}^{sid}$ *has not been requested from* $\mathcal{S}^{sid}$.

$$\mathsf{AdvSUF}_{\mathcal{A}} = \Pr\left[\begin{matrix}\mathtt{T}_{\mathtt{sid}} \textit{is accepted} \wedge \mathtt{T}_{\mathtt{sid}} \textit{is not revealed} \\ \wedge \textit{Output of } \mathcal{T}^{sid} \notin \mathcal{S}\textit{-List}\end{matrix} \middle| test(sid) \leftarrow \mathcal{A}_{SU}^{\mathcal{S},\mathcal{T}}\right]$$

The mutual authentication algorithm satisfies Forward-secure (Server) Unforgeability if $\mathsf{AdvFCSF}_{\mathcal{A}}$ *is negligible.*

The above requirement provides the property of impossibility of the server's impersonation.

Definition 4. (*Resiliency of Desynchronization*) *A mutual authentication between an honest server and tag succeeds with overwhelming probability, independent of the result of previous sessions.*

The above requirement provides the resistance property against DOS-like attacks, as presented by Juels [12].

3 Construction

In this section, we show a proposed mutual authentication scheme for lightweight devices like RFID tags.

3.1 Proposed Protocol and its Design Concepts

Our proposed protocol meets the security requirements set out in section 2. [4] The details of the protocol are shown in Fig. 1. In the figure, a lightweight device is described as an RFID-tag. Additionally, i is the times of key updating events in Server; i.e., counts of key updating in Server. $sk_{\mathsf{ID},i}$ is the i-th secret key of a tag. ID is the tag's ID. i' is the times of key updating events in Tag; i.e., counts of key updating in Tag. Data generated in a tag are described with quotation mark, for example, Y'. [5]

The basic concept is combining the OSK protocol and key update mechanism from mutual authentication. Let H_0 and H_2 be hash functions (random oracles).

[4] You can find e- rint version in [16]. Formally our rotocol is ublished in this a er.

[5] This figure ut being easy to understand above o timization of a rocess. We will show the descri tion utting a to riority on o timization of a rocess before very long.

H_0 and H_2 work in the same manner as the output function and key update function in the OSK protocol, respectively. In this protocol, the tag executes as follows. First, the client (i.e., tag) is requested from the server, a secret key is then input, which is recorded in the tag's memory, to H_2. The output is then input to H_0. At the end, the tag outputs the calculated results of H_0 to the server. The server receives the tag's output and then searches its database for the relevant secret key, which is shared with, and is unique data for each tag, to H_2 and then inputs the output to H_0 using the same processes.

To accomplish key updates in both the RFID tag and the server, we must cope with the problem of desynchronization. If only one side of the party updates its secret key, the protocol fails upon further authentication attempts. Such desynchronization not only causes failure of authentication, but also risks breaching privacy. We prevent desynchronization by using a key update in the mutual authentication which consists of two challenge-response protocols via hash functions. Key update is allowed just only if the result of verification is OK, AND the verified secret keys between tag and server are getting into synchronization. First, the server sends a random challenge, and then the tag calculates a response with H_0. The second challenge-response is initiated by the tag. The tag sends the challenge with the calculated response in Y'. The server calculates the response by using H_1 with the current secret or previous (old) secret. Note that the server stores both the current and previous secret key and which secret key is used depends on which one the server detects to calculate the received Y'. The server only calculates the value with the previous secret key if it detects that the received Y' is calculated using it, and, likewise, calculates the value with the current updated secret key if it detects that the received Y' is calculated using it. This mechanism deals with desynchronization by communication error. After that, the server sends the (second) response to the tag. Only when the tag confirms the response will it update the secret key. The basic security requirements are fulfilled with OSK-like construction, and desynchronization is solved by mutual authentication holding two secret keys, current and previous, on the server side.

3.2 Strong Points of the Proposed Scheme

Synchronization: The secret key is updated by both servers and clients, and if desynchronization occurs, the server can distinguish it and follow to the client's current state in the next session. This protocol can therefore solve the desynchronization problem.
Resiliency against DOS-like attack: Juels and Weis discussed the requirements of RFID protocols in [14]. They introduced the attack against a hash-chain-based scheme like a DOS-like attack against a server via the Internet. In the proposed scheme, the event in which the secret key updated in the tag proceeds only when the verification check is OK; and therefore a DOS-like attack cannot be applied to this scheme.
Saving computational cost of server: Generally, hash-based identification schemes like [17] require many server calculations in order to identify a tag (i.e., a client),

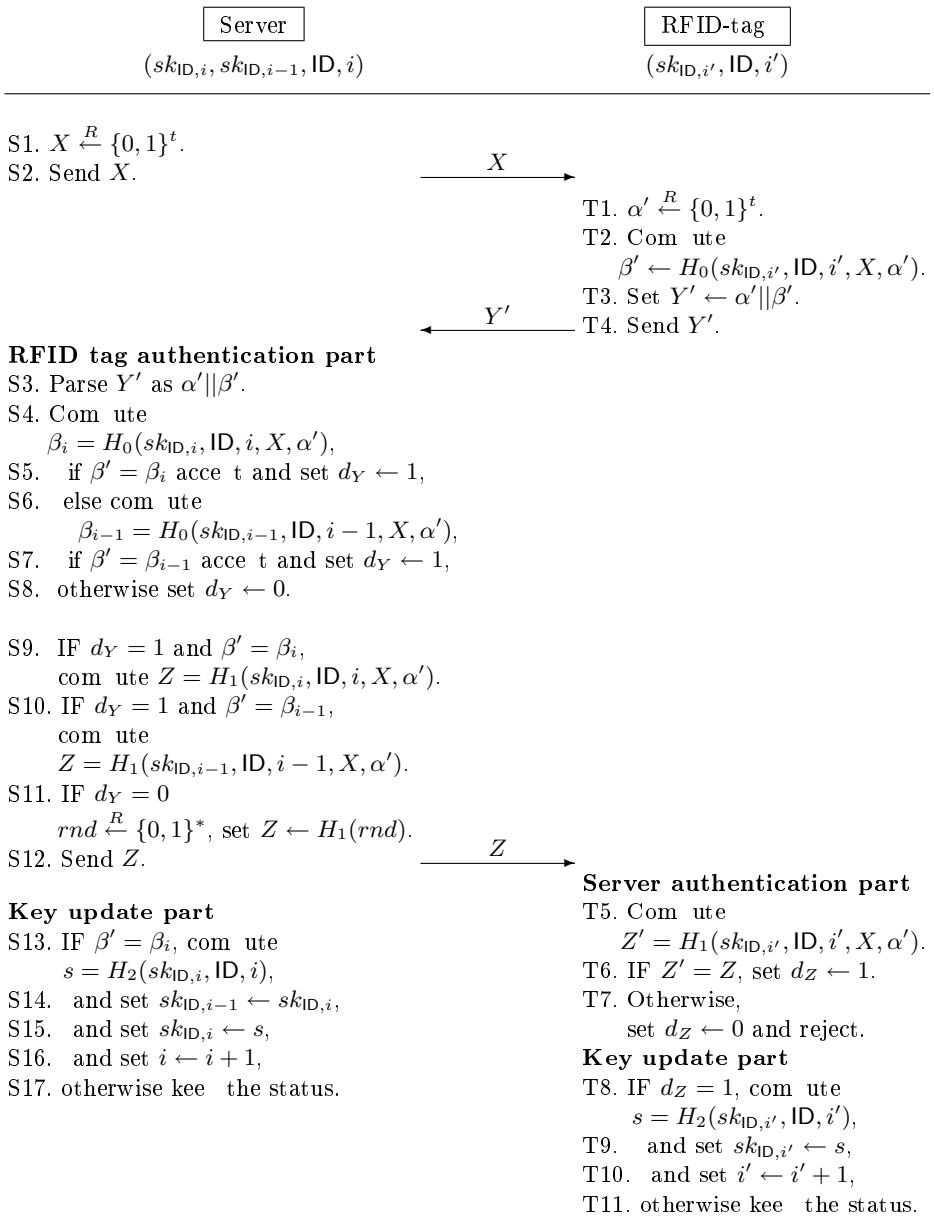

Fig. 1. Pro osed rotocol

and the server must compute $2m$ hash calculations for each tag, where m is a maximum number of updates of the secret key. In the proposed scheme, a server and a tag can share the current state of the common secret key; therefore a server only needs to compute two hash calculations for each tag, and the server's huge computational cost can thereby be saved.
Resiliency against replay attack: In the proposed scheme, fresh randomnesses chosen by both a server and a tag are used; therefore a replay attack cannot be applied to it.

4 Security Verification using CryptoVerif and Security Proofs

In this section, we show the security proof of the proposed protocol using CryptoVerif verification results.

4.1 Significance of Results

Two types of cryptographic protocol security exist. Symbolic security, e.g., Dolev-Yao model, assumes that cryptographic primitives that construct the cryptographic protocol are ideally secure. Some symbolic security can be automatically checked using formal verification tools such as ProVerif [6]. However, the symbolic security might not correspond to the real system's security. On the other hand, in computational security, the vulnerability of cryptographic primitives is considered. Computational security expresses the security of the real system compared with symbolic security. Recently, some frameworks that verify computational security have been proposed, and formal verification tools such as CryptoVerif[7] were developed.

Formal verification of privacy for RFID systems has been discussed[10,4]. Brusò et al. [10] gave a formal model for RFID privacy, expressing unlinkability and forward privacy as equivalences in the applied pi calculus[2], and proved the privacy of the OSK protocol[17] using Proverif. Only [10,4] discussed symbolic security.

We introduce formal models for unlinkability (Client indistinguishability) and mutual authenticity (Client unforgeability and Server unforgeability) of the proposed protocol without key update functionality by the probabilistic polynomial-time process calculus, and prove its computational security using CryptoVerif. This is evidently the first paper to show RFID system computational security by using a formal verification tool. Moreover, by manually using the formal verification results we prove froward privacy and forward mutual authenticity of the proposed protocol with key update functionality.

4.2 Formalization of proposed protocol

Since indexes of alignment cannot be controlled in CryptoVerif, the key update property in the proposed protocol cannot be described. Therefore we omit the

key update and obtain the CryptoVerif result. Using the result, we then show the entire security proof by handwriting. Details are as follows. The proposed protocol is constructed using three kind of random oracles. The key updating algorithm uses a "hash-chain." A new updated key is generated by a hash function, i.e., the previous session's secret key is set as input of the hash function, and its output is set as the new secret key. Since CryptoVerif does not permit control of indexes of a list of the hash function's output directly, we formalized the proposed protocol omitting the key updating algorithm and applied it to CryptoVerif. We set that in CryptoVerif, the adversary is not allowed to send *Reveal* queries since there is no protocol that satisfies security requirements after a session key is revealed. Instead of these queries, a state of revealing the secret key is described in which the adversary is given a secret key of the next session of a target session. With the above setting, the (simplified) proposed protocol can be applied to CryptoVerif. Using the CryptoVerif output, the entire security of the proposed protocol is shown in handwritten form. We first introduce the description of the random oracle formalized by Blanchet, et al. [9], and then show formalization of a proposed protocol and the security requirements. Due to space limitations, this paper omits the details of CryptoVerif rules. This information is available in the CryptoVerif manual [5].

Formalization of random oracles The distribution of the random oracle's output is uniformly random. We assume ideal hash functions in the proposed protocol, i.e., random oracle. In the random oracle model, the adversary obtains the functions' results by making a request to the random oracles. The oracle has a list of pairs of input and output called a hash-list. When a random oracle receives a query, if the query was recorded in the hash-list, (meaning the query was not previously requested) the random oracle outputs the value recorded in the hash-list. If the query was not recorded in the hash-list (meaning this is the first request of query), the random oracle outputs a random value and then records the input and output pair in the hash-list. Eq.(1) is a formalization of random oracles presented by Blanchet, et al. [9], where $hash : bitstring \rightarrow D$. OH is a formalization of an oracle that receives a query x, and outputs $hash(x)$, where the number of queries is at most qH. $A \stackrel{?}{=} B$ is a Boolean function that outputs "true" when $A = B$, and outputs "false" when $A \neq B$.

$$\begin{array}{l} \textbf{foreach}\ \ i_h \leq n_h\ \textbf{do}\ OH(x : bitstring) := \textbf{return}(hash(x))\ [all] \\ \approx_0\ \textbf{foreach}\ \ i_h \leq n_h\ \textbf{do}\ OH(x : bitstring) := \textbf{find}\ u \leq n_h\ \textbf{suchthat} \\ (\textbf{defined}(x[u], r[u]) \wedge x \stackrel{?}{=} x[u])\ \textbf{then return}(r[u]) \\ \textbf{else}\ r \stackrel{R}{\leftarrow} D; \textbf{return}(r) \end{array} \quad (1)$$

Next, we describe the random oracle OH using the following rule that the view of the output of a random oracle that receives input x and outputs $hash(x)$ (at most q_H times), is the same as that of a random oracle, that receives input x and, if there is $x = x[u]$ in its list, outputs $r[u]$, otherwise it chooses a random value r and outputs it. All inputs and chosen r are recorded as alignments in CryptoVerif. This means that if the oracle receives the i-th input x and randomly

chooses a value r uniformly, the value of x is assigned to $x[i]$, and the value of r is assigned to $r[i]$. In Eq.(1), a pair of $(x[i], r[i])$, of which index i is the same, means a pair of input and output of the random oracle. As above, functions of a random oracle are described as alignments of $x[]$ $r[]$. In the proposed protocol, the following three random oracles are used. $H_0 : key \times IDs \times index \times nonce \times nonce \rightarrow h_0$, $H_1 : key \times IDs \times index \times nonce \times nonce \rightarrow h_1$, $H_2 : key \times IDs \times index \rightarrow key$. OH_0, OH_1, OH_2 are processes that receive input and send output of a random oracle, where key is a group of session keys, IDs is a group of ID, $index$ is a group of indexes of session keys, and $nonce$ is a group of t-bit random values.

Formalization of attack games Attack games consist of three processes Server, Tag, and Challenge and random oracles H_0, H_1, H_2. The processes Server and Tag are a formalization of a server and tag in the proposed protocol. Challenge is a formalization of the attacker's target, and depends on security requirements and that is the verifier. The following are descriptions of the Server, Tag, and attack model. First we define functions $test1, test2, test3$. $test1 : bool \times nonce \times nonce \rightarrow nonce$, $test2 : bool \times h_0 \times h_0 \rightarrow h_0$, $test3 : bool \times h_1 \times h_1 \rightarrow h_1$. $test1(b, A, B)$ is a function for which if b is $true$, outputs A, if b is $false$, outputs B for any $A, B \in nonce$, In the function, the view of the process in which if input is $b : bool$, chooses $A, B \xleftarrow{R} nonce$ and outputs $test(b, A, B)$, is same as that of the process in which if input is $b : bool$, outputs $C \xleftarrow{R} nonce$. The property is formalized as Eq.(2). $test2$ and $test3$ are defined in the same way.

$$\begin{aligned} &\textbf{foreach}\ \ i_t \leq n_t\ \textbf{do}\ test1(b : bool) := A, B \xleftarrow{R} nonce; \textbf{return}(test1(b, A, B))[all] \\ &\approx_0 \textbf{foreach}\ \ i_t \leq n_t\ \textbf{do}\ test1(b : bool) := C \xleftarrow{R} nonce; \textbf{return}(C) \end{aligned} \tag{2}$$

Next, we formalize the processes of Server and Tag, as Eq.(3) and Eq. (4) respectively, where "yield" indicates a process of stopping and doing nothing.

$$\begin{aligned} &\textbf{foreach}\ \ i_p \leq n_p\ \textbf{do}\ Server := \textbf{input}();\ X_s \xleftarrow{R} nonce;\ \textbf{return}(X_s); \\ &\quad \textbf{input}(\alpha_s : nonce, \beta_s : h_0); \textbf{if}\ \beta_s \stackrel{?}{=} H_0(SK0, ID, i_0, X_s, \alpha_s)\ \textbf{then} \\ &\qquad \textbf{return}(H_1(SK0, ID, i_0, X_s, \alpha_s)) \\ &\quad \textbf{else}\ Z_{rnd} \xleftarrow{R} h_1; Z_s \leftarrow H_1(SK1, ID, i_1, X_s, \alpha_s); \\ &\quad b_s : bool \leftarrow (\beta_s \stackrel{?}{=} H_0(SK1, ID, i_1, X_s, \alpha_s));\ \textbf{return}(test3(b_s, Z_s, Z_{rnd})). \end{aligned} \tag{3}$$

$$\begin{aligned} &\textbf{foreach}\ \ i_p \leq n_p\ \textbf{do}\ Tag := \textbf{input}(X_t : nonce);\ \alpha_t \xleftarrow{R} nonce; \\ &\quad \beta_t \leftarrow H_0(SK1, ID, i_1, X_t, \alpha_t); \textbf{return}(\alpha_t, \beta_t); \\ &\quad \textbf{input}(Z_t : h1); \textbf{if}\ Z_t \stackrel{?}{=} H_1(SK1, ID, i_1, X_t, \alpha_t)\ \textbf{then}\ \mathsf{yield}. \end{aligned} \tag{4}$$

By using the above processes, an attack game is formalized as Eq.(5).

$$\begin{aligned} &OH_1 | OH_2 | OH_3 | (\textbf{input}(i_0 : index, i_1 : index, i_2 : index); ID \xleftarrow{R} IDs; \\ &seed \xleftarrow{R} key; SK0 \leftarrow H_2(seed, ID, i_0); SK1 \leftarrow H_2(SK0, ID, i_1); \\ &SK2 \leftarrow H_2(SK1, ID, i_2); \textbf{return}(ID, SK2) | (Server | Tag | Challenge)). \end{aligned} \tag{5}$$

Formalization of Client indistinguishability game *Challenge* on the Client indistinguishability game is formalized as Eq.(6).

$$\begin{array}{l} Challenge := \mathbf{input}(X^* : nonce);\ b^* \stackrel{R}{\leftarrow} \{true, false\}; \alpha^* \stackrel{R}{\leftarrow} nonce; \\ \beta^* \leftarrow H_0(SK1, ID, i_1, X^*, \alpha^*); \alpha_{rnd} \stackrel{R}{\leftarrow} nonce;\ \beta_{rnd} \stackrel{R}{\leftarrow} h_0; \\ \mathbf{return}(test1(b, \alpha^*, \alpha_{rnd}), test2(b, \beta^*, \beta_{rnd})); \end{array} \tag{6}$$

In the above attack game, if *secrecy* is shown when b is chosen in *Challenge*, in the proposed protocol, the adversary cannot distinguish between Tag's output and random values. In this case, we can say that the protocol satisfies the property of Client indistinguishability.

Formalization of Client unforgeability game *Challenge* on the Client unforgeability game is formalized as Eq.(7).

$$\begin{array}{l} Challenge := \mathbf{input}();\ X^* \stackrel{R}{\leftarrow} nonce; \mathbf{return}(X^*); \\ \quad \mathbf{input}(\alpha^* : nonce, \beta^* : nonce); \mathbf{if}\ \beta^* \stackrel{?}{=} H_0(SK0, ID, i_0, X^*, \alpha^*)\ \mathbf{then} \\ \quad\quad \mathbf{find}\ u \leq n_p\ \mathbf{suchthat}\ (\mathbf{defined}(X_t[u]) \wedge X^* \stackrel{?}{=} X_t[u])\ \mathbf{then}\ \mathsf{yield} \\ \quad\quad \mathbf{else}\ \text{event bad} \\ \quad \mathbf{else\ if}\ \beta^* \stackrel{?}{=} H_0(SK1, ID, i_1, X^*, \alpha^*)\ \mathbf{then} \\ \quad\quad \mathbf{find}\ u \leq n_p\ \mathbf{suchthat}\ (\mathbf{defined}(X_t[u]) \wedge X^* \stackrel{?}{=} X_t[u])\ \mathbf{then}\ \ \mathsf{yield} \\ \quad\quad \mathbf{else}\ \text{event bad}. \end{array} \tag{7}$$

The event "event bad" has occurred, only when α^*, β^* received by *Challenge* is accepted and X^* given to the adversary by *Challenge* have not been requested to the Tag oracle. This means that the adversary successfully forges the Tag's output only when "event bad" occurs. Tag unforgeability is satisfied if the probability of "event bad" is negligible.

Formalization of Server unforgeability game *Challenge* on Server unforgeability game is formalized as Eq.(8).

$$\begin{array}{l} Challenge := \mathbf{input}(X^* : nonce);\ \alpha^* \stackrel{R}{\leftarrow} nonce; \beta^* \leftarrow H_0(SK1, ID, i_1, X^*, \alpha^*); \\ \quad \mathbf{return}(\alpha^*, \beta^*); \\ \quad \mathbf{input}(Z^* : h_1); \mathbf{if}\ Z^* \stackrel{?}{=} H_1(SK1, ID, i_1, X^*, \alpha^*)\ \mathbf{then} \\ \quad\quad \mathbf{find}\ u \leq n_p\ \mathbf{suchthat}\ (\mathbf{defined}(\alpha_s[u], \beta_s[u]) \wedge \alpha^* \stackrel{?}{=} \alpha_s[u] \wedge \beta^* \stackrel{?}{=} \beta_s[u]) \\ \quad\quad \mathbf{then}\ \mathsf{yield}\ \mathbf{else}\ \text{event bad} \end{array} \tag{8}$$

The event "event bad" occurs, only when Z^* received by *Challenge* is accepted and, α^*, β^* given to the adversary by *Challenge* have not been requested to the Server oracle. This means that the adversary successfully forges server's output only when "event bad" occurs. Server unforgeability is satisfied if the probability of "event bad" is negligible.

4.3 Theorems

In section 2, we defined security notions to achieve not only basic but also extended properties. As a result, the following theorems can be proven by using CryptoVerif.

Theorem 1. Forward-secure Indistinguishability
The proposed scheme is Forward-secure Indistinguishable, if hash functions $H_0, H_1,$ *and* H_2 *are random oracles.*

Theorem 2. Forward-secure (Client)-Unforgeability
The proposed scheme is Forward-secure (Client)-Unforgeable if hash functions $H_0, H_1,$ *and* H_2 *are random oracles.*

Theorem 3. Forward-secure (Server)-Unforgeability
The proposed scheme is Forward-secure (Server)-Unforgeable if hash functions $H_0, H_1,$ *and* H_2 *are random oracles.*

4.4 Output of verification on CryptoVerif

Following are the result of CryptoVerif verification. We use a PC as follows, Intel(R) Core(TM)2 Duo CPU U9300 @ 1.20GHz, RAM 2.85GB. CryptoVerif is version 1.10pl1.

Client indistinguishability game result The adversary has an advantage in this game, which can be shown to be negligible by applying the following commands: auto, simplify, remove_assign all, auto, and transforming the game 34 times. Running time is about 45 seconds. The result is as follows.

RESULT Proved secrecy of b with probability $2.*n_{H0}/|key|+3.*n_{H1}/|key|+7.*n_{H2}/|key|+n_p*n_{H0}/|key|+3.*n_p/|h_0|+n_p*n_{H0}/|nonce|+4.*n_p*n_p/|nonce|+n_{H0}/|nonce|+5.*n_p/|nonce|+6.*1./|key|$

$|key|, |h_0|, |nonce|$[6] are exponent functions of the security parameter, n_p, n_{H_0}, n_{H_1}, n_{H_2} are polynomial functions of the security parameter. From the above results, we can say that the advantage of breaking the secrecy of b is negligible.

Client unforgeability game result The adversary has an advantage in this game, which can be shown to be negligible by applying the following commands: auto, simplify, auto, and transforming the game 14 times. Running time is about 40 seconds. The result is as follows.

RESULT Proved event bad ==> $false$ with probability$3.*n_{H0}/|key|+3.*n_{H1}/|key|+7.*n_{H2}/|key|+n_p*n_{H0}/|key|+3.*n_p/|h_0|+n_p*n_{H0}/|nonce|+3.*n_p/|nonce|+4.*n_p*n_p/|nonce|+2.*1./|h_0|+6.*1./|key|$

$|key|, |h_0|, |nonce|$ are exponent functions of the security parameter, n_p, n_{H_0}, n_{H_1}, n_{H_2} are polynomial functions of the security parameter. From the above results, we can say that the probability of "event bad" is negligible.

Server unforgeability game result The adversary has an advantage in this game, which can be shown to be negligible by applying the following commands: auto and transforming the game 13 times. Running time is about 15 seconds. The result as the follows.

[6] $|A|$ is an element number of a grou A.

RESULT Proved event bad $==> false$ with probability $2.*n_{H0}/|key|+2.*n_p*n_{H0}/|key|+n_{H1}/|key|+7.*n_{H2}/|key|+n_p/|h_0|+n_p*n_{H0}/|nonce|+2.*n_p*n_p/|nonce|+n_{H0}/|nonce|+4.*n_p/|nonce|+1./|h_1|+6.*1./|key|$

$|key|, |h_0|, |h_1|, |nonce|$ are exponent functions of the security parameter, n_p, $n_{H_0}, n_{H_1}, n_{H_2}$ are polynomial functions of the security parameter. As above results, we can say that the probability of "event bad" is negligible.

4.5 Proof Sketch

If there is an adversary $\mathcal{A}$, that can break the proposed protocol, there is an adversary $\mathcal{B}$ that breaks the knocked-down one shown in the above session. We can show that the proposed protocol is secure by showing the above proof. $\mathcal{A}$ is allowed to access server oracle $\mathcal{S}_A$ at most n times, and is also allowed to access tag oracle $\mathcal{T}_A$ at most n times, and in total is allowed to access the random oracles at most q_H times. Mutual authentications between tag and server succeed at most n times; therefore the variations of session keys are at most n. $\mathcal{B}$ is allowed to access the server oracle $\mathcal{S}_B$ at most n times, and also is allowed to access the tag oracle $\mathcal{T}_B$ at most n times, and totally is allowed to access the random oracles at most q_H times. We first show how to construct with $\mathcal{B}$. Fig.2

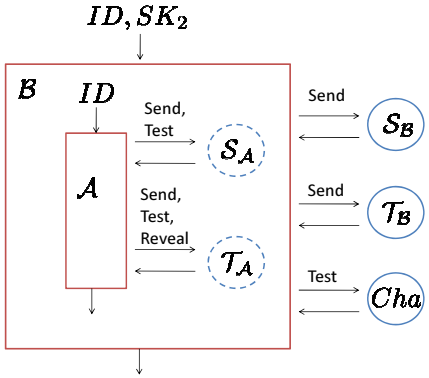

Fig. 2. Rough construction of $\mathcal{B}$ by using $\mathcal{A}$.

is a rough construction of $\mathcal{B}$ by using $\mathcal{A}$. $\mathcal{B}$ executes the following, when given target ID and the next session key $SK2$ of the target session, $\mathcal{B}$ guesses $i \stackrel{R}{\leftarrow} [1, n]$ that is the number of the session key that $\mathcal{A}$ uses for its attack, sets the session key as $SK_{i+1} \leftarrow SK2$, and then sends a query to random oracle H_2 with the $SK_{i+1} \leftarrow SK2$ and acquires $SK_{i+2}, \ldots SK_n$. Next, $\mathcal{B}$ gives ID to $\mathcal{A}$ and runs $\mathcal{A}$. $\mathcal{B}$ simulates $\mathcal{S}_A^{sid}$ and $\mathcal{T}_A^{sid}$ as follows, letting $sid = j||ssid$. $\mathcal{B}$ transfers all queries to the random oracles from $\mathcal{A}$ to the random oracles for $\mathcal{B}$, and receives output from the random oracles for $\mathcal{B}$ and transfers the value to $\mathcal{A}$.

- If $j < i$,
  - $\mathcal{S}_A^{sid}$: If he
    - receives (), chooses $X_{sid} \stackrel{R}{\leftarrow} \{0,1\}^t$, then out uts X_{sid}.
    - receives $\alpha_{sid}, \beta_{sid}$, chooses $Z_{sid} \stackrel{R}{\leftarrow} h_1$, then out uts Z_{sid}.
    - receives a *Test* query, sto the simulation.
  - $\mathcal{T}_A^{sid}$: If he
    - receives X'_{sid}, chooses $\alpha_{sid} \stackrel{R}{\leftarrow} \{0,1\}^t, \beta_{sid} \stackrel{R}{\leftarrow} h_0$, then out uts $(\alpha_{sid}, \beta_{sid})$.
    - receives Z'_{sid}, out uts ().
    - receives *Revealed* queries, sto the simulation.
    - receives a *Test* query, sto the simulation.
- If $j = i$,
  - $\mathcal{S}_A^{sid}$: If he
    - receives (), send () to $\mathcal{S}_B$ and receivesX_{sid}, then out uts X_{sid}.
    - receives $\alpha_{sid}, \beta_{sid}$, send $\alpha_{sid}, \beta_{sid}$ to $\mathcal{S}_B$ and receives Z_{sid}, then out uts Z_{sid}.
    - receives a *Test* query, transfer the query to *Challenge*, and receives a value, then out uts the received value.
  - $\mathcal{T}_A^{sid}$: If he
    - receives X'_{sid}, send X'_{sid} to $\mathcal{T}_B$, receives $(\alpha_{sid}, \beta_{sid})$, then out uts $(\alpha_{sid}, \beta_{sid})$.
    - receives Z'_{sid}, send Z'_{sid} to $\mathcal{T}_B$, receives (), then out uts the ().
    - receives a *Revealed* query, sto the simulation.
    - receives a *Test* query, transfer the query to *Challenge* oracle, and receives a value, then out uts the value.
- If $j > i$,
  - $\mathcal{S}_A^{sid}$:If he
    - receives (), chooses $X_{sid} \stackrel{R}{\leftarrow} \{0,1\}^t$, then out uts X_{sid}.
    - receives $\alpha_{sid}, \beta_{sid}$, calculate Z_{sid} using SK_j along the rotocol, and oututs the calculated result.
    - receives a *Test* query, sto the simulation.
  - $\mathcal{T}_A^{sid}$: If he
    - receives X'_{sid}, calculate $(\alpha_{sid}, \beta_{sid})$ using SK_j along the rotocol, then out uts the calculated $(\alpha_{sid}, \beta_{sid})$.
    - receives Z'_{sid}, executes using SK_j along the rotocol, then out uts ().
    - receives a *Revealed* query, out uts SK_j.
    - receives a *Test* query, sto the simulation.

If the guessed i is correct, $\mathcal{B}$ can set the target problem as a problem of $\mathcal{A}$'s target. Therefore if there exists an adversary $\mathcal{A}$ that successfully attacks with non-negligible probability, there exists an adversary $\mathcal{B}$ that successfully attacks with non-negligible probability at least for the guessed i. (This is because if the success probability of adversary $\mathcal{B}$ at the guessed i is negligible, the success probability of adversary $\mathcal{A}$ is also negligible. It is $1/n$ at least that the probability that the i guessed by $\mathcal{B}$ is correct. Since the upper bound of queries, n, is a polynomial function of the security parameter, the above proof means that if there is an adversary $\mathcal{A}$, there is an adversary $\mathcal{B}$ that can succeed with non-negligible probability.

4.6 Note

The proposed protocol satisfies resiliency of desynchronization (Def.4) against an adversary that can only control the communicated data, because the proposed protocol satisfies the mutual authenticity. If the adversary can control not only the communicated data but also the number of steps that tag executes, the proposed protocol cannot satisfy the property of resiliency of desynchronization. If, however, the proposed protocol can satisfy this property by modifying it as follows, i.e., when the server receives Y from a tag, check the following formula with $sk_{\mathsf{ID},i'}$ and $i'-1$,

$\beta' \stackrel{?}{=} H_0(sk_{\mathsf{ID},i'}, \mathsf{ID}, i'-1, X, \alpha')$

as well as the verifications in the original proposed protocol. By this modification, the server can detect the desynchronization of a session ID i. Proof against such an adversary as that above is a topic for future work.

5 Conclusion

This paper, proposed a formal security model and definitions for an RFID authentication protocol that is robust against a man-in-the-middle adversary. An RFID authentication protocol from the OSK protocol and synchronization mechanism was then put forth. The security of the protocol was proved by combining a handwritten proof and CryptoVerif results as well as by using a manual proof method in a game-based approach.

References

1. RFID security & rivacy lounge. htt ://www.avoine.net/rfid/.
2. Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *POPL*, ages 104–115, 2001.
3. Martín Abadi and Philli Rogaway. Reconciling two views of cry togra hy (the com utational soundness of formal encry tion). In *IFIP TCS*, ages 3–22, 2000.
4. Myrto Ara inis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the a lied i calculus. In *CSF*, ages 107–121, 2010.
5. Bruno Blanchet. Cry toVerif com utationally sound, automatic cry togra hic rotocol verifier user manual. `http://www.cryptoverif.ens.fr/`.
6. Bruno Blanchet. An Efficient Cry togra hic Protocol Verifier Based on Prolog Rules. In *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, ages 82–96, Ca e Breton, Nova Scotia, Canada, June 2001. IEEE Com uter Society.
7. Bruno Blanchet. A com utationally sound mechanized rover for security rotocols. Cry tology ePrint Archive, Re ort 2005/401, 2005.
8. Bruno Blanchet. A com utationally sound mechanized rover for security rotocols. In *IEEE Symposium on Security and Privacy*, ages 140–154, Oakland, California, May 2006.
9. Bruno Blanchet and David Pointcheval. Automated security roofs with sequences of games. In *CRYPTO*, ages 537–554, 2006.
10. Mayla Brusò, Konstantinos Chatzikokolakis, and Jerry den Hartog. Formal verification of rivacy for RFID systems. In *CSF*, ages 75–88, 2010.

11. Gerhard Hancke and Markus Kuhn. An RFID Distance Bounding Protocol. In *Conference on Security and Privacy for Emerging Areas in Communication Networks – SecureComm 2005*, ages 67–73, Athens, Greece, Se tember 2005. IEEE, IEEE Com uter Society.
12. Ari Juels. RFID security and rivacy: A research survey. Manuscri t, Se tember 2005.
13. Ari Juels and Ste hen Weis. Defining strong rivacy for RFID. Cry tology ePrint Archive, Re ort 2006/137, 2006.
14. Ari Juels and Ste hen Weis. Defining Strong Privacy for RFID. In *International Conference on Pervasive Computing and Communications – PerCom 2007*, ages 342–347, New York City, New York, USA, March 2007. IEEE, IEEE Com uter Society Press.
15. Shin'ichiro Matsuo, Kunihiko Miyazaki, Akira Otsuka, and David A. Basin. How to evaluate the security of real-life cry togra hic rotocols? - the cases of ISO/IEC 29128 and CRYPTREC. In *Financial Cryptography and Data Security, FC 2010 Workshops, RLCPS, WECSR, and WLC 2010, Tenerife, Canary Islands, Spain,*, volume 6054 of *Lecture Notes in Computer Science*, ages 182–194. S ringer, January 2010.
16. Miyako Ohkubo, Shin'ichiro Matsuo, Yoshikazu Hanatani, Kazuo Sakiyama, and Kazuo Ohta. Robust RFID authentication rotocol with formal roof and its feasibility. Cry tology ePrint Archive, Re ort 2010/345, 2010.
17. Miyako Ohkubo, Koutarou Suzuki, and Shingo Kinoshita. Cry togra hic a roach to " rivacy-friendly" tags. In *RFID Privacy Workshop*, MIT, MA, USA, November 2003.
18. Paise Radu-Ioan and Serge Vaudenay. Mutual Authentication in RFID: Security and Privacy. In *Proceedings of the 3rd ACM Symposium on Information, Computer and Communications Security – ASIACCS'08*, ages 292–299, Tokyo, Ja an, 2008. ACM Press.
19. Serge Vaudenay. On Privacy Models for RFID. In *Advances in Cryptology - Asiacrypt 2007*, volume 4833 of *Lecture Notes in Computer Science*, ages 68–87, Kuching, Malaysia, December 2007. S ringer-Verlag.