Multifactor Authenticated Key Renewal

Shin'ichiro Matsuo[1], Daisuke Moriyama[1], and Moti Yung[2,3]

[1] National Institute of Information and Communications Technology (NICT), Japan
[2] Columbia University
[3] Google Inc.

Abstract. Establishing secure channels is one of the most important and fundamental trust issues in information security. It is of high important not only for servers and users computers but also for global connectivity among any kind of network devices. Most existing technologies for establishing secure channels are based on asymmetric cryptography which requires heavy computations, large memory and complicated supporting mechanism such as PKI. In this paper, we consider the setting of authentication with small devices possibly held by humans and possibly embedded in a semi secure environment. We propose a authenticated key renewal protocol which uses only symmetric cryptography. The protocol takes into account other factors important for embedded and human held network devices: It covers multi-factor authentication to take advantage of secrets possessed by the secure device as well as the memorable password of the device owner. The protocol can, further, allow partial leakage of stored secret from a secure device. The protocol's considerations are a good demonstration of designing "trusted procedure" in the highly constrained environment of mobile and embedded small devices which is expected to be prevalent in the coming years.

Keywords: Key exchange, Multi-factor authentication, and Leakage resilience.

1 Introduction

1.1 Background

Staring with Diffie-Hellman key exchange [4], establishing secure channels is one of the most fundamental issues in information security. Establishing such a channel among communicating entities, has two important trust requirements: authentication and session key secrecy. That is, when the session key is shared the counterpart of sharing must be correctly authenticated and the shared session encryption/decryption key must be kept secret against unauthorized entities.

Many types of authenticated key sharing protocol exist, for example SSL/TLS [5], IPSec [10] and SSH [18], are proposed and used in today's Internet. Most of these protocols are based on asymmetric key cryptography, because their basic mechanism of key sharing is based on the DH key exchange protocol. We consider here the setting of small low-power devices such as smartphones,

L. Chen, M. Yung, and L. Zhu (Eds.): INTRUST 2011, LNCS 7222, pp. 204–220, 2012.

sensors, and RFID-tags. When considering this setting, computing modular exponentiation is costly. Moreover, when we use asymmetric key cryptography for authentication, we need PKI, which is costly as well, and may not be accessible at a device level (i.e., CA identity not accessible for establishing root of trust). It is much easier to initiate a device (at manufacturing) with symmetric key capability.

Currently, the typical computing environment is a network with many mobile devices (such as iPad and Android devices). Whereas trusted module like TPM chips ensure many security functionalities required in computing devices in regular computers [17], in our setting it is hard to assume the devices will have modular exponentiation capability (even if some have, we want interoperability), and it is interesting to consider what is doable based on symmetric cryptography alone. Furthermore, in embedding devices the authentication is based on possession of a secure token. This means the authentication (and the key exchange) is based on ownership of the device. Then the authentication is slightly different from "entity" authentication (the entity may be embedded in another entity, and may be possessed by a human operator/ owner). In this direction, PAKE protocol realizes authentication function by using password, which is based on human memory and tightly related to the entity itself [3,8]. However in PAKE, the security parameter is limited by human capability. Long password is not suitable for PAKE, therefore it is not strong in cryptographic sense. To increase the accuracy of the authentication, it is good desirable to include multifactor into the authentication protocol. Examples of the factors are memory, ownership, biometric, device keys, etc. Also, small devices may operate in an environment which is leaky and this has to be taken into account as well.

1.2 Contribution of This Paper

In this paper, we propose a system model and a protocol for multifactor authenticated key establishment, named "Multifactor Authenticated Key Renewal (MAKeR) protocol". It aims to establish a random session key using multi-factor information.

MAKeR protocol uses symmetric cryptography only to realize authenticated key sharing, that is, uses only hash/pseudorandom function and pseudorandom generator. Though this protocol is not secure against online dictionary attack due to the limitation of symmetric cryptography techniques, it still present the best properties achievable under symmetric cryptography only conditions. Thus, this research goal is implementing better security mechanisms to low-power devices such as the smartphones, smart-cards, sensors, and RFID-tags.

Our proposal includes shared secret stored in trusted device (the device key), and password or other information as authentication factors. The former helps realization of both authentication factor of "what one possesses" and "secrecy" of session key. The latter is used for authentication by "human-related information" ("what one knows").

1.3 Related Works

As mentioned above, key sharing (exchange) protocols are extensively researched and many secure protocols have been proposed. Basic type of authenticated key exchange protocol is combination of entity authentication and Diffie-Hellman like key exchange protocol. Entity authentication protocols consist of asymmetric key based protocol and symmetric cryptography based protocol. However, Diffie-Hellman key exchange protocol needs asymmetric cryptographic calculation. As a user-friendly authenticated key exchange protocol, "Password-based Authenticated Key Exchange (PAKE)" protocols are widely studied. This protocol uses pre-shard (short) password as a factor of authentication. Most PAKE protocols include data from password into Diffie-Hellmann key exchange protocol, they still need asymmetric cryptographic calculation.

For low-power devices, many entity authentication protocol for RFID tags are proposed [1,11,2,12,16]. Most of these protocols are based on symmetric cryptography such as block cipher, stream cipher, hash function and so on. Our proposal is based on the existing authentication protocol for RFID-tags.

The other works related to this paper are about multi-factor authentication. Klesnikov and Rackoff proposed a multi-factor authenticated key exchange protocol [9]. In this paper, they use three factors: password, secret keys for symmetric key cryptography and asymmetric key cryptography. Pointcheval et al. proposed security model and authenticated key exchange protocol which uses biometrics as well as secret information as authentication factors [15]. However, the protocol is based on asymmetric cryptography.

This paper also deals with leakage of secret information from a device, which is realized by side-channel attacks or cold-boot attacks. This is a very current area of research. In ordinary cryptographic research, the security model does not consider leakage of secret information. In the symmetric world, Petit et al. [14] proposed a leakage-resilient pseudo-random generator from ideal ciphers. Dziembowski et al. [6] proposed a leakage-resilient stream cipher based on pseudo-random generator in the standard model. Then Pietrzak [13] proposed simplified leakage-resilient stream cipher from wPRF. We will use the same model of leakage as [6,13] in this paper.

2 System Model

The MAKeR protocol consists of two entity, Alice and Bob. Alice can be treated as client of some service. Alice has one device $\mathtt{DEV}_\mathtt{A}$ connected to the Internet. An application program $\mathtt{PROG}_\mathtt{A}$ runs on the device and accepts multiple inputs from Alice such as password and other authentication factors. The device also havs some trusted module/ token $\mathtt{TPM}_\mathtt{A}$ attached to it. $\mathtt{TPM}_\mathtt{A}$ stores secret information $\mathtt{sec}_\mathtt{A}$, and conducts some (symmetric) cryptographic operations. $\mathtt{TPM}_\mathtt{A}$ may be implemented into $\mathtt{DEV}_\mathtt{A}$ (by the manufacturer, say) or is attached using a card slot and so on. $\mathtt{TPM}_\mathtt{A}$ communicates with $\mathtt{PROG}_\mathtt{A}$.

Bob can be treated as a server. It is realized as a device $\mathtt{DEV}_\mathtt{B}$ and also connected to the Internet. An application program $\mathtt{PROG}_\mathtt{B}$ executes authenticated

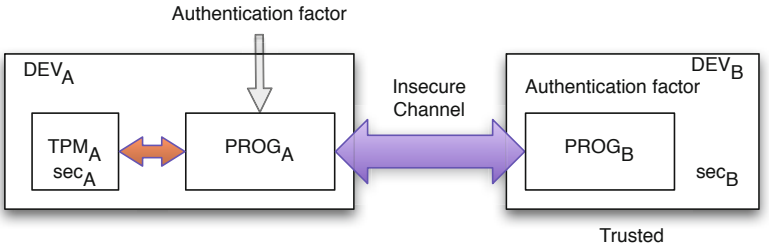

Fig. 1. System model of multi-factor authenticated key renewal

key renewal protocol. Some secret information $\mathtt{sec}_B$ and other multiple authentication factors shared with each client are stored in $\mathtt{DEV}_B$, and $\mathtt{PROG}_B$ accesses them in the execution of the protocol. We assume that $\mathtt{DEV}_B$ and $\mathtt{PROG}_B$ is trusted. That is, the malicious adversary $\mathcal{A}$ against this protocol cannot corrupt $\mathtt{DEV}_B$ and $\mathtt{PROG}_B$. Thus, $\mathcal{A}$ cannot obtain $\mathtt{sec}_B$ and any internal state of $\mathtt{DEV}_B$ and $\mathtt{PROG}_B$.

$\mathtt{DEV}_A$ and $\mathtt{DEV}_B$ communicates over the Internet, which can be treated as insecure channel. That is $\mathcal{A}$ can eavesdrop, alter, intercept all communication data and send arbitrary message to Alice and Bob.

When Alice would like to establish a secure channel with Bob, she inputs required authentication information (which is shared) to $\mathtt{PROG}_A$ through user interface of $\mathtt{DEV}_A$. In the protocol execution, $\mathtt{PROG}_A$ calculates protocol messages with help of $\mathtt{TPM}_A$. $\mathtt{PROG}_A$ and $\mathtt{PROG}_B$ execute session key renewal protocol, then share the session key $\mathtt{key}$ at the end of the protocol.

This system model is shown in Fig.1.

3 Security Model

The goal of the protocol is establishing a session key from multi-factor secrets. The basic security for multi-factor key renewal protocol is that even if a malicious adversary $\mathcal{A}$ interacts with clients and the server, the communication messages leak no information about the session key computed by the server. Additionally, the protocol must guarantee secrecy of the future session key, and privacy for the past session key even if an attack is attempted (namely, it must be based on fresh randomness and be forward secure).

The device $\mathtt{DEV}_A$ stores secret value $\mathtt{sec}_A$. We firstly consider the basic case that the device $\mathtt{DEV}_A$ as tamper-free, next we consider the several types of secret information leakage.

The formal security definition in the basic case (without any key leakage from $\mathtt{DEV}_A$) is as follows.

3.1 Security Definition without Corruption

The basic security model is similar to that of multi-factor authentication proposed by Pointcheval et al. [15].

Participants, sessions and partnering. Let $\mathtt{DEV_A}$ be a client device of entity `Alice` to be authenticated and $\mathtt{DEV_B}$ be a (trusted) server organized entity `B`. We consider that the server and every client can initiate several instances at a time, in order to run several sessions concurrently. The i-th instance of the entity U, where U is a client or the server, is denoted as Π_U^i. This instance includes three variables:

- pid_U^i: the partner identifier which is the instance with whom Π_U^i believes it is interacting,
- sid_U^i: the unique session identifier, in practice it can be the transcript seen by Π_U^i (concatenation of the received/sent flows, excepted the last one).
- acc_U^i: a boolean variable which is determined at the end of the session and denotes whether the instance Π_U^i goes in an accepted ($a_U^i = 1$) state or not ($acc_U^i = 0$).

The two instances Π_U^i and $\Pi_{U'}^j$ are said to be partners if the following conditions are fulfilled:

1. $pid_U^i = \Pi_{U'}^j$ and $pid_{U'}^j = \Pi_U^i$;
2. $sid_U^i = sid_{U'}^j \neq null$;
3. $acc_U^i = acc_{U'}^j = 1$.

Adversarial capabilities and goals. The semantic security of the key is modeled using the Real-or-Random paradigm. At the beginning of the game, the challenger chooses a random bit b which determines its behavior when answering `Test`-query during the game (it provides either real session key or random to the adversary). The adversary may interact with protocol instances through several oracles, and at the end of the game, she outputs a bit b. If $b = b'$, she wins, otherwise, she looses. The available queries are as follows:

- $\mathtt{Send}(m, \Pi_U^i)$: this query allows the adversary to play with the instances, by intercepting, forwarding, modifying or creating messages. The output of this query is the answer generated by instance Π_U^i to the message m.
- $\mathtt{Reveal}(\Pi_U^i)$: this query models the leakage of information about the session key agreed on by the parties. For example, if it is misused afterward. Therefore, if no session key is defined for this instance, or if the instance (or its partner) has been tested (see below), then the output is $\perp$. Otherwise, the oracle outputs the session key computed by the instance Π_U^i.

To model the semantic security with respect to client authentication formally, the adversary can ask `Test`-query, but to the server S only: we are interested in the privacy of the key established with the real server only. We only consider the adversary whose goal is to impersonate a client to the server. Of course, to achieve this goal, the adversary may try to impersonate the server to the client in order to learn some information about the internal state $\mathtt{sec_A}$ of $\mathtt{TPM_A}$ or other multiple authentication factors. But only a client impersonation will be considered as a successful attack:

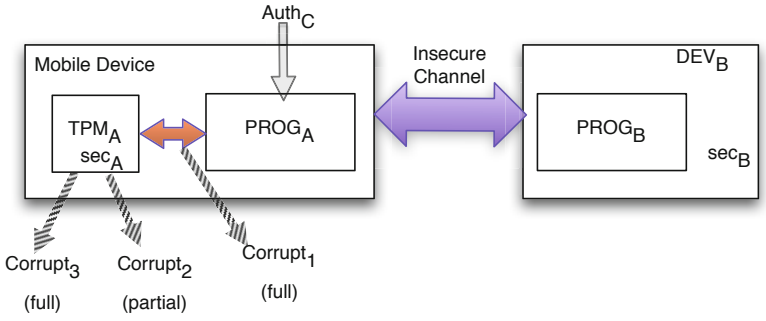

Fig. 2. Corruption model of multi-factor authenticated key renewal

– $\texttt{Test}(\Pi_S^i)$: The oracle responds
  - the session key of instance (Π_S^i) (that is $Reveal(\Pi_S^i)$), if $b = 1$ - the real case;
  - a random key from the same domain, if $b = 0$ - the random case.

Semantic Security. Let denote by $Succ$ the event that the adversary $\mathcal{A}$ correctly guesses the bit b used by the challenger during the above attack game. We require that the test session where adversary issues the test query must be fresh (see below). The *maker-advantage* $adv_P^{maker}(\mathcal{A})$ and the advantage function of the protocol P are respectively:

$$adv_P^{maker}(\mathcal{A}) = |2 \cdot Pr[Succ] - 1|, adv_P^{maker}(t, Q) = \max_{\mathcal{A}}\{adv_P^{maker}(\mathcal{A})\}$$

where the maximum is over all the attackers with time-complexity at most t and number of queries at most Q.

Client authentication. We also usually model an attack against the unilateral authentication of the client to the server by considering sessions where the server accepts, but without any client-partner. Let denote by $Succ$ the event that a server instance accepts with no partner instance of the client (with the same partial transcript).

The *auth-success* $Succ_P^{auth}(\mathcal{A})$ and the success function of the protocol P are respectively:

$$Succ_P^{auth}(\mathcal{A}) = Pr[Succ], Succ_P^{auth}(t, Q) = \max_{\mathcal{A}}\{Succ_P^{auth}(\mathcal{A})\}$$

where the maximum is over all the attackers with time-complexity at most t and number of queries at most Q.

3.2 Considering Corruption of Client Device

Here, we think about corruption of client device $\texttt{DEV}_\texttt{A}$. We consider the following three types of attacks; (1) communication channel between $\texttt{TPM}_\texttt{A}$ and $\texttt{PROG}_\texttt{A}$ and

(2) $\mathtt{TPM}_\mathtt{A}$ itself. These attack on $\mathtt{TPM}_\mathtt{A}$ can be categorized in two types; (2-1) partial leakage of $\mathtt{sec}_\mathtt{A}$ and (2-2) full leakage of $\mathtt{sec}_\mathtt{A}$. These attacks are shown in fig. 2.

For corruption, we must consider two new security notions, *freshness* and *forward secrecy.*

Freshness. The freshness notion basically defines session keys that are not trivially known to the adversary. Since we will focus on the freshness of the server only, we say that the session key of instance Π_S^i is fresh if:

- upon acceptance, $\mathtt{DEV}_\mathtt{A}$ (corresponding to the partner of Π_S^i) was not fully corrupted.
- no Reveal-query is sent to either Π_S^i or its partner.

Backward and forward-Secrecy. Backward-secrecy mean that after the time of corruption, the session keys in any following session remains secret against the adversary. In order to capture this security, the model must allow the adversary to perform `Test`-queries, which we will define after, on sessions occurred after the corruption. Forward-secrecy means that as soon as a session key is securely generated (semantically secure), it will remain secure even after corruption. In order to capture this security, the model must allow the adversary to perform `Test`-queries, on sessions completed before the corruption.

From here, we consider the cases which the adversary obtains leaked information. Note that in the following, we will restrict to non-adaptive corruptions: no corruption can be performed during a session, but before a new session starts.

Full Leakage in Communication Channel between $\mathtt{TPM}_\mathtt{A}$ and $\mathtt{PROG}_\mathtt{A}$. This leakage models interception of full information between $\mathtt{TMP}_\mathtt{A}$ and $\mathtt{PROG}_\mathtt{A}$. This type of attack is most easiest among three types of corruption, because this channel is not tamper-resistant in general. Moreover the $\mathtt{TPM}_\mathtt{A}$ is a device which may be attached to $\mathtt{DEV}_\mathtt{A}$, the interception is quite easy. The adversary can acquire all communication data between $\mathtt{TPM}_\mathtt{A}$ and $\mathtt{PROG}_\mathtt{A}$ by this attack.

To model this attack, we introduce $\mathtt{CORRUPT}_1$ oracle.

$\mathtt{CORRUPT}_1(\mathtt{DEV}_\mathtt{A})$**:** Upon asking this query, the adversary can acquire all communication between $\mathtt{TPM}_\mathtt{A}$ and $\mathtt{PROG}_\mathtt{A}$.

Partial Leakage of Internal Key. This attack is difficult because it need much expertise to do it. However, recent extensive researches on side-channel attacks, such attacks help the adversary to obtain internal secret information against tamper resistant mechanisms of $\mathtt{TPM}_\mathtt{A}$ itself.

To model this attack, we introduce following $\mathtt{CORRUPT}_2$ oracle.

$\mathtt{CORRUPT}_2(\mathtt{DEV}_\mathtt{A}, \lambda)$**:** Upon asking this query, the adversary can acquire partial information of secret $\mathtt{sec}_\mathtt{A}$ stored in $\mathtt{TPM}_\mathtt{A}$. The output of this oracle is $\lambda(\mathtt{sec}_\mathtt{A})$, where $\lambda(\cdot)$ is a leakage function which models this partial leakage.

Full Leakage of Internal Key. Here we consider full corruption of internal secret $\mathtt{sec}_{\mathtt{A}}$ of $\mathtt{TPM}_{\mathtt{A}}$. To conduct this attack, it takes much time. However, it is considered in the existing researches on key sharing protocols.

To model this attack, we introduce following $\mathtt{CORRUPT}_3$ oracle.

$\mathtt{CORRUPT}_3(\mathtt{DEV}_{\mathtt{A}})$**:** Upon asking this query, the adversary can acquire $\mathtt{sec}_{\mathtt{A}}$ stored in $\mathtt{TPM}_{\mathtt{A}}$.

Why Consider Both Partial and Full Leakage from TPM? As described above, in forward secrecy, we allow the adversary to obtain the full internal state of the device, denoted as "full leakage of internal key". Surely, we must consider this type of attack as the worst case. It is worth noting that, in the real usage of mobile devices, it apparently takes quite much time and effort to conduct attacks leading to full leakage of internal key (For example, the adversary steals the tag and brings it to his laboratory to obtain the internal state.) However, the adversary certainly has no chance to give the device to the original owner again.

In this paper, we additionally consider an attack scenario which we call "multi-time partial leakage". Namely, in the life time of a device, its internal state may be partially leaked in a gradual way. It is obvious that partial leakage is more likely to occur than full leakage, because the adversary can conduct such attacks in a shorter time, with cheap and small-size devices. Furthermore, the adversary has enough time to bring back the TPM/device to the original owner. Therefore, it is practical to consider the multi-time partial-leakage scenarios.

The partial leakage allows the adversary to conduct further key renewal. However, when full leakage occurred, the original user notice the attack and he could revoke the mobile device for key renewal. Thus, we consider only partial leakage for key indistinguishability against random and consider full leakage for forward secrecy.

4 Protocol Description

4.1 Basic Protocol

At first, we show the basic protocol which is secure against an adversary without corruption. Next, we will show the protocol which is secure against all types of corruptions.

The proposed protocol is combination of lightweight authentication scheme studied for RFID-tag and PRF (Pseudo Random Function).

Both the client (the person) $\mathtt{DEV}_{\mathtt{A}}$ and the server $\mathtt{DEV}_{\mathtt{B}}$ have fixed secret information for authentication. They consist of password, and so on. We represent it as $Auth_C = (pw_A, \ldots)$. $\mathtt{TPM}_{\mathtt{A}}$ also has a pseudo-random generator implemented in $\mathtt{TPM}_{\mathtt{A}}$. It outputs a tuple of pseudorandom value $(k_i, k_i^{'}, k_i^{''}, k_i^{'''})$ for i-th authenticated key renewal.

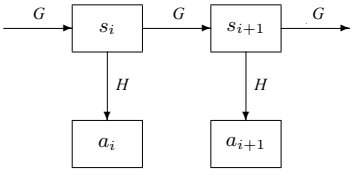

Fig. 3. Hash chain of OSK protocol

In the basic protocol, the pseudo-random generator is constructed from a kind of hash chain. OSK protocol [12], which is one of the most popular RFID authentication protocol, uses the hash chain. This hash chain is shown in Fig. 3.

In the basic protocol, $a_i = (k_i, k_i^{'}, k_i^{''}, k_i^{'''})$ are computed as three consecutive outputs of OSK hash chain. That is,

$$k_i = S_{4i-3}, k_i^{'} = S_{4i-2}, k_i^{''} = S_{4i-1}, k_i^{'''} = S_{4i}$$

The server $\mathtt{DEV_B}$ also has the same pseudo-random generator, which outputs same tuple of pseudorandom value for each authenticated key renewal.

The authenticated key renewal consists of authentication part and key establishment part.

Authentication Part

Step1. Client $\mathtt{DEV_A}$ generates a random value r_A. Then $\mathtt{DEV_A}$ sends r_A and identity A to the server $\mathtt{DEV_B}$.

Step2. The server $\mathtt{DEV_B}$ calculates message authentication code as follows:

$$Auth_B = MAC_{k_i^{'}}(F_{k_i}(r_A||r_B) \oplus Auth_C),$$

where $MAC_k(m)$ is a message authentication code of message m using key k, $F_K(\cdot)$ is pseudo random function with key K (for example, a block cipher like AES), $F_K(\cdot)$ is a message authentication code with key K and $||$ represents concatenation of two data. Then, $\mathtt{DEV_B}$ generates a random number r_B and sends $Auth_B, r_B, r_A$ and identity B to $\mathtt{DEV_A}$.

Step3. The client $\mathtt{DEV_A}$ verifies the message authentication code as follows:

$$Auth_B \stackrel{?}{=} MAC.Verify_{k_i^{'}}((F_{k_i}(r_A||r_B) \oplus Auth_C))$$

where $MAC.Verify_k(m)$ is verification algorithm of message authentication code of message m using key k. Then, $\mathtt{DEV_A}$ calculates message authentication code as follows:

$$Auth_A = MAC_{k_i^{''}}(F_{k_i}(r_B) \oplus Auth_C),$$

and send $Auth_A$ to the server.

Step4. The server $\mathtt{DEV_B}$ verifies the message authentication code as follows:

$$Auth_A \stackrel{?}{=} MAC.Verify_{k_i^{''}}((F_{k_i}(r_B) \oplus Auth_C)$$

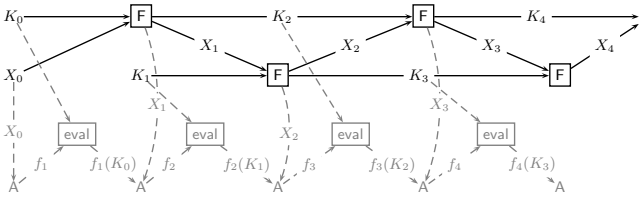

Fig. 4. Leakage resilient pseudorandom generator by Pietrzak et al.

Key Establishment Part: Both the server S and the client C calculates the session key as follows.

$$key_i = F_{k_i'''}(r_A) \oplus F_{k_i'''}(r_B) \oplus Auth_C$$

4.2 Protocol with Leakage Resilience

Next we move to the protocol with leakage resilience. To make the protocol with leakage resilience, we use Pietrzak's pseudorandom generator [13] for the physical pseudorandom generator. This pseudorandom generator has two sequences of random internal states and outputs one random value a_i for each authenticated key exchange. The Pietrzak's pseudorandom generator is shown in Fig.4.

As same as basic protocol we use three consecutive outputs of this pseudorandom generator. That is

$$a_i = (k_i = X_{5i-4}, k_i^{'} = X_{5i-3}, k_i^{''} = X_{5i-2}, k_i^{'''} = X_{5i-1}),$$

which are derived from $(K_{5i-5}, K_{5i-4}, X_{5i-5})$. a_i is transmitted to the $\mathtt{PROG_A}$, then the same protocol as the basic protocol is executed.

5 Security Evaluation

5.1 Completeness

Theorem 1 (Completeness). *The* $\mathtt{DEV_A}$ *and* $\mathtt{DEV_B}$ *who have same internal secret and* $Auth_C$ *can authenticate each other and compute the same session key.*

Proof. $\mathtt{DEV_A}$ can confirm the correctness of $\mathtt{DEV_B}$ in the step 3, because she knows $k_i, k_i^{'}, r_A, Auth_C$. Similarly, $\mathtt{DEV_B}$ can confirm the correctness of $\mathtt{DEV_A}$ in the step 4, because she knows $k_i, k_i^{''}, r_B, Auth_C$. Both $\mathtt{DEV_A}$ and $\mathtt{DEV_A}$ can calculate same session key key_i, because they knows $k_i^{'''}, r_A, r_B$ and $Auth_C$. □

5.2 Security

At first we describe that our usage of OSK-protocol and Pietezak's pseudorandom generator is still pseudo-random. In the following theorems, we consider the pseudorandom generator implemented in $\texttt{TPM}_\texttt{A}$ as $\mathcal{G}(i,\cdot)$. When we input the latest secret key s_{i-1}, this generator outputs $(s_i, a_i) = \mathcal{G}(i, s_{i-1})$ where $a_i = (k_i, k_i', k_i'', k_i''')$.

Theorem 2 (Security of underlying pseudorandom generator). *Output of our usage of OSK protocol is still pseudo-random without leakage of internal state. And output of our usage of Pietrzak's pseudo-random generator is still pseudo-random against the adversary who try to partial leakage allowed in Pietrzak's mode-of-operation.*

Proof. This security of this theorem is distinguishing $(k_i, k_i', k_i'', k_i''')$ from random using $(k_{i-1}, k_{i-1}', k_{i-1}'', k_{i-1}''')$. Let the success probability of distinguishing k_i from random using k_{i-1}''' be $Pr[Succ(\mathcal{A}_{PRG})]$ and success probability of distinguishing $(k_i, k_i', k_i'', k_i''')$ from random using $(k_{i-1}, k_{i-1}', k_{i-1}'', k_{i-1}''')$ be $\Pr[Succ(\mathcal{A}_{4PRG})]$.

Then

$$\begin{aligned}|1 - 2 \cdot \Pr[Succ(\mathcal{A}_{4PRG})] \leq |1 - 2 \cdot ((&\Pr[Succ(\mathcal{A}_{PRG})])^4 \\ &+4/2^{|k|} \times (\Pr[Succ(\mathcal{A}_{PRG})])^3 \\ &+6/2^{2|k|} \times (\Pr[Succ(\mathcal{A}_{PRG})])^2 \\ &+4/2^{3|k|} \times (\Pr[Succ(\mathcal{A}_{PRG})]) \\ &+1/2^{4|k|})|\end{aligned}$$

Theorem in [12] shows that $\Pr[Succ(\mathcal{A}_{PRG})]$ is negligible without corruption. Similarly, theorem in [13] shows that $\Pr[Succ(\mathcal{A}_{PRG})]$ is negligible with partial leakage. Thus, $\Pr[Succ(\mathcal{A}_{4PRG})]$ is negligible. □

Theorem 3 (Key security without any corruption). *The proposed protocols (both basic and leakage-resilient versions) have semantic security of renewal session key against any adversary who observe protocol messages. This means the adversary cannot obtain no information about further session keys.*

Proof. We proceed in games, starting with Game 0 which is the original security game between a challenger and adversary in the proposed protocol. The challenger simulates all party's registration and the response to the oracle queries that $\mathcal{A}$ issues. Let $Succ_f(\mathcal{B})$ be the event that an probabilistic algorithm $\mathcal{B}$ breaks the security property of the function f.

In each Game i, we define adv_i as the advantage that the adversary wins the game. We consider the following games:

Game 0. This is the original security game with adversary $\mathcal{A}$ so that $\mathsf{adv}_0 = adv_P^{maker}(t, Q)$.

Game 1. We proceed as Game 0 but add the following abort rule. The challenger proceeds as Game 0 but aborts the game if it does not correctly guess the test session.

Game 2. We transform Game 1 to Game 2 by changing the secret key $(k_i, k_i', k_i'', k_i''')$ used in the test session to random strings.

Game 3. We modify Game 2 by changing pseudorandom function $F_{k_i}(\cdot)$ to truly random function RF used at the test session.

Game 4. We proceed as Game 3 but add the following rule. If the adversary changes the communication message at the test session and the verification of the MAC function is accepted, the challenger aborts the security game in Game 4.

Game 5. We modify Game 4 by changing pseudorandom function $F_{k_i'''}(\cdot)$ to truly random function RF used at the test session.

Game 6. We modify Game 5 to Game 6 by changing the session key at the test session to the truly random string.

We evaluate the relations between the game transformation with the following claims.

Claim 1. We have $\mathsf{adv}_0 \leq Q/2 \cdot \mathsf{adv}_1$.

Proof. From the definition of the security model, the upper bound of the oracle queries issued by the adversary $\mathcal{A}$ is at most Q. To establish a server's accepted session, $\mathcal{A}$ must issue two send queries from the specification of the proposed protocol and the server executes at most $Q/2$ sessions in the security game. When the challenger uniformly selects i from $1, \ldots, q$ such that server's i-th session will be chosen as the test session. Then the probability that the challenger correctly guesses the test session is at least $2/Q$. Therefore, $\mathsf{adv}_0 \leq Q/2 \cdot \mathsf{adv}_1$.

Claim 2. We have $|\mathsf{adv}_1 - \mathsf{adv}_2| \leq \Pr[Succ_{\mathcal{G}}(\mathcal{B}'_{4PRG})]$.

Proof. If the adversary $\mathcal{A}$ can distinguish Game 2 from Game 1 with non-negligible probability, there exists an probabilistic algorithm $\mathcal{B}'_{4PRG}$ that can break the security of pseudorandom generator $\mathcal{G}(i, \cdot)$.

For a given instance $(k_i, k_i', k_i'', k_i''')$, $\mathcal{B}'_{PRG}$ proceeds as Game 1 except that $(k_i, k_i', k_i'', k_i''')$ is assigned as the secret key for the i-th server's session. When the adversary output a guess bit b' for the test session, $\mathcal{B}'_{4PRG}$ outputs the same bit b'. If the tuple is computed by pseudorandom generator, this game is equivalent to Game 1. Otherwise, $\mathcal{B}'_{4PRG}$ simulates Game 2 from the view point of the adversary. Therefore, it is clear that if the adversary can distinguishes Game 2 from Game 1 with probability ϵ_1, $\mathcal{B}'_{4PRG}$ can distinguishes $(k_i, k_i', k_i'', k_i''')$ with the same probability. However, Theorem 2 showed that this probability is already negligible.

Claim 3. We have $|\mathsf{adv}_2 - \mathsf{adv}_3| \leq \Pr[Succ_F(\mathcal{B}_{PRF})]$.

Proof. If the adversary $\mathcal{A}$ can distinguish Game 3 from Game 2 with non-negligible probability, there exists an probabilistic algorithm $\mathcal{B}_{PRF}$ that can break the security of pseudorandom function.

Consider that $\mathcal{B}_{PRF}$ interacts with pseudo-random function $F_{k_i}(\cdot)$ or truly random function RF. $\mathcal{B}_{PRF}$ proceeds as Game 2 and simulates all the oracle queries except the test session. We assume that the adversary sends $(r'_A, \cdot)$ to the server and $(r'_B, \cdot)$ to the client at the test session. $\mathcal{B}_{PRF}$ issues $r_A \| r'_B$ and $r'_A \| r_B$ to the oracle query where (r_A, r_B) is chosen by $\mathcal{B}_{PRF}$ and $\mathcal{B}_{PRF}$ computes $(Auth_B, Auth_A)$ using the response from the challenger. When the adversary output a guess bit b' for the test session, $\mathcal{B}_{PRF}$ outputs the same bit b'. From the security proof of the previous claim, each secret key input to the pseudorandom function is chosen by uniformly random. So if the adversary can distinguish these games, $\mathcal{B}_{PRF}$ can break the security of PRF with the same probability.

Claim 4. We have $|\mathsf{adv}_3 - \mathsf{adv}_4| \leq \Pr[Succ_{\mathsf{MAC}}(\mathcal{B}_{MAC})] + 2/2^{l_F}$ where l_F is the output length of the pseudorandom function $F_{k_i}(\cdot)$.

Proof. It is clear that these games are equivalent if the adversary does not output the modification message such that the party accepts it. We show that if the adversary succeeds in outputting such a valid message with non-negligible probability, there exists an probabilistic algorithm $\mathcal{B}_{MAC}$ that can break the existential unforgeability of the MAC function.

Consider that the adversary sends $(r'_A, \cdot)$ to the server, $(r'_B, Auth'_B)$ to the client and $Auth'_A$ to the server at each round of the test session. If $r'_A \neq r_A$ or $r'_B \neq r_B$, the client and the server input different variables to the pseudorandom function $F_{k_i}(\cdot)$ and its outputs are completely independent from the previous claim. Thus, $m_B = F_{k'_i}(r'_A \| r_B) \oplus Auth_C$ computed by the server and $m_A = F_{k'_i}(r_A \| r'_B) \oplus Auth_C$ computed by the client are different messages except the negligible probability $1/2^F$. If $Auth'_B$ is accepted by the client, we can construct $\mathcal{B}_{MAC}$ who simulates the game as Game 3 and outputs $(m_A, Auth'_B)$ as a forgery. $\mathcal{B}_{MAC}$ issues m_B to the MAC oracle but $m_A = m_B$ happens with probability $1/2^{l_F}$. The same argument holds for the MAC value $Auth'_A$ if we assume that $r'_B \neq r_B$. So the probability that server accepts with modified message is also negligible. Therefore, we have $\|\mathsf{adv}_3 - \mathsf{adv}_4\| \leq \Pr[Succ_{\mathsf{MAC}}(\mathcal{B}_{MAC})] + 2/2^{l_F}$.

Claim 5. We have $|\mathsf{adv}_4 - \mathsf{adv}_5| \leq \Pr[Succ_F(\mathcal{B}_{PRF})]$.

Proof. We can show the proof of this claim as previous claim for the difference between Game 2 and 3. In this case, $\mathcal{B}_{PRF}$ interacts with pseudo-random function $F_{k'''_i}$ or truly random function RF and issues (r_A, r_B) to the oracle query. So if the adversary can distinguishes these games, $\mathcal{B}_{PRF}$ can break the security of PRF with the same probability.

Claim 6. We have $|\mathsf{adv}_5 - \mathsf{adv}_6| = 0$.

Proof. The session key at the test session is already randomized since $F_{k'''_i}(r_B)$ is replaced by uniformly random string and it is effectively a one-time pad. Therefore, this change is purely conceptual and we obtain $|\mathsf{adv}_5 - \mathsf{adv}_6| = 0$.

It is obvious that $\mathsf{adv}_6 = 0$, and we obtain

$$adv_P^{maker}(t,Q) \leq Q/2 \cdot (\Pr[Succ_{\mathcal{G}}(\mathcal{B}'_{4PRG})] + 2 \cdot \Pr[Succ_F(\mathcal{B}_{PRF})] + \Pr[Succ_{\mathsf{MAC}}(\mathcal{B}_{MAC})] + 2/2^{l_F})$$

and conclude the proof of Theorem 3. □

We remark that $Sucv_P^{auth}(t,Q)$ is also negligible from Claim 4.

Theorem 4 (Key security with $\mathtt{CORRUPT}_1$). *The proposed protocols (both basic and leakage-resilient versions) have semantic security of renewal session key against any adversary who observe protocol messages and obtain output from* $\mathtt{TPM}_\mathtt{A}$. *This means the adversary cannot obtain no information about further session keys.*

Proof. The security proof of this theorem is derived from Theorem 2 and 3. The difference between Theorem 3 and 4 is that the adversary can obtain $\{(k_j, k'_j, k''_j, k'''_j)\}$ for $j = 1, \ldots, i-1$ whose variables are used as the secret key before the test session. Nonetheless, Theorem 2 shows that pseudorandom generator $\mathcal{G}(i,\cdot)$ still holds the security and $(k_i, k'_i, k''_i, k'''_i)$ is independent from these keys. Then we can easily construct the security proof based on Theorem 3. Therefore,

$$adv_P^{maker}(t,Q) \leq Q/2 \cdot (\Pr[Succ_{\mathcal{G}}(\mathcal{B}_{4PRG})] + 2 \cdot \Pr[Succ_F(\mathcal{B}_{PRF})] + \Pr[Succ_{\mathsf{MAC}}(\mathcal{B}_{MAC})] + 2/2^{l_F})$$

and $\mathcal{B}_{4PRG}$ is negligible from Theorem 2.

Theorem 5 (Key security with $\mathtt{CORRUPT}_1$ and $\mathtt{CORRUPT}_2$). *The proposed protocols (only leakage-resilient versions) have for key security of renewal session key against any adversary who observe protocol messages and obtain output and partial internal information from* $\mathtt{TPM}_\mathtt{A}$. *This means the adversary cannot obtain no information about further session keys.*

Proof. In addition to Theorem 4, this type of adversary can issue leakage oracle and obtain $\lambda(a_i)$, where $\lambda(\cdot)$ is leakage function chosen by the adversary. When we consider the hash chain likes OSK protocol or traditional pseudorandom generator, the security is no more ensured. However, if we use the leakage resilient pseudorandom generator proposed by Pietrzak et al., our protocol also satisfies the leakage resilience and we can describe the security proof as in the Theorem 4. Therefore, we have

$$adv_P^{maker}(t,Q) \leq Q/2 \cdot (\Pr[Succ_{\mathcal{G}}(\mathcal{B}''_{4PRG})] + 2 \cdot \Pr[Succ_F(\mathcal{B}_{PRF})] + \Pr[Succ_{\mathsf{MAC}}(\mathcal{B}_{MAC})] + 2/2^{l_F})$$

and we can show that $\Pr[Succ_{\mathcal{G}}(\mathcal{B}''_{4PRG})]$ is negligible from the theorem in Pietrzak et al. □

Theorem 6 (Forward secrecy with $\texttt{CORRUPT}_1$ and $\texttt{CORRUPT}_3$). *The proposed protocols (both basic and leakage-resilient versions) have forward secrecy of renewal session key against any adversary who observe protocol messages and obtain output and full internal information from $\texttt{TPM}_\texttt{A}$. This means the adversary cannot obtain no information about previous session keys.*

Proof. This type of adversary can obtain all the internal secret used after the test session, in addition to the adversary described in Theorem 3.

When we consider the OSK-type hash chain pseudorandom generator $\mathcal{G}$, the adversary can receive $(i+1)$-th session's internal secret $s_{i+1} = G(H^{-1}(k_i'''))$ with $\texttt{CORRUPT}_3$ query. Note that the other secret keys are derived from this value. In this case, we can easily show the independence of the secret key of the test session. When we set as $t = H^{-1}(k_i''')$, we have $s_{i+1} = G(t)$ and $k_i''' = H(t)$. Since the each pseudorandom generator G and H implemented in the $\texttt{TPM}_\texttt{A}$ is independently chosen in our protocol, s_{i+1} does not affect the pseudo-randomness of k_i'''. Of course, we can say that the pseudo-randomness of k_i'', k_i' and k_i also holds recursively.

In the case of leakage resilient pseudorandom generator $\mathcal{G}$, the output from the $\texttt{TPM}_\texttt{A}$ at the test session is $a_i = (k_i, k_i', k_i'', k_i''') = (X_{5i-4}, X_{5i-3}, X_{5i-2}, X_{5i-1})$. When the adversary issues $\texttt{CORRUPT}_3$ query to the $(i+1)$-th session, he can receive $s_{i+1} = (K_{5i}, K_{5i+1}, X_{5i})$. Note that $(K_{5i}, X_{5i-1}) = \mathsf{F}(k_{5i-2}, X_{5i-2})$ and $(K_{5i+1}, X_{5i}) = \mathsf{F}(k_{5i-1}, X_{5i-1})$ where F is iterated pseudorandom generator used in $\mathcal{G}$. Thus s_{i+1} does not affect the pseudo-randomness of a_i.

Therefore, the corruptions after the test session does not affect the security proof and

$$adv_P^{maker}(t,Q) \leq Q/2 \cdot (\Pr[Succ_{\mathcal{G}}(\mathcal{B}'_{4PRG})] + 2 \cdot \Pr[Succ_F(\mathcal{B}_{PRF})] + \Pr[Succ_{\mathsf{MAC}}(\mathcal{B}_{MAC})] + 2/2^{l_F}).$$

We can evaluate the security reduction as Theorem 3. □

5.3 Discussion

From the above security evaluation, we can see that security of one side of multi-factor $Auth_C$ cannot be guaranteed after any corruption (by only $\texttt{CORRUPT}_1$). This is because the adversary can obtain $r_A, r_B, Auth_A, Auth_B$ and its encryption key, then the security of $Auth_C$ is not guaranteed by definition of pseudo-random functions. This is the so, as is the fact that this protocol is not secure against online dictionary attack. However this is due to inherent limitations of using symmetric key cryptography, which is pointed out by Halevi et al [7]. The only countermeasure for this is slow-downing of calculation of inverses of pseudo-random function by using multiple PRFs.

However, generally $Auth_C$ contains a password, which can be changeable at any time (and is changed according to systems rules every so often). Even if $\texttt{CORRUPT}_1$ or $\texttt{CORRUPT}_3$ are made, the security of $Auth_C$ part revives after changing password. This is a different characteristics than that of internal secret (keys).

Internal secrets cannot be changed after corruption, thus it should be protected by hash chain or Pietrzak's mode. However, password can be easily changed, thus, the current setting is still effective in practice.

6 Conclusion

This paper proposed multifactor authenticated key renewal protocol, which realizes authenticated key establishment from symmetric (shared) key cryptography only. The authentication part of this protocol deals with both authentication from device's secret and human related authenticator like memorable password. The system model of this protocol is suitable for the situation of current mobile computing and other devices. In practice, all functions in this protocol are realized by the standard AES function or other secure light-weight block cipher. Thus, this is efficient enough for smartphone, mobile devices and smartcards. The proposed protocol deals with many types of attacks, such as data interception between TPM and application program, and side channel attacks on the TPM. This protocol assures session key secrecy and forward secrecy against the above attacks.

Acknowledgement. This work was supported by KAKENHI, Grant-in-Aid for Young Scientists (B) (22760287).

References

1. Avoine, G., Oechslin, P.: A scalable and provably secure hash-based RFID protocol. In: Proc. of the PerCom 2005 Workshops (2005)
2. Juels, A., Pappu, R.: Squealing Euros: Privacy Protection in RFID-Enabled Banknotes. In: Wright, R.N. (ed.) FC 2003. LNCS, vol. 2742, pp. 103–121. Springer, Heidelberg (2003)
3. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure against Dictionary Attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
4. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory IT-22(6), 644–654 (1976)
5. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. Internet Draft, RFC 5246 (2008)
6. Dziembowski, S., Pietrzak, K.: Leakage-resilient cryptography. In: Proc. FOCS, October 25-28, pp. 293–302 (2008)
7. Halevi, S., Krawczyk, H.: Public-key cryptography and password protocols. ACM Transactions on Information and System Security (TISSEC) 2(3), 230–268 (1999)
8. Katz, J., Ostrovsky, R., Yung, M.: Forward Secrecy in Password-Only Key Exchange Protocols. In: Cimato, S., Galdi, C., Persiano, G. (eds.) SCN 2002. LNCS, vol. 2576, pp. 29–44. Springer, Heidelberg (2003)
9. Kolesnikov, V., Rackoff, C.: Key Exchange Using Passwords and Long Keys. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 100–119. Springer, Heidelberg (2006)

10. Kent, S., Seo, K.: Security Architecture for the Internet Protocol. Internet Draft, RFC 4301 (2005)
11. Matsuo, S., Phong, L.T., Ohkubo, M., Yung, M.: Leakage-Resilient RFID Authentication with Forward-Privacy. In: Ors Yalcin, S.B. (ed.) RFIDSec 2010. LNCS, vol. 6370, pp. 176–188. Springer, Heidelberg (2010)
12. Ohkubo, M., Suzuki, K., Kinoshita, S.: Cryptographic Approach to "Privacy-Friendly" Tags. In: RFID Privacy Workshop. MIT, USA (2003)
13. Pietrzak, K.: A Leakage-Resilient Mode of Operation. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 462–482. Springer, Heidelberg (2009)
14. Petit, C., Standaert, F.-X., Pereira, O., Malkin, T., Yung, M.: A Block Cipher based Pseudo Random Number Generator Secure against Side-channel Key Recovery. In: Proc. of ASIACCS 2008, pp. 56–65 (2008)
15. Pointcheval, D., Zimmer, S.: Multi-factor Authenticated Key Exchange. In: Bellovin, S.M., Gennaro, R., Keromytis, A.D., Yung, M. (eds.) ACNS 2008. LNCS, vol. 5037, pp. 277–295. Springer, Heidelberg (2008)
16. Sarma, S.E., Weis, S.A., Engels, D.W.: RFID Systems and Security and Privacy Implications. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 454–469. Springer, Heidelberg (2003)
17. Trusted Computing Group, `http://www.trustedcomputinggroup.org/`
18. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Protocol Architecture. Internet Draft, RFC 4541 (2006)