Algorithmic Tamper Proof (ATP) Counter Units for Authentication Devices Using PIN

Yuichi Komano[1], Kazuo Ohta[2], Hideyuki Miyake[1], and Atsushi Shimbo[1]

[1] Toshiba Corporation,
1, Komukai Toshiba-cho, Saiwai-ku, Kawasaki 212-8582, Japan
{yuichi1.komano,hideyuki.miyake,atsushi.shimbo}@toshiba.co.jp
[2] The University of Electro-Communications,
Chofugaoka 1-5-1, Chofu-shi, Tokyo 182-8585, Japan
ota@ice.uec.ac.jp

Abstract. Though Gennaro et al. discussed the algorithmic tamper proof (ATP) devices using the personal identification number (PIN) with less tamper-proof devices, and proposed counter units which count the number of wrong attempts in user authentication; however, as for the counter unit, they only constructed one which counts the *total* number of wrong attempts. Although large number for the limit of wrong attempts is required for usability, it allows an attacker to search PIN up to the limit and degrades the security. The construction of secure counter units which count the number of *consecutive* wrong attempts remains as an open problem. In this paper, we first formalize the ATP security of counter units, and propose two constructions of counter unit which count the number of consecutive wrong attempts. The security of each construction can be proven under the assumptions of secure signature scheme and random function. The former one is required to store two states in secure memory area (RP-Mem) with low computation cost; and the latter one has high computation cost but is required to store only one state in RP-Mem. This shows the trade-off between the costs of hardware and algorithm.

Keywords: algorithmic tamper proof (ATP), counter unit, PIN authentication.

1 Introduction

1.1 Background

Physical attacks like the fault attack [2] threaten the security of devices and several countermeasures with particular hardware are proposed. Gennaro et al. [3] discussed the algorithmic tamper proof (ATP) devices in which software algorithm cooperates with hardware to ensure the security against such attacks. It relaxes the hardware requirements and gives us practical device implementations.

In [3], they constructed the ATP devices, such as decryption and signature generation ones. Moreover, they insisted that, in order to prohibit unauthorized

M. Abdalla et al. (Eds.): ACNS 2009, LNCS 5536, pp. 306–323, 2009.

users from running the devices, an authentication using the personal identification number (PIN) should be performed. In order to secure the device, they claimed that it should self-destruct if more wrong attempts of authentication are detected than the limit number which is pre-determined as a system parameter.

In considering ATP devices, the property of memory area must be carefully discussed. They [3] prepare two kinds of memory area for the device. One is a *read-proof* but tamperable area (we call it *read proof memory*, RP-Mem); and the other is a *tamper-proof* but readable one (*tamper proof memory*, TP-Mem). Read-proof (resp. tamper-proof) memory is one where attacker is not permitted to read (write by fault injection etc.) data in (into) it. In order to decrease the manufacture cost, TP-Mem should be a read only memory where the common data for devices is hardwired (see [3] or section 3.2). This paper also uses RP-Mem and TP-Mem to construct ATP counter units as well as [3].

There are two types of counter unit. One counts the *total* number of wrong attempts; and the other counts the number of *consecutive* wrong ones. In [3], they gave a concrete unit for the former one and claimed its security (without a formal discussion); however, the former one has a dilemma in choosing the limit of wrong attempts. Although the limit number should be large enough in case the legal user misses the authentication during the life time of devices; it allows an attacker to search the PIN by try and error up to the limit, and degrades the security. The latter one can solve this dilemma but the ATP secure construction of such unit remains as an open problem.

Let us review the counter unit proposed in [3] for the total number of wrong attempts (see appendix for detail). The unit stores a state (R_i) in RP-Mem which is a node of hash chain and relates to the total number of wrong attempts. The leaf of hash chain is signed by the trusted party. When an attempt fails, it updates the state with hashing it (move to next node). If a certain attacker over-writes the state maliciously, then it can be detected because the signature must not be valid for a leaf of hash chain from the over-written state; and then the device self-destructs to prevent the attacker from accessing it. If the state is readable for the attacker, then it is possible to rewind the counter by copying the state; therefore, the state should be stored in RP-Mem. See appendix A.1 for detail.

1.2 Our Contribution

In this paper, we first define a security model of counter units by giving the attack scenario and goal formally. In this model, an attacker can read (write) data in (into) TP-Mem (RP-Mem); and moreover, she can run functionalities of counter units at her will. We then propose two constructions of a counter unit which counts the number of consecutive wrong attempts and prove their security.

Note that the counter unit which counts the number of consecutiv wrong attempts requires a reset functionality and our security model allows an attacker to use it. In order to prevent the attacker maliciously running this functionality, we let reset the counter after PIN is certified in the reset functionality.

The first construction, named ATP-CU1 (algorithmic tamper-proof counter unit 1), stores *two* states $state_1$ and $state_2$ (nodes of a hash chain) in RP-Mem.

$state_1$ represents the total number of wrong attempts like [3] and $state_2$ relates to the latest success of the authentication. The correctness of these states can be checked whether $state_2$ is mapped to $state_1$ with the hash function (strictly, we assume a pseudorandom function); and the number of consecutive wrong attempts is measured by the distance between them (figure 2). The security of ATP-CU1 can be ensured by the pseudorandomness of hash function and the unforgeability of underlying signature scheme.

The other one, named ATP-CU2, is based on *one* state and a chameleon hash function [6,1]. The state directly correlates to the number of consecutive wrong attempts and its correctness is checked whether the chameleon hash function maps the state to the pre-fixed data with respect to the signature stored in RP-Mem. The state is updated with the open algorithm of the chameleon hash function. ATP-CU2 is secure if the chameleon hash function is collision resistant one-way and if the underlying signature scheme is unforgeable. Compared to ATP-CU1, ATP-CU2 utilizes the chameleon hash function to make data (corresponds to $state_2$ in ATP-CU1) fixed and to decrease the variable state by one; however, the chameleon hash function generally requires high computation and long output. This shows the trade-off between the costs of hardware and algorithm.

This paper is organized as follows. Section 2 reviews the definitions of preliminaries. In section 3, we model the ATP devices using PIN with counter units and formalize ATP security of counter units. We then construct two counter units for the consecutive wrong attempts and prove their security in sections 4 and 5, respectively. Section 6 makes the discussion and section 7 concludes this paper. Moreover, in appendix A, we revisit the counter unit [3] for the total number of wrong attempts to give a formal discussion.

2 Preliminaries

Let us review the signature scheme, pseudorandom function and chameleon hash function. We first recall the model of signature scheme and its security [5] this paper assumes. Since the existential unforgeability against key only attack, *not* against chosen message attack, is enough for ensuring the security of our units, this section review the former one. We assume the *trusted* device vendor generates a pair of public and secret keys, makes signatures and stores the pulic key and signatures in *tamper-proof* and *read-proof* memories (see section 3.2), respectively; and therefore, no one except the vendor gets a pair of message and signatures and our assumption of key only attack is meaningful.

Definition 1 (Signature Scheme). The signature scheme consists of the following three algorithms.

1) **Key Generation Algorithm,** Gen**:** Given a security parameter k as an input, it outputs a key pair of public and secret keys $\mathsf{Gen}(1^k) = (pk, sk)$. This algorithm is probabilistic.

2) **Signing Algorithm,** Sign**:** Given a message m and a secret key sk as inputs, it outputs a signature $\mathsf{Sign}_{sk}(m) = s$. This algorithm may be probabilistic.
3) **Verification Algorithm,** Ver**:** Given a message mCa signature s, and a public key pk as inputs, it outputs $\mathsf{Ver}_{pk}(m, s) = 1$ (accept) if s is valid for m and pk; or $\mathsf{Ver}_{pk}(m, \sigma) = 0$ (reject) otherwise. This algorithm is deterministic.

Definition 2 (Existential Unforgeability against Key Only Attack). We say that a signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is existentially unforgeable against key only attack in (τ, ϵ) if, for all attacker A whose running time is bounded by τ, the success probability of A is at most ϵ. Here, the success probability of A is defined by $\Pr[(pk, sk) \leftarrow \mathsf{Gen}(1^k); (m^*, s^*) \leftarrow A(pk) : \mathsf{Ver}_{pk}(m^*, s^*) = 1]$.

We then recall the definition of pseudorandom function [4].

Definition 3 (Pseudorandom Function). For a security parameter k, the family of hash functions $\{f_k : \{0,1\}^k \rightarrow \{0,1\}^k\}$ is pseudorandom if it satisfies the following property: For all polynomial time algorithm M, $\epsilon = |\Pr[M^{f_{U_k}}(1^k) = 1] - \Pr[M^{F_k}(1^k) = 1]| < \frac{1}{p(k)}$ holds for all positive function p and sufficiently large k. Here, F_k is an arbitrary function from $\{0,1\}^k$ to $\{0,1\}^k$ and U_k is an element chosen randomly and uniformly from $\{0,1\}^{|k|}$.

Finally, let us recall the definition of chameleon hash function [6,1].

Definition 4 (Chameleon Hash Function). Let R be a recipient and let k, k_m, k_r be security parameters. The chameleon hash function consists of the following three algorithms.

1) **Key Generation Algorithm,** CGen**:** Given k, it outputs a key pair of public and private keys $\mathsf{CGen}(1^k) = (CPK, CSK)$ for R. This algorithm is probabilistic.
2) **Chameleon Hash Algorithm,** CH**:** Given a message $m \in \{0,1\}^{k_m}$, a randomness $r \in \{0,1\}^{k_r}$, and CPK; it outputs a hash value $\mathsf{CH}_{CPK}(m, r) \in \{0,1\}^k$. This algorithm is deterministic.
3) **Open Algorithm** CHI**:** Given messages $m, m' \in \{0,1\}^{k_m}$, a randomness $r \in \{0,1\}^k$, and CSK; it outputs $\mathsf{CHI}_{CSK}(m, r, \mathsf{CH}_{CPK}(m, r)) = r'$ such that $\mathsf{CH}_{CPK}(m, r) = \mathsf{CH}_{CPK}(m', r')$ holds.

Definition 5 (Properties of CH**).** Let R be a recipient and let CPK and CSK denote the public and secret keys of R, respectively. We say that the chameleon hash function CH_{CPK} (and its open algorithm CHI_{CSK}) is secure if it satisfies the following properties.

1) **Uniformity:** For all m, if r is chosen at random from $\{0,1\}^{k_r}$, then $\mathsf{CH}_{CPK}(m, r)$ is uniformly distributed in $\{0,1\}^k$.
2) **Collision Resistance:** We say that the chameleon hash function CH_{CPK} is collision resistant in (τ_C, ϵ_C) if, for all attacker A whose running time is bounded by τ_C, the success probability of A is at most ϵ_C. Here, the success probability of A is defined by $\Pr[(CPK, CSK) \leftarrow \mathsf{CGen}(1^k); \{(m, r), (m', r')\} \leftarrow A(CPK) : (m \neq m') \wedge (\mathsf{CH}_{CPK}(m, r) = \mathsf{CH}_{CPK}(m', r'))]$.

3 ATP Devices Using PIN

This section formalizes the ATP devices which authorize a user with a PIN. We assume that no one can guess PIN with success probability more than ϵ_P within time bound τ_P. Note that ϵ_P is polynomial, *e.g.*, $\frac{1}{10^4}$ for four digit PIN; however, we assume that it has enough entropy (*i.e.*, PIN is chosen randomly) for disturbing a few guesses allowed by devices with our counter unit.

3.1 Function of Devices

This paper deals with devices which self-destruct if the authentication fails more than the limit number (*e.g.*, 3 times), in order to prevent malicious users from using functionalities of such devices. For this purpose, a counter unit which counts the number of wrong attempts in authentication should be implemented. See figure 1 for its construction.

There are two types of functionalities of devices. One consists of sensitive functionalities that should be invoked only after the authentication succeeds (*e.g.*, signing and decryption functionalities); the other consists of ones which can be called at any time (*e.g.*, authentication functionality).

The device works in the following way. The counter is initialized with zero. A user who requests to use a sensitive functionality first runs the authentication functionality. If the authentication succeeds, then she is allowed to run the sensitive one. If not, on the other hand, the device increments the counter with update functionality (subroutine of authentication) and rejects her request.

3.2 Memory Area of Devices

We classify the memory unit of device into four types with respect to readability and tamperability for an attacker as in Table 1. In this table, NP-Mem stands for a non-protected memory and an attacker can read and write data in NP-Mem by probing etc. RP-Mem is a read-proof memory where an attacker can write data into it by injecting faults etc., but can not read data in it. It can be realized by the

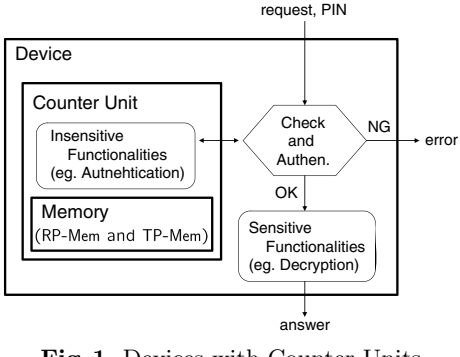

Fig. 1. Devices with Counter Units

Table 1. Types of Memory Unit

		Read	
		Yes	No
Write	Yes	NP-Mem	RP-Mem
	No	TP-Mem	RTP-Mem

special hardware [7] or memory encryption[1], for example. TP-Mem is a tamper-proof memory where an attacker can read data in it but can not write data into it. From the viewpoint of mass production [3], TP-Mem stores (hardwires) the common data for devices and is regarded as a read only memory (ROM). Our aim is, following that of [3], to implement a secure counter unit with RP-Mem and TP-Mem (and NP-Mem for storing temporary data).

Note that, if the read-proof and tamper-proof area (RTP-Mem in Table 1.) is available, the secure counter unit can be obviously constructed when RTP-Mem holds the number of wrong attempts. However, RTP-Mem costs high and seems not to be implemented in small devices; therefore, we follow [3] to assume RTP-Mem is not available. Note also that, since memory area of RP-Mem and TP-Mem tends to highly cost compared to NP-Mem, we should estimate the allocation for each area and decrease the use of area highly costing.

3.3 Security Requirements for ATP Devices Using PIN

In this subsection, we model the ability (attack scenario) of attacker against the device and her aim (attack goal).

Ability of Attacker. We assume that an attacker can invoke insensitive functionalities at will. In this scenario, the device can be utilized as a black box to give her hints. Moreover, she is allowed to change the data stored in tamperable memory areas (NP-Mem and RP-Mem) and to read the data stored in readable memory areas (NP-Mem and TP-Mem) on her will.

This scenario is an extension of the fault attack to cryptographic devices [2]. Note that while the practical fault attacks change the data by chance; our scenario allows her to specify the position and value of data flipped by her attack. She is, however, allowed neither to run the sensitive functionality directly (bypassing the authentication) nor to skip the operation inside the insensitive functionality[2].

[1] Devices encrypt the data and store the encrypted one. The devices hold a decryption key for read the data. In practice, the memory encryption seems a practical but the decryption key should be managed properly. In order to discuss the provable security, if the memory encryption is utilized to realize RP-Mem, the key management is also modeled formally; however, it is an out of scope how RP-Mem is realized for this paper, we omit it.

[2] See Remark 1 in appendix A for the necessity of this assumption.

We give the attack model with following queries.

- *data read query:* With the query $\mathsf{Read}(\mathsf{type}, \mathsf{var})$, the attacker receives the value val of the variable var if it is stored in $\mathsf{type} \in \{\mathsf{NP\text{-}Mem}, \mathsf{TP\text{-}Mem}\}$; or $\perp$ otherwise. Assume that the output is returned immediately after the query.
- *data write query:* With the query $\mathsf{Write}(\mathsf{type}, \mathsf{var}, h)$, the attacker replaces the value val with $h(\mathsf{val})$ of the variable var if it is stored in $\mathsf{type} \in \{\mathsf{NP\text{-}Mem}, \mathsf{RP\text{-}Mem}\}$; or receives $\perp$ otherwise. Assume that the replacement (or the output of $\perp$) is performed immediately after the query.
- *functionality run query:* With the query $\mathsf{Run}(\mathsf{func}, \mathsf{ope})$, the attacker runs the functionality func with the operand ope and receives its output if it exists. Assume that, while some functionality runs with this query, he is allowed to request data read and write queries but she is disallowed to request another functionality run query concurrently[3].

Note that functionalities performed by the functionality run query above vary from units. For instance, ATP-CU1 has three functionalities (Main, Update_1, and Update_2) and we allow an attacker to run them with the query.

Goal of Attacker. Although the supreme goal of an attacker is to recover *PIN*, we impose her with another goal: to attempt the authentication more than the limit. For example, assume that the limit number is three. In this setting, the device should self-destruct if fourth wrong attempt happens (in total or consecutive). We say that an attacker succeeds in this attack if she performs fifth attempt with no knowledge about *PIN*. Note that, if this attack is mounted, there is a chance to find *PIN* by try and error.

ATP Security for Devices with Counter Unit. We call the device with counter unit is ATP secure if there is no polynomial time attacker who achieves the attack goal under the attack scenarios above with non-negligible success probability.

4 Algorithmic Tamper-Proof Counter Unit 1: ATP-CU1

This section shows the counter unit ATP-CU1 which counts the number of consecutive wrong attempts. This is one of solutions to the open problem of [3].

4.1 Construction of ATP-CU1

ATP-CU1 utilizes two states $state_1 = R$ and $state_2 = S$ which are the nodes of a hash chain (strictly, a chain of pseudorandom function). R represents the total number of wrong attempts as in [3] and S relates to the latest success attempt. Their correctness can be checked whether S is mapped to R with the hash function G; $R = \mathsf{G}^i(\sigma_1, S)$ for $i \leq m$ where σ_1 and m are the signature

[3] See Remark 2 in appendix A for the necessity of this assumption.

Table 2. Memory Allocation for ATP-CU1

TP-Mem	$m, k, \mathsf{G}, \mathsf{Ver}, pk$
RP-Mem	$state_1 = R, state_2 = S, \sigma_1, \sigma_2$

and limit number, respectively (see below). If they are correct, the number of consecutive wrong attempts is measured by the distance between them; i in the previous equality. If the attempt fails, it updates R by replacing it with $\mathsf{G}(\sigma_1, R)$. On the other hand, if the attempt succeeds, it updates $state_2 = S$ by replacing it with $\mathsf{G}^{i-1}(\sigma_1, S)$. See figure 2 for example.

Notation. We introduce the notations of variables and preliminaries for ATP-CU1.

- PIN: PIN
- m: limit number of wrong attempts
- k: security parameter
- $\{\mathsf{G}(\sigma, \cdot) : \{0,1\}^k \times \{0,1\}^k \to \{0,1\}^k\}$: pseudorandom function[4]
- $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$: signature scheme
- (pk, sk): public and secret keys for signature scheme with respect to the trusted device vendor[5]
- $state_1 = R$ and $state_2 = S$

The vendor runs $\mathsf{Gen}(1^k)$ to generate pk and sk that are used for all devices in advance.

Initialization. The vendor initializes the device as follows.

1. He decides PIN (if necessary, by interacting with a user).
2. He computes $\sigma_1 \leftarrow \mathsf{Sign}_{sk}(PIN)$ and $\sigma_2 \leftarrow \mathsf{Sign}_{sk}(\sigma_1)$.
3. He chooses $S \in \{0,1\}^k$ at random.
4. He computes $R \leftarrow \mathsf{G}(\sigma_1, S)$.
5. He initializes $state_1 \leftarrow R$ and $state_2 \leftarrow S$.
6. He allocates the data as in Table 2.

Note that at step 2, PIN may be padded with a random nonce (in this case, it is stored in RP-Mem and used in verification) in order to enlarge the entropy of the input of signing function.

The authentication (check for the correctness of inputted PIN') is performed by the verification of PIN' and σ_1. Moreover, σ_1, which is a part of input of G, disable an attacker to predict the output of G. On the other hand, σ_2 ensures that σ_1 (stored in tamperable area) is unchanged by $\mathsf{Write}(\mathsf{RP\text{-}Mem}, \sigma_1, h)$ with signing verification. This verification ensures that σ_2 is also unchanged.

We denote, by $\mathsf{G}^n(\sigma_1, R)$ for $\sigma_1 \in \{0,1\}^k$ and $R \in \{0,1\}^k$, the operation where G is repeatedly applied n times. For instance, $\mathsf{G}^2(\sigma_1, R) = \mathsf{G}(\sigma_1, \mathsf{G}(\sigma_1, R))$.

[4] Assume that the first input of G (signature σ_1 below) is a key for pseudorandom function.

[5] Hereafter, we assume that the device vendor is trusted.

Authentication Process. In correspondence with a request from a user, the device authorizes the user and updates the states as follows. Figure 2 shows the translation of states utilized in ATP-CU1.

—— $\mathsf{Main}(PIN')$ ——

1. If $R \notin \{\mathsf{G}(\sigma_1, S), \cdots, \mathsf{G}^{m+1}(\sigma_1, S)\}$ holds, then it self-destructs.
2. If $\mathsf{Ver}_{pk}(\sigma_1, \sigma_2) = 0$ holds, then it self-destructs.
3. Otherwise, the following steps are performed.
   (a) If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 1$ holds, then the following steps are performed.
      i. If $\mathsf{G}(\sigma_1, S) = R$ holds, then it accepts the request.
      ii. If $\mathsf{G}(\sigma_1, S) \neq R$ holds, then it runs $\mathsf{Update}_2(PIN')$ and accepts the request.
   (b) If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 0$ holds, then it runs $\mathsf{Update}_1(\phi)$ and rejects the request.

—— $\mathsf{Update}_1(\phi)$ ——

1. If $\mathsf{Ver}_{pk}(\sigma_1, \sigma_2) = 0$ holds, then it self-destructs.
2. It updates R with $\mathsf{G}(\sigma_1, R)$.

—— $\mathsf{Update}_2(PIN')$ ——

1. If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 0$ holds, then it self-destructs.
2. If $\mathsf{Ver}_{pk}(\sigma_1, \sigma_2) = 0$ holds, then it self-destructs.
3. Otherwise, the folloing steps are performed.
   (a) If $R = \mathsf{G}^n(\sigma_1, S) \in \{\mathsf{G}^2(\sigma_1, S), \cdots, \mathsf{G}^{m+1}(\sigma_1, S)\}$ holds, then it updates S with $\mathsf{G}^{n-1}(\sigma_1, S)$.
   (b) If $R \notin \{\mathsf{G}^2(\sigma_1, S), \cdots, \mathsf{G}^{m+1}(\sigma_1, S)\}$ holds, then it self-destructs.

The first step of Main computes the hash chain for each invocation and checks the number of consecutive wrong attempts is within the limit.

Note that the number of consecutive wrong attempt is estimated by $n - 1$ if $R = \mathsf{G}^n(\sigma_1, S)$ holds at step 1 in Main. Also note that we assume that an attacker can run the subroutines of authentication (Update_1 and Update_2) directly. Therefore, in order to prevent the attacker from running such subroutines maliciously, the checks performed in Main are also done in Update_1 and Update_2.

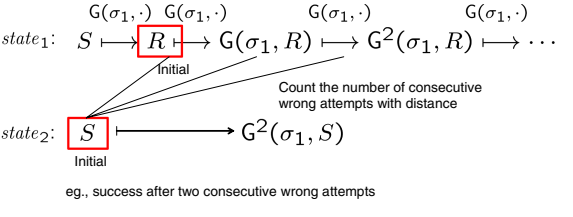

Fig. 2. Translation of States in ATP-CU1

4.2 Security Consideration of ATP-CU1

As for the security of ATP-CU1, we will prove the following theorem.

Theorem 1. If it is infeasible to find *PIN* within time bound τ_P and with the probability more than ϵ_P, if it is infeasible to break the pseudorandomness of G within time bound τ_G with the probability more than ϵ_G, and if it is infeasible to break existential unforgeability against key only attack within time bound τ_B with the probability ϵ_B, then it is infeasible to break the ATP security of ATP-CU1 within time bound τ_A with the probability more than ϵ_A. The following inequalities hold.

$$\begin{cases} \tau_B \leq (m+2)\tau_P + \tau_G + \tau_A + O(1)\mathsf{T}, \\ \epsilon_B \geq \epsilon_A - (m+2)\epsilon_P - \epsilon_G - 1/2^{k-1} \end{cases}$$

Here, T denotes the running time of authentication device once.

Proof: By using the attacker A against ATP-CU1 as a black box, we construct an algorithm B which takes the key only attack to existentially forge a signature of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$.

The aim of A is to perform the PIN authentication $m+2$ times. As described in section 3.3, A outputs the following queries: data read query $\mathsf{Read}(\mathsf{type}, \mathsf{var})$, data write query $\mathsf{Write}(\mathsf{type}, \mathsf{var}, h)$, and functionality run query $\mathsf{Run}(\mathsf{func}, \mathsf{ope})$ where $\mathsf{func} \in \{\mathsf{Main}, \mathsf{Update}_1, \mathsf{Update}_2\}$.

B, given a public key pk of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ and a pseudorandom function G as inputs, simulates the answers to queries from A and uses A as a black box to output a forgery (m^*, s^*) of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$. B first initializes the input of A and data for a device as follows.

Initialization:

1. B decides *PIN*.
2. B chooses $\sigma_1, \sigma_2 \in \{0,1\}^k$ at random.
3. If $\mathsf{Ver}_{pk}(PIN, \sigma_1) = 1$ holds, then B outputs (PIN, σ_1) as a forgery and stops.
4. If $\mathsf{Ver}_{pk}(\sigma_1, \sigma_2) = 1$ holds, then B outputs (σ_1, σ_2) as a forgery and stops.
5. B chooses $S \in \{0,1\}^k$ at random.
6. B computes $R \leftarrow \mathsf{G}(\sigma_1, S)$.
7. B sets $state_1 = R$ and $state_2 = S$.

B invokes A with inputs G and pk, and answers to queries from A as follows.

Answers to $\mathsf{Read}(\mathsf{type}, \mathsf{var})$*:*

1. If $\mathsf{type} \notin \{\mathsf{NP\text{-}Mem}, \mathsf{TP\text{-}Mem}\}$ holds, then B returns $\perp$ to A.
2. B returns the value val of the variable var to A.

Answers to $\mathsf{Write}(\mathsf{type}, \mathsf{var}, h)$*:*

1. If $\mathsf{type} \notin \{\mathsf{NP\text{-}Mem}, \mathsf{RP\text{-}Mem}\}$ holds, then B returns $\perp$ to A.
2. B replaces the value val of the variable var with $h(\mathsf{val})$.

Answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$*:*

1. If another $\mathsf{Run}(*, *)$ is being performed, then B returns $\perp$ to A.
2. If A replaces at least one of σ_1 and σ_2 by different value with Write query, then the following steps are performed.
   (a) Same as step 3 of Initialization.
   (b) Same as step 4 of Initialization.
   (c) Otherwise, B halts (this case corresponds to the self-destruction).
3. If A does not replace σ_1 nor σ_2 by different value with Write query, then B updates R with $\mathsf{G}(\sigma_1, R)$.

Answers to $\mathsf{Run}(\mathsf{Update}_2, PIN')$*:*

1. Same as step 1 of answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$.
2. If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 1$ holds, then B outputs (PIN', σ_1) as a forgery and stops.
3. Same as step 2 of answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$.
4. If A does not replace σ_1 nor σ_2 by different value with Write query, then B terminates and fails to output a forgery (B aborts).

Answers to $\mathsf{Run}(\mathsf{Main}, PIN')$*:*

1. Same as step 1 of answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$.
2. If $R \notin \{\mathsf{G}(\sigma_1, S), \cdots, \mathsf{G}^{m+1}(\sigma_1, S)\}$ holds, then B halts.
3. Same as step 2 of answers to $\mathsf{Run}(\mathsf{Update}_2, PIN')$.
4. Same as step 2 of answers to $\mathsf{Run}(\mathsf{Update}_1, \phi)$.
5. If this is the $(m+2)$-th run for $\mathsf{Run}(\mathsf{Main}, *)$ and if A does not replace σ_1 by different value with Write query, then B aborts.
6. Otherwise (before the $(m+2)$-th run and none of σ_1 and σ_2 is replaced by different value), the following steps are performed.
   (a) If $PIN' = PIN$ holds, then B aborts.
   (b) If $PIN' \neq PIN$ holds, then B runs $\mathsf{Update}_1(\phi)$ and rejects the request.

In the following three cases, B fails to simulate the answers and to output a forgery (*i.e.*, B aborts): step 4 of answers to $\mathsf{Run}(\mathsf{Update}_2, PIN')$, and steps 5 and 6(a) of answers $\mathsf{Run}(\mathsf{Main}, PIN')$. We should show that these cases occur by accident.

As for the case in step 4 of answers to $\mathsf{Run}(\mathsf{Update}_2, PIN')$, A can distinguish B from ATP-CU1 only when A inputs $PIN' = PIN$ but B aborts. It happens with probability less than ϵ_P.

We then estimate the probability with which B aborts in step 5 of answers to $\mathsf{Run}(\mathsf{Main}, PIN')$. Note that σ_1 (which is unchanged with Write query) is randomly chosen from $\{0,1\}^k$ and stored in RP-Mem. In this case, unless A successfully guesses σ_1 with probability $1/2^k$, σ_1 is uniformly distributed in $\{0,1\}^k$ for A. In the $(m+2)$-th run of Main, in order to pass $R \in \{\mathsf{G}(\sigma_1, S), \cdots, \mathsf{G}^{m+1}(\sigma_1, S)\}$ (step 2) for $state_1 = R$ and $state_2 = S$, at least one of R and S should be replaced by different value with Write query. Under the assumption of uniformity of σ_1

for A, however, the output of $\mathsf{G}(\sigma_1, \cdot)$ is also uniformly distributed in $\{0,1\}^k$, unless A breaks the pseudorandomness of G with probability ϵ_G. Namely, the probability with which A replaces R and/or S to pass the check of step 2 is bounded by $1/2^k$. Therefore, A can distinguish B from ATP-CU1 in this step with probability less than $\epsilon_G + 1/2^{k-1}$.

In step 6(a) of answers to $\mathsf{Run}(\mathsf{Main}, PIN')$, A can distinguish B from ATP-CU1 with probability less than $(m+1)\epsilon_P$, because A is allowed to attempt the authentication at most $m+1$ times. ■

5 Algorithmic Tamper-Proof Counter Unit 2: ATP-CU2

We then give another construction of counter unit ATP-CU2 which also counts the number of consecutive wrong attempts and discuss its security.

5.1 Construction of ATP-CU2

ATP-CU2 utilizes a chameleon hash function CH [6,1] to update the state $state = (i, R_i)$ which corresponds to the number of consecutive wrong attempts i. The correctness of (i, R_i) is checked whether the chameleon hash function maps (i, R_i) to the pre-fixed data V with respect to the signature σ_2 stored in $\mathsf{RP\text{-}Mem}$; it checks whether $V = \mathsf{CH}(\sigma_1, i, R_i)$ and σ_2 pass the signature verification or not. The state is updated with the open algorithm CHI of the chameleon hash function in order for the state to be mapped to the pre-fix data by the chameleon hash function; it finds R_j such that $\mathsf{CH}(\sigma_1, i, R_i) = V = \mathsf{CH}(\sigma_1, j, R_j)$ where $j = 0$ if the attempt succeeds and $j = i + 1$ otherwise. See figure 3 for example.

Notation. In addition to PIN, m, k, $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, and (pk, sk) described in section 4.1, the following notations are used.

- $(\mathsf{CGen}, \mathsf{CH}, \mathsf{CHI})$: chameleon hash function[6] where CH_{CPK} is $\{0,1\}^{k_{m_s}} \times \{0,1\}^{k_{m_i}} \times \{0,1\}^{k_r} \to \{0,1\}^k$ and CHI_{CSK} is $\{0,1\}^{k_{m_s}} \times \{0,1\}^{k_{m_i}} \times \{0,1\}^{k_r} \times \{0,1\}^{k_{m_s}} \times \{0,1\}^{k_{m_i}} \times \{0,1\}^k \to \{0,1\}^{k_r}$, respectively
- (CPK, CSK): public and secret keys for chameleon hash function with respect to the device vendor
- $state = (i, R_i)$: state $(i \geq 0)$

The vendor runs Gen and CGen to generate (pk, sk) and (CPK, CSK) that are used for all devices in advance. Note that CH_{CPK} is collision resistant one-way and has the following open property. With the secret key CSK, CHI_{CSK} finds another preimage of $V = \mathsf{CH}_{CPK}(\sigma_1, i, R_i)$ with different $j(\neq i)$; $\mathsf{CHI}_{CSK}(\sigma_1, i, R_i, \sigma_1, j, V)$ returns R_j such that $\mathsf{CH}_{CPK}(\sigma_1, j, R_j) = V$ holds.

[6] We assume that $k_{m_s} + k_{m_i} = k_m$ holds and that the first two (and last two moreover) inputs for CH (CHI) are contatenated into k_m bit string.

Table 3. Memory Allocation for ATP-CU2

TP-Mem	$m, k, \mathsf{CH}, \mathsf{CHI}, CPK, \mathsf{Ver}, pk$
RP-Mem	$state = (i, R_i), CSK, \sigma_1, \sigma_2$

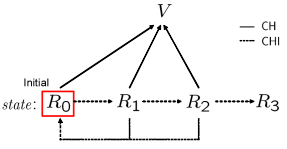

Fig. 3. Translation of State in ATP-CU2

Initialization. The vendor initializes the device as follows.

1. He decides PIN (if necessary, by interacting with a user).
2. He chooses $R_0 \in \{0,1\}^k$ at random.
3. He computes $\sigma_1 \leftarrow \mathsf{Sign}_{sk}(PIN)$; $V \leftarrow \mathsf{CH}_{CPK}(\sigma_1, 0, R_0)$; and $\sigma_2 \leftarrow \mathsf{Sign}_{sk}(V)$.
4. He initializes $state = (i, R_i) \leftarrow (0, R_0)$.
5. He allocates the data as in Table 3.

Authentication Process. In correspondence with a request from a user, the device authorizes the user and updates the states as follows. Figure 3 shows the translation of state utilized in ATP-CU2.

—— $\mathsf{Main}(PIN')$ ——
1. If $\mathsf{Ver}_{pk}(\mathsf{CH}_{CPK}(\sigma_1, i, R_i), \sigma_2) = 0$ holds, then it self-destructs.
2. Otherwise, the following steps are performed.
   (a) If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 1$ holds, then the following steps are performed.
      i. If $i > 0$ holds, then it runs $\mathsf{Update}_4(PIN')$ and accepts the request.
      ii. If $i = 0$ holds, then it accepts the request.
   (b) If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 0$ holds, then it runs $\mathsf{Update}_3(\phi)$ and rejects the request.

—— $\mathsf{Update}_3(\phi)$ ——
1. If $\mathsf{Ver}_{pk}(\mathsf{CH}_{CPK}(\sigma_1, 0, R_i), \sigma_2) = \cdots = \mathsf{Ver}_{pk}(\mathsf{CH}_{CPK}(\sigma_1,\ m, R_i), \sigma_2) = 0$ holds, then it self-destructs.
2. Otherwise, it updates (i, R_i) with $(i+1, \mathsf{CHI}_{CSK}(\sigma_1, i, R_i, \sigma_1, i+1, \mathsf{CH}_{CPK}(\sigma_1,\ i, R_i)))$.

—— $\mathsf{Update}_4(PIN')$ ——
1. If $\mathsf{Ver}_{pk}(PIN', \sigma_1) = 0$ holds, then it self-destructs.
2. If $\mathsf{Ver}_{pk}(\mathsf{CH}_{CPK}(\sigma_1, i, R_i), \sigma_2) = 0$ holds, then it self-destructs.
3. It updates (i, R_i) with $(0, R_0 \leftarrow \mathsf{CHI}_{CSK}(\sigma_1, i, R_i, \sigma_1, 0, \mathsf{CH}_{CPK}(\sigma_1, i, R_i)))$.

If the number of consecutive wrong attempts exceeds m, the device self-destructs and does not work any more at the first stem of Update_3.

5.2 Security Consideration of ATP-CU2

The security of ATP-CU2 can be similarly discussed as in section 4.2; and therefore, we omit the detail here. With regard to the security of ATP-CU2, the following theorem holds.

Theorem 2. If it is infeasible to find *PIN* within time bound τ_P with probability more than ϵ_P, if it is infeasible to break the security (collision resistance) of CH within time bound τ_C with probability more than ϵ_C, and if it is infeasible to break the existentially forgeability against key only attack, then it is infeasible to break the ATP security of ATP-CU2 within time bound τ_A with probability more than ϵ_A. The following inequalities hold.

$$\begin{cases} \tau_B \leq (m+2)\tau_P + \tau_G + \tau_A + O(1)\mathsf{T}, \\ \epsilon_B \geq \epsilon_A - (m+2)\epsilon_P - \epsilon_G - m/2^{k-1} \end{cases}$$

Here, T denotes the running time of authentication device once.

6 Discussion

This section first considers a naive construction of counter unit which counts the number of consecutive wrong attempts from the counter unit of [3] which counts the total number of wrong attempts. After that, let us compare the counter units; that of [3], ATP-CU1 and ATP-CU2.

6.1 Naive Construction and Its Weakness

Let us consider a counter unit which counts the number of consecutive wrong attempts with states $state_1 = (i, R)$ and $state_2 = (j, S)$ (initially, we set $i = 1$ and $j = 0$) in the same manner as the counter unit of [3]. Its algorithm is almost same as that of ATP-CU1 except it checks whether $i - j \in [1, m+1]$ and $R = \mathsf{G}^{i-j}(\sigma_1, S)$ hold instead of step 1 of $\mathsf{Main}(PIN')$ and step 3(a) of $\mathsf{Update}_2(PIN')$ for ATP-CU1.

In order to prove the security of above naive construction, we require another restriction to an attacker as follows: She is disallowed to make a data write query between the check of $i - j \in [1, m+1]$ and $R = \mathsf{G}^{i-j}(\sigma_1, S)$ is performed. Note that since i and j correlate the number of wrong attempts, she can guess them. If she is allowed to make the query between the check, the following attack can be done. (1) Before the check of $i - j \in [1, m]$, she changes $i = 1$ and $j = 0$ with data write query to pass the check, and (2) after the check of $i - j \in [1, m]$, she rewrite i and j with guessed data with data write query to pass the check of $R = \mathsf{G}^{i-j}(\sigma_1, S)$.

With the above restriction, the naive construction can be proven to be ATP secure from the similar discussion of the ATP security of ATP-CU1. On the other hand, ATP-CU1 removes the restriction by checking, with high computation cost, whether one of $R = \mathsf{G}(\sigma_1, S)$, $R = \mathsf{G}^2(\sigma_1, S)$, $\cdots$, or $R = \mathsf{G}^{m+1}(\sigma_1, S)$ holds.

6.2 Comparison

This section compares the constructions of counter units. Note that the unit of [3] has to use large m in case of mistakes of legal users; however, in order to prevent attackers from searching PIN by try and error, m cannot be enlarged so enough. On the other hand, the proposed units can use small m with keeping high security if we assume that the legal user cannot input wrong PIN consecutively so many times.

Let us consider the proposed ones. Note that the computation of chameleon hash function (CH and CHI) requires higher cost than that of ordinal hash function (G). Therefore, ATP-CU1 can require lower cost than ATP-CU2; however, ATP-CU1 should store and update two states in $\mathsf{RP\text{-}Mem}$ in secure way. This seems a trade-off between the memory use and the computation cost.

7 Conclusion

This paper is first one to model the ATP security of counter unit and to construct two ATP secure counter units counting the number of consecutive wrong attempts. The security of the first one is proven in the assumption of existentially unforgeable signature scheme against key only attack and of pseudorandom function. Moreover, the security of the second one is also proven in the assumption of existentially unforgeable signature scheme against key only attack and of secure chameleon hash function. The first one is required to store two states into $\mathsf{RP\text{-}Mem}$ but has less computation (with hash function); on the other hand, the second one needs high computation (with chameleon hash function) but is required to store one state into $\mathsf{RP\text{-}Mem}$. This paper revisits the counter unit proposed in [3] counting the total number of wrong attempts in authentication with PIN in appendix.

References

1. Ateniese, G., de Medeiros, B.: On the key exposure problem in chameleon hashes. In: Blundo, C., Cimato, S. (eds.) SCN 2004. LNCS, vol. 3352, pp. 165–179. Springer, Heidelberg (2005)
2. Boneh, D., DeMillo, R.A., Lipton, R.J.: On the importance of checking cryptographic protocols for faults. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 37–51. Springer, Heidelberg (1997)
3. Gennaro, R., Lysyanskaya, A., Malkin, T., Micali, S., Rabin, T.: Algorithmic tamper-proof (ATP) security: Theoretical foundations for security against hardware tampering. In: Naor, M. (ed.) TCC 2004. LNCS, vol. 2951, pp. 258–277. Springer, Heidelberg (2004)
4. Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. In: 25th Annual IEEE Symposium on Foundations of Computer Science (FOCS 1984), pp. 464–479. IEEE, Los Alamitos (1984)
5. Goldwasser, S., Micali, S., Rivest, R.: A digital signature scheme against adaptive chosen message attack. Journal of Computing (Society for Industrial and Applied Mathematics) 17(2), 281–308 (1988)

6. Krawczyk, H., Rabin, T.: Chameleon signatures. In: Network and Distributed System Security Symposium, NDSS 2000. The Internet Society (2000)
7. Tuyls, P., Schrijen, G.-J., Škorić, B., van Geloven, J., Verhaegh, N., Wolters, R.: Read-proof hardware from protective coatings. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 369–383. Springer, Heidelberg (2006)

A Counter Unit of Gennaro et al. [3]

Gennaro et al. [3] constructed the counter units for the total number of wrong attempts and for the number of consecutive wrong attempts, respectively. Since the construction of latter one is not described in detail ([3] only claims that the unit uses the modulus number), this section discusses the former one only. As for the former one, however, reference [3] did not specify its algorithm nor the memory allocation for each data. This section takes form its construction.

A.1 Necessity of RP-Mem

They [3] did not mention the necessity of RP-Mem[7]. We show here that their counter unit is vulnerable when RP-Mem is not utilized. Without RP-Mem, data is stored in readable area, TP-Mem or NP-Mem.

The counter unit of [3] prepares the state R_0 (the following subsection gives notations in detail), hash chain $R_i = \mathsf{H}(R_{i-1})$ for $i = 1, 2, \cdots, m+1$, and signature σ for R_{m+1}. The state is initially set with R_0 and updated by applying H when an authentication is failed. The correctness of state R_i is checked by the verification of $\mathsf{H}^{m+1-i}(R_i)$ and σ.

Assume the device with TP-Mem and NP-Mem, without RP-Mem. In order to be updated (re-stored), the state should not be stored in TP-Mem but in NP-Mem. In this setting, the attacker A first read R_0 and copy it in local area (like his computer). When A fails in attempt, R_0 in the device is updated to $R_1 = \mathsf{H}(R_0)$. Since R_1 is stored in NP-Mem, A can replace it with R_0 (stored in local area) and try attempt infinitely.

Therefore, the read-proof (but tamperable) area, RP-Mem, should be required for storing and updating the state.

A.2 Construction of Counter Unit of [3]

A state *state* is used for counting the total number of wrong attempts. *state* is updated and unchanged when an authentication fails and succeeds, respectively.

Notation. In addition to PIN, m, k, $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, and (pk, sk) described in section 4.1, the following notations are used.

- $\mathsf{H} : \{0,1\}^k \to \{0,1\}^k$: collision resistant one-way function
- $state = (i, R_i)$: state $(i \geq 0)$

[7] TP-Mem should be required for storing the code of hash function H.

Table 4. Memory Allocation for Counter Unit of [3]

TP-Mem	$m, k, \mathsf{H}, \mathsf{Ver}, pk$
RP-Mem	$state = (i, R_i), R_{m+1}, \sigma$

Initialization. The vendor initializes the device as follows.

1. He decides PIN (if necessary, by interacting with a user).
2. He chooses $R_0 \in \{0,1\}^k$ at random.
3. He computes $R_{m+1} \leftarrow \mathsf{H}^{m+1}(R_0)$.
4. He computes $\sigma \leftarrow \mathsf{Sign}_{sk}(PIN, R_{m+1})$.
5. He initializes $state = (i, R_i) \leftarrow (0, R_0)$.
6. He allocates the data as in Table 4.

We denote, by $\mathsf{H}^n(R)$ for $R \in \{0,1\}^k$, the operation where H is repeatedly applied n times. For instance, $R_{l+2} = \mathsf{H}^2(R_l) = \mathsf{H}(\mathsf{H}(R_l))$.

In table 4, R_{m+1} and σ is stored in RP-Mem. If R_{m+1} and σ (namely, R_0) are identical for all devices, they may be stored in TP-Mem.

Authentication Process. In correspondence with a request from a user, the device authorizes the user and updates the state as follows.

—— $\mathsf{Main}(PIN')$ ——
1. If $\mathsf{H}^{m+1-i}(R_i) \neq R_{m+1}$ holds, then it self-destructs.
2. Otherwise, the following steps are performed.
   (a) If $\mathsf{Ver}_{pk}(PIN', \mathsf{H}^{m+1-i}(R_i), \sigma) = 1$ holds, then it accepts the request.
   (b) Otherwise, it runs $\mathsf{Update}(\phi)$ and rejects the request.

—— $\mathsf{Update}(\phi)$ ——
1. It updates $(i, R_i) \leftarrow (i+1, \mathsf{H}(R_i))$.

A.3 Security of Counter Unit of [3]

The reference [3] claims, without a proof, that the unit above is ATP secure if the signature scheme is existentially unforgeable [5] and if the hash function is collision resistant one-way. This subsection discusses its security. We have following three remarks.

Remark 1. If an attacker is allowed to skip the operations inside the insensitive functionality (see the third paragraph of section 3.3), she can mount an attack to the counter unit [3] as follows: She makes the unit skip step 2(b) of Main (or first step of Update) not to update *state*, she can attempt to authenticate more than the limit number. Therefore, we assume that the attacker is disallowed to skip the operations.

Remark 2. If an attacker can run functionalities concurrently with functionality run queries (see the definition of functionality run query in section 3.3), she can also mount an attack to the counter unit [3]. For instance, if she makes another function run query before the unit proceeds the step 2(b) of Main (or first step of Update), she can attempt to authenticate more than the limit number. Hence, we restrict the attacker not to make functionality run queries concurrently.

Remark 3. Note that in order to achieve its security, the signature scheme should be utilized once (*i.e.*, one-time signature scheme) and resulting signature should be stored in RP-Mem. This is because, if an attacker can obtain or read a signature corresponding some *PIN*, she may rewind the counter with the signature and exhaustively search *PIN*.

With regard to the security of their unit, the following theorem holds.

Theorem 3. If it is infeasible to find *PIN* within time bound τ_P with probability more than ϵ_P, if it is infeasible to break the pseudorandomness of H within time bound τ_H with probability more than ϵ_H, and if it is infeasible to break the existentially forgeability of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ against key only attack, then it is infeasible to break the ATP security of their unit within time bound τ_A with probability more than ϵ_A. The following inequalities hold.

$$\begin{cases} \tau_B \leq (m+2)\tau_P + \tau_H + \tau_A + O(1)\mathsf{T}, \\ \epsilon_B \geq \epsilon_A - (m+2)\epsilon_P - \epsilon_H - m/2^k \end{cases}$$

Here, T denotes the running time of authentication device once.