Hindawi
Security and Communication Networks
Volume 2019, Article ID 1719585, 13 pages
https://doi.org/10.1155/2019/1719585

Research Article

Single-Round Pattern Matching Key Generation Using Physically Unclonable Function

Yuichi Komano,[1] **Kazuo Ohta**,[2] **Kazuo Sakiyama**,[2] **Mitsugu Iwamoto**,[2] **and Ingrid Verbauwhede**[3]

[1]*Toshiba Corporation, Kawasaki, Japan*
[2]*The University of Electro-Communications, Tokyo, Japan*
[3]*KU Leuven, Leuven, Belgium*

Correspondence should be addressed to Yuichi Komano; yuichi1.komano@toshiba.co.jp

Received 6 July 2018; Revised 9 November 2018; Accepted 12 December 2018; Published 1 January 2019

Guest Editor: Daniel Schneider

Paral and Devadas introduced a simple key generation scheme with a physically unclonable function (PUF) that requires no error correction, e.g., by using a fuzzy extractor. Their scheme, called a pattern matching key generation (PMKG) scheme, is based on pattern matching between auxiliary data, assigned at the enrollment in advance, and a substring of PUF output, to reconstruct a key. The PMKG scheme repeats a round operation, including the pattern matching, to derive a key with high entropy. Later, to enhance the efficiency and security, a circular PMKG (C-PMKG) scheme was proposed. However, multiple round operations in these schemes make them impractical. In this paper, we propose a single-round circular PMKG (SC-PMKG) scheme. Unlike the previous schemes, our scheme invokes the PUF only once. Hence, there is no fear of information leakage by invoking the PUF with the (partially) same input multiple times in different rounds, and, therefore, the security consideration can be simplified. Moreover, we introduce another hash function to generate a check string which ensures the correctness of the key reconstruction. The string enables us not only to defeat manipulation attacks but also to prove the security theoretically. In addition to its simple construction, the SC-PMKG scheme can use a weak PUF like the SRAM-PUF as a building block if our system is properly implemented so that the PUF is directly inaccessible from the outside, and, therefore, it is suitable for tiny devices in the IoT systems. We discuss its security and show its feasibility by simulations and experiments.

1. Introduction

The Internet of Things (IoT) is widely spread to make us more intelligent, efficient, and comfortable. In IoT systems, devices are located everywhere to exchange their sensing data and their control information. On the other hand, lots of devices in these systems are resource-constrained where it is hard to implement security functions. Unlike the closed system with a limited number of devices, in the IoT systems, attackers are able to obtain devices to analyze them maliciously, and therefore they can be weak points of these systems.

Let us consider the safety of the IoT systems. First of all, the reliability of data is important because the devices work unwillingly with the improper data, especially with the data manipulated by the attacker. Moreover, the correctness of the firmware including the safety functionalities, such as the fail-stop and the fail-tolerance, is also important. In order to avoid the manipulation of the firmware by attacker so that the safe functionality does not work, the firmware should be well protected and securely updated. Both for the data reliability and for the firmware correctness, the security is essential. Hence, securing such devices is one of our emerging challenges. Particularly, the management of key is a crucial task.

The physically unclonable function (PUF) is one of the promising primitives to improve the security of tiny devices. It derives a unique value for each device (function) from its fine characteristics. For example, the SRAM-PUF [1–3] uses initial states of the SRAM cells just after the power-on as such characteristics; and, the Arbiter-PUF [4–6] uses the logic delays of a dual-rail circuit as such ones. The unique value derived is used as (a source of) an identity (ID) or a

cryptographic key. The PUF is suitable for a tiny device for the following two reasons. First, since these characteristics naturally arise during the manufacturing process, we can remove the process of embedding an ID or a key and decrease the manufacturing cost. Second, since analyzing the unique and fine characteristics is difficult, it can be regarded as a secure and tamper- evident storage which proves the physical manipulation for analyzing the key.

On the other hand, the output of PUF may include a small noise for each invocation, e.g., 5% or up to 15% noise in the signal, because of external factors such as external/ambient temperature variations and/or supplied voltage variations. In order to use the output as a cryptographic key, the noise should be removed by error correction techniques, such as the fuzzy extractor [7, 8], because the cryptographic results completely differ if the key differs by only one bit. The fuzzy extractor, however, includes a complex error correction code whose cost might be high for the tiny devices.

1.1. Pattern Matching Key Generation Using PUF. Paral and Devadas [9] gave an interesting solution to remove the noises without an error correcting code. Their proposal uses a pattern matching which only estimates the Hamming distance between two strings. Their main idea is to regard the index (bit position) indicating a substring of the PUF output as a secret, instead of the (sub)string of PUF output itself. This scheme is called a pattern matching key generation (PMKG) scheme.

The PMKG scheme consists of multiple rounds. In each round of its enrollment phase, a key generation device extracts a substring of PUF output indicated by a secret index and stores the substring into the public (and maybe insecure) nonvolatile memory (NVM) area, as auxiliary data. The PMKG scheme regards a (hash value of a) concatenation of the substrings as a secret key. In each round of its reconstruction phase, the device compares a noisy PUF output with the stored auxiliary data and recovers the corresponding index and eventually the key.

Against the PMKG scheme, Delvaux and Verbauwhede [10, 11] gave attacks, named *snake attacks*, to recover the secret indices. Their attacks modify the auxiliary data stored in the NVM bit by bit in each round and detect the index by running the device with the modified data. To avoid the snake attacks, they also suggested regarding the PUF output as circular data. We call the scheme a circular PMKG (C-PMKG) scheme.

The PMKG (C-PMKG, respectively) scheme stores, for each round, the substring (resp., circularly shifted string) of PUF output into the NVM as auxiliary data. If two auxiliary data pieces for different rounds have the same substring (intersection), it might leak information on the secret indexes for the corresponding rounds. To mitigate the leakage, there are several solutions. The first one is to control the inputs of PUF for rounds to avoid such intersection. Another one is to enlarge the input/output space so as to neglect the intersection. These solutions increase the implementation cost and may be inappropriate for tiny devices.

1.2. Our Contribution. In this paper, we propose a simple and efficient PMKG scheme, named a single-round circular PMKG (SC-PMKG) scheme. Unlike the previous schemes, the SC-PMKG scheme consists of a single-round operation with multiple indexes. It divides a PUF output into multiple substrings, circularly shifts each substring by each secret index, and stores the shifted data into the NVM as auxiliary data. Moreover, it also stores a hash value, as a check string to check the correctness of the key reconstruction, into the NVM.

By reducing the number of rounds in the (C-)PMKG scheme to one, we need not care for the information leakage from the PUF output strings in the auxiliary data with the (partially) same input, and therefore the security discussion can be simplified. In addition, the check string not only disables the manipulation attacks but also ensures the provable security of our scheme in the random oracle model [12]. We give a security proof under the condition (assumption), for simplicity, that there is no adversarial interface to access the internal PUF. This condition also provides our choice of the PUF candidates applicable to the SC-PMKG scheme. Without such interfaces, machine learning attacks to the PUF are impractical, and, hence, various PUFs can be applicable to the SC-PMKG, including a weak PUF such as the SRAM-PUF. (The modeling attack requires challenge-response pairs (CRPs) of PUF, which is applicable to PUF applications where adversaries can obtain the PUF CRPs, such as an authentication with PUF. In the application of the key generation, adversaries can obtain the PUF outputs from auxiliary data. Against this application, the modeling attack could be meaningless for the following two reasons. First, the corresponding inputs for the auxiliary data are unknown to adversaries. Second, the number of auxiliary data is limited; for example, if the key generation device is specific to a key of certain application, adversaries can obtain PUF outputs only for the key.) Note that, on the provable security, we can relax the assumption where there are adversarial interfaces to the PUF *except* one to directly access the output corresponding to the target key, as we discuss in Section 4.3. Such discussions on the adversarial conditions and the security show that the security coengineering for the developments of the PUF devices and of the PUF application is important.

Furthermore, we check its feasibility by simulations and experiments. We also check the validity with theoretical estimation. Our theoretical estimation can derive parameter candidates not only for the SC-PMKG scheme but also for the C-PMKG schemes. From our discussion, the SC-PMKG scheme is secure and efficient rather than the previous PMKG schemes; and, therefore, it is suitable for tiny devices to tackle the emerging threats against the IoT systems.

1.3. Organization. In Section 2, we give the definitions of PUF and PUF-based key generation scheme. We then review the constructions of previous PMKG schemes and discuss their drawbacks in Section 3. In Section 4, we present the SC-PMKG scheme and discuss its security. We then test its feasibility and make comparisons among the PMKG schemes in Sections 5 and 6, respectively. Finally, Section 7 concludes this paper.

2. Definition

In this section, we review definitions of the PUF and PUF-based key generation scheme.

2.1. PUF. References, such as [13, 14], gave a definition of the PUF with several properties. In this paper, we regard the PUF as a physical function satisfying relaxed properties below, in order for various types of PUF to be applicable to our PMKG.

Definition 1 (PUF). A family of physical functions is called the family of PUFs if these functions satisfy the following properties.

(1) Each function works within polynomial steps

(2) Each function produces an output including a small noise; however, the signal-to-noise ratio is high enough to remove the noise by using an error correcting code. Hence, it acts as a function which provides a particular output for an input

(3) It is practically hard to characterize or clone each function

(4) Functions produce unique outputs per each of them

The third property does not ensure that it is hard to characterize or clone the physical function. The function may be characterized or cloned in future, if more sophisticated physical equipment to analyze it appears. In this paper, we regard the SRAM-PUF, Arbiter-PUF, and ring-oscillator PUF (RO-PUF) with fine processes and careful designs as PUFs.

We give a security assumption of PUF, which is a formal description of above third condition, to be used in the security proof of the SC-PMKG scheme.

Definition 2 (indistinguishable PUF). Let ℓ_c and ℓ_p be input and output lengths of PUF, respectively. We state that a physically unclonable function PUF is indistinguishable in (τ, q, ϵ) if any adversary $\mathscr{A}$, within the time bound τ, cannot achieve an advantage more than ϵ, even if $\mathscr{A}$ observes q input-output pairs $\{(x_i, y_i)\}$. The advantage is defined with $\mathsf{Adv}_{\mathscr{A}}^{\mathsf{Ind-PUF}} = \Pr[\mathscr{A}(y) = 1 \mid \mathscr{A}(1^{\ell_c}, 1^{\ell_p}) = x; y = \mathsf{PUF}(x)] - \Pr[\mathscr{A}(y) = 1 \mid \mathscr{A}(1^{\ell_c}, 1^{\ell_p}) = x; y \longleftarrow \{0,1\}^{\ell_p}]$. Here, $\mathscr{A}$ is allowed to output x where there is no overlap between $\mathsf{PUF}(x)$ and the observed PUF outputs $\mathsf{PUF}(x_i)$ for $i \in [1, q]$.

This indistinguishability captures the unpredictability and unclonability, which relate to the third property of Definition 1. Intuitively, the advantage $\mathsf{Adv}_{\mathscr{A}}^{\mathsf{Ind-PUF}}$ is a metric of adversary's ability to distinguish the PUF output from a random string. The distinguishing game is as follows: The adversary $\mathscr{A}$ chooses an input x for the PUF and receives either an output $y = \mathsf{PUF}(x)$ or a random string $y \in \{0,1\}^{\ell_p}$. $\mathscr{A}$ outputs 1 if $\mathscr{A}$ supposes that $y = \mathsf{PUF}(x)$. In this game, $\mathscr{A}$ may collect input-output pairs of PUF, as hints, by accessing the PUF oracle.

Let us assume the SRAM-PUF. In the SRAM-PUF, the challenge is an ℓ_c-bit address x, and the output y is a sequence of initial states of ℓ_p SRAM cells starting with an address x. Definition 2 allows $\mathscr{A}$ to collect a pair $\{(x_i, y_i)\}$ for $1 \le i \le q$. With these pairs, the $\mathscr{A}$'s restriction stated in Definition 2 means that $\mathscr{A}$ is disallowed to output x where any subsequence of $y = \mathsf{PUF}(x)$ is included in $\{y_i\}_{i \in [1,q]}$.

2.2. PUF-Based Key Generation. We then review a definition of the PUF-based key generation (PBKG) scheme.

Definition 3 (PUF-based key generation, PBKG). A PBKG scheme consists of the following two phases.

(i) **Enrollment phase**: A key generation device in the PBKG scheme assigns a key and auxiliary data which help to reconstruct the key from a noisy PUF output in the next phase. The auxiliary data are stored into a public (maybe insecure) nonvolatile memory (NVM). Note that the device can generate the key inside itself or receive the key from outside.

(ii) **Reconstruction phase**: The key generation device regenerates the PUF output with some noises, loads the auxiliary data from the NVM, and reconstructs the key from these data.

We then review a definition of security for the PBKG scheme. As Datta et al. [15] discussed, the indistinguishability [16, 17] may be inadequate if the PBKG scheme is composed of another cryptographic protocol, such as an encryption scheme or message authentication code (MAC). This is because even though a key derived from the PBKG scheme is indistinguishable from a random string, the indistinguishability can be lost when the key is used, e.g., to encrypt a (partially) known message. In order to ensure the security for a combination of the PBKG scheme and a cryptographic protocol, we should customize the definition of its security by modeling an adversary to collect hints from both the PBKG scheme and the cryptographic protocol; namely, to access oracles of the scheme and the protocol, respectively. In this paper, we propose a PBKG scheme for a general purpose. Therefore, we discuss the indistinguishability [16, 17] as follows.

Definition 4 (indistinguishable PBKG). Let ℓ_k be a bit length of a key derived from a PBKG scheme. We say that the PBKG scheme is indistinguishable in (τ, q, ϵ) if any adversary $\mathscr{A}$, within the time bound τ, cannot achieve an advantage more than ϵ even if $\mathscr{A}$ invokes a key generation device with a (maybe intentionally modified) auxiliary data at most q times. Here, the advantage is defined with $\mathsf{Adv}_{\mathscr{A}}^{\mathsf{pbkg}} = |p_1 - p_2|$, where $p_1 = \Pr[\mathscr{A}(y) = 1 \mid y \longleftarrow \mathsf{PBKG}(1^{\ell_k})]$ and $p_2 = \Pr[\mathscr{A}(y) = 1 \mid y \longleftarrow \{0,1\}^{\ell_k}]$.

3. Previous Works and Their Drawbacks

This section reviews the PMKG and C-PMKG schemes and discusses their drawbacks.

3.1. PMKG Scheme. Figures 1 and 2 show the building blocks of the PMKG scheme [9] in its enrollment and reconstruction

phases, respectively. They use the following functions.

(i) PUF: Physically unclonable function which, given an ℓ_c-bit input x, outputs an ℓ_p-bit string y

(ii) CS: Challenge sequencer which, given at least one $\ell_i (= \lceil \log_2 \ell_p \rceil)$-bit index ind and a flag $flag$ as inputs, outputs the input of PUF x

(iii) KGF: Key generation function which, given a set of ℓ_i-bit indexes $\{ind_i\}_i$, outputs an ℓ_k-bit key key

(iv) pattern match: Comparator which, given ℓ_w-bit ($\ell_w < \ell_p$) auxiliary data stored in a public NVM and an ℓ_p-bit PUF output, looks for an index that leads an ℓ_w-bit substring of PUF output near to the auxiliary data. It may look for an index with which the distance between auxiliary data and substring indicated by the index is less than some predetermined threshold and/or with which the distance is the smallest among all substring candidates

Paral and Devadas [9] used the Arbiter-PUF as a building block which outputs only one bit per an input. In order to obtain an ℓ_p-bit output with the Arbiter-PUF, CS generates ℓ_p inputs $\{x_j\}_{j\in[1,\ell_p]}$ sequentially from index(es), instead of one input x as above, within each round. And then PUF returns ℓ_p-bit output in total with the ℓ_p inputs in the round. More precisely, we extend the above notations; we extend CS to output l_p inputs (by regarding x as $\{x_j\}_{j\in[1,\ell_p]}$), and we extend PUF to output an ℓ_p-bit concatenation $\mathsf{PUF}(x_1)\| \cdots \|\mathsf{PUF}(x_{\ell_p})$ for inputs $\{x_j\}_j$. Note that the above notations naturally cover the SRAM-PUF. Throughout this paper, for simply, we use the above notations regardless of the type of PUF.

The PMKG scheme uses four registers: an input register, an output register, an index register, and a key register, as depicted in Figures 1 and 2. The input register holds an input of PUF. The output register holds an output of PUF and it is randomly accessible. The index register holds a set of indexes. The key register holds a recovered key to pass it to the cryptographic protocol.

The PMKG scheme also uses the public NVM. In the enrollment phase, a key generation device stores a set of substrings of PUF outputs as a set of auxiliary data into the NVM. In the reconstruction phase, the key generation device loads the auxiliary data and uses it to reconstruct the key. Note that the NVM may be insecure and the stored auxiliary data may be modified by an adversary.

Protocol 1 (PMKG). Let PUF, CS, KGF, and pattern match be functions as above. A PMKG scheme consists of two phases: an enrollment phase and a reconstruction phase below.

Enrollment phase: A key generation device repeats a round operation r times with indexes $\{ind_i\}_{i\in[0,r]}$. The index ind_0 is an initial input of CS and we assume that it is fixed and stored into the NVM in advance. (Since the NVM is public, we assume that ind_0 is observable by adversaries. At the final steps of enrollment and reconstruction phases, we remove ind_0 from the input of KGF, because its observable value does not increase the entropy of a key.) Other indexes may be also fixed in advance or randomly generated inside the device, and we assume the former case here. This phase consists of the following steps.

(1) The key generation device sets $flag$ to one

(2) For $i \in [1, r]$, the device repeats the following steps:

(a) It inputs $(\{ind_k\}_{k<i}, flag)$ into CS to generate an input of PUF x_i

(b) It inputs x_i into PUF to generate a PUF output y_i

(c) It stores a substring of PUF output $z_i = y_i[ind_i : ind_i + \ell_w - 1]$ into the NVM. Here, $s[a : b]$ denotes a substring of s, consisting of its a-th to b-th bits

(3) It regards $\mathsf{KGF}(\{ind_i\}_{i\in[1,r]})$ as key

Reconstruction phase: The key generation device repeats a round operation r times to recover the indexes and eventually the key, assigned at the enrollment phase. Note that a PUF output in this phase may differ from that in the enrollment phase. We add prime marks to the variables in this phase. This phase consists of the following steps.

(1) The key generation device sets $flag$ to one

(2) For $i \in [1, r]$, the device repeats the following steps:

(a) It inputs $(\{ind'_k\}_{k<i}, flag)$ into CS to generate an input of PUF x'_i

(b) It inputs x'_i into PUF to generate a PUF output y'_i

(c) It loads auxiliary data z_i from the NVM

(d) It looks for an index ind'_i indicating the substring of y'_i (from ind'_i-th bit to $(ind'_i + \ell_w - 1)$-th bit) near to z_i from the pattern match. If a mismatch happens, namely, if such an index is not detected or if more than one index is detected, ind'_i and $flag$ are set with a constant value, e.g., ℓ_w and zero, respectively

(3) If $flag = 1$, then it inputs $\{ind'_i\}_{i\in[1,r]}$ into KGF to recover key. Otherwise, it aborts.

Paral and Devadas [9] regarded PUF as a 4-XOR PUF which blends outputs of four independent Arbiter-PUFs. The blended PUF decreases a correlation between inputs and outputs of PUF; and therefore it makes the machine learning attacks [18] meaningless. The more the number of blend increases, the more difficult the attack is. In this paper, we assume that the PUF is ideal and indistinguishable as in Definition 2 where there is no correlation between the inputs and the outputs. To realize such PUF, the blending is one of solutions.

The PMKG scheme has several drawbacks as follows. First of all, it requires lots of PUF output, and therefore it may be unsuitable for tiny devices in the IoT systems. The key generation device regards the ℓ_w-bit substring out of the ℓ_p-bit PUF output as auxiliary data. In other words, it discards

the remaining $(\ell_p - \ell_w)$-bit in each round. In [9], they assumed that, as an example, $\ell_p = 1279, \ell_w = 256$ and $r = 16$. With this example, the $(1279 - 256) \times 16 = 16368$ bits, out of $1279 \times 16 = 20464$ PUF output bits, are discarded. It makes the PMKG scheme inefficient, by requiring the execution time to generate the 16368 PUF output bits and the power consumption more.

In addition to its inefficiency, it has a security vulnerability, caused by attacks with an NVM manipulation. Such a manipulation leads mismatches in its reconstruction and these mismatches change the device's behavior. Delvaux and Verbauwhede [10, 11] gave attacks, named *snake attacks*, which modify the auxiliary data in the NVM, invoke the key generation with the modified one, and guess the index ind_i by observing its behavior.

Specifically, their attacks repeatedly modify the auxiliary data in the NVM to be (noncircularly) one bit shifted by guessing the next bit of PUF output. If the modified auxiliary data move out of the range of the PUF output with a large amount of shift, a mismatch happens and the device's behavior changes; until the amount is less than or equal to the distance to head or end, the device outputs a key, which should be different from the assigned one because inputs for CS and KGF differ; on the other hand, if it exceeds the distance, the device aborts because the mismatch happens. Namely, the snake attacks repeat the shift and guess by increasing its amount of shift until the device's behavior changes and return the amount as its guess for the secret index.

3.2. C-PMKG Scheme. To enhance the efficiency and/or the security, a circular PMKG (C-PMKG) scheme was proposed in [10, 19] independently and further discussed in [11, 20], respectively. It regards PUF outputs circularly shifted by secret indices $\{ind_i\}$ as auxiliary data.

Figures 3 and 4 show its building blocks. As for PUF, CS, pattern match, and KGF, we use the same notations as ones for the PMKG scheme, while we set $\ell_w = \ell_p$ in PUF. In addition to them, it also uses the function rotate:

(i) rotate: rotate function which, given an ℓ_w-bit string y and an amount of circularity $ind \in [0, \ell_w - 1]$, circularly shifts y by ind-bit

Protocol 2 (C-PMKG). Let PUF, CS, pattern match, and KGF be functions as in the PMKG scheme, except that the output length of the PUF is not ℓ_p-bit but ℓ_w-bit. Also let rotate be a function as above. The C-PMKG scheme consists of following two phases: an enrollment phase and a reconstruction phase below.

Enrollment phase: A key generation device repeats a round operation r times with indexes $\{ind_i\}_{i\in[0,r]}$. The index ind_0 is an initial input of CS and we assume that it is fixed in advance and stored in the NVM. Other indexes may be also fixed in advance or randomly chosen inside the device, and we assume the former case. This phase consists of the following steps.

(1) The key generation device sets *flag* to one

(2) For $i \in [1, r]$, the device repeats the following steps:

(a) It inputs $(\{ind_k\}_{k<i}, flag)$ into CS to generate an input of PUF x_i

(b) It inputs x_i into PUF to generate a PUF output y_i

(c) It stores a circularly shifted PUF output $z_i = \mathsf{rotate}(y_i, ind_i)$ into the NVM as auxiliary data

(3) It regards $\mathsf{KGF}(\{ind_i\}_{i\in[1,r]})$ as *key*

Reconstruction phase: The key generation device repeats a round operation r times to recover the indexes and eventually key, assigned at the enrollment phase. This phase consists of the following steps.

(1) The key generation device sets *flag* to one

(2) For $i \in [1, r]$, the device repeats the following steps:

(a) It inputs $(\{ind'_k\}_{k<i}, flag)$ into CS to generate an input of PUF x'_i

(b) It inputs x'_i into PUF to generate a PUF output y'_i

(c) It loads auxiliary data z_i from the NVM

(d) It looks for an index ind'_i indicating the circularity shifted string $\mathsf{rotate}(y'_i, ind'_i)$ near to z_i from the pattern match. If a mismatch happens, namely, if such an index is not detected or if more than one index is detected, ind'_i and $flag$ are set with a constant value ℓ_w and zero, respectively

(3) If $flag = 1$, then it inputs $\{ind'_i\}_{i\in[1,r]}$ into KGF to recover *key*. Otherwise, it aborts

Unlike the PMKG scheme, the C-PMKG scheme uses the whole (circularly shifted) PUF output as auxiliary data. Removing the discarding of PUF outputs enhances its efficiency compared to the PMKG scheme. Moreover, since there is no head or end in the circularity, the snake attacks are meaningless to the C-PMKG scheme.

However, in order for the C-PMKG scheme to be in use, there are technical issues to be considered. The most important one is a careful design of CS. The circularly shifted PUF outputs are stored in the public NVM as auxiliary data in the C-PMKG scheme; and, here, if some of them have intersections, the information of the corresponding indexes may be leaked. Hence, to ensure the security of C-PMKG scheme, CS should be designed leading to no intersection over rounds.

4. Single-Round Circular PMKG Scheme

In this section, we propose a simple PMKG scheme, named single-round circular PMKG (SC-PMKG), and show its security against manipulation attacks.

4.1. Our Idea. We construct the SC-PMKG scheme, based on the C-PMKG scheme, by improving the simplicity and security as follows.

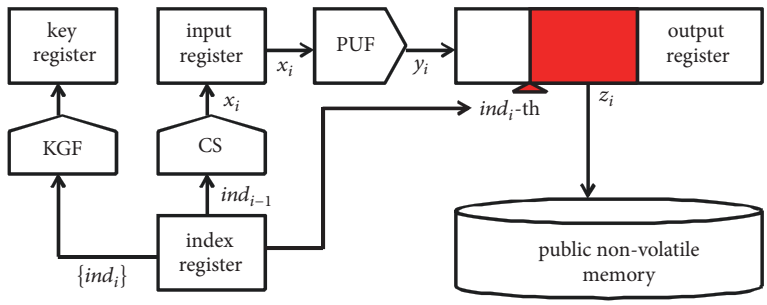

Figure 1: Building blocks for enrollment phase in PMKG.

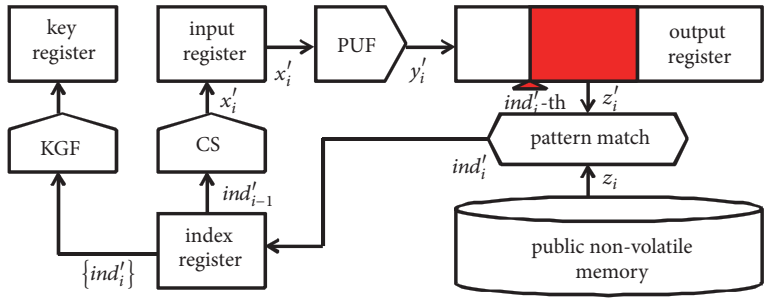

Figure 2: Building blocks for reconstruction phase in PMKG.

(i) *Simplicity*: Unlike the previous PMKG schemes with r round operations, the SC-PMKG scheme is performed within a single-round. In order to increase the entropy of key efficiently, the previous schemes prepare r PUF output strings with ℓ_w-bit each. On the other hand, the SC-PMKG scheme prepares an $(n \cdot \ell_w)$-bit PUF output string and divides it into n substrings with ℓ_w-bit each. From this change, there is no fear to overlap among the (sub)strings and the security discussion can be simplified. Moreover, the simple design helps us to develop devices.

(ii) *Security*: To detect and defeat the manipulation attack, we introduce a check string cs to confirm the correctness of the key reconstruction. The string is derived from a cryptographic hash function with which, by regarding it as a random oracle, we can theoretically prove the security of the SC-PMKG scheme.

The following subsections give our construction and the security consideration.

4.2. Construction. As for PUF, CS, rotate, pattern match, and KGF, we use the same notations for the C-PMKG scheme, except that we assume $\ell_p > \ell_w$ for PUF as in the PMKG scheme. Additionally, our scheme uses a hash function hash, to check the integrity of a reconstructed key.

(i) hash: Cryptographic one-way hash function which, given an ℓ_k-bit key, returns an ℓ_{cs}-bit check string cs

Protocol 3 (SC-PMKG). Let PUF, CS, rotate, pattern match, and KGF be functions as in the C-PMKG scheme, except that we assume $\ell_p = n \cdot \ell_w$ for an integer n, and let hash be a hash function as above. The SC-PMKG scheme consists of the following two phases: an enrollment phase and a reconstruction phase below.

Enrollment phase: A key generation device with indexes $\{ind_i\}_{i\in[0,n]}$ performs an enrollment as follows. The index ind_0 is an initial input for CS and we assume that it is fixed in advance and stored in the NVM. Other indexes may be also fixed in advance or randomly chosen inside the device, and we assume the former case. This phase consists of the following steps.

(1) The key generation device inputs ind_0 into CS to generate an input of PUF x

(2) It inputs x into PUF to generate a PUF output

(3) It divides the PUF output into n substrings $\{y_i\}_{i\in[1,n]}$ with ℓ_w-bit each

(4) It stores $\{z_i = \mathsf{rotate}(y_i, ind_i)\}$ for $i \in [1, n]$ into the NVM as auxiliary data

(5) It computes $key = \mathsf{KGF}(\{ind_i\}_{i\in[1,n]})$

(6) It computes $cs = \mathsf{hash}(ind_0, \{(z_i, ind_i)\}_{i\in[1,n]}, key)$ and stores it into the NVM as a check string

Reconstruction phase: The key generation device recovers the indexes and eventually the key, assigned at the enrollment phase. This phase consists of the following steps.

(1) The device sets $flag$ to one

(2) It inputs ind'_0 into CS to generate an input of PUF x'

(3) It inputs x' into PUF to generate a PUF output

(4) It divides the PUF output into n substrings $\{y'_i\}_{i\in[1,n]}$ with ℓ_w-bit each

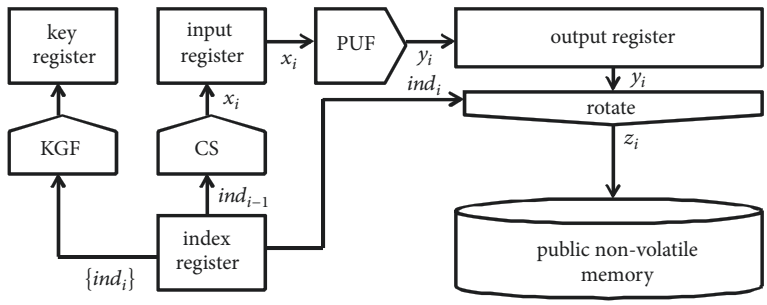

Figure 3: Building blocks for enrollment phase in C-PMKG.

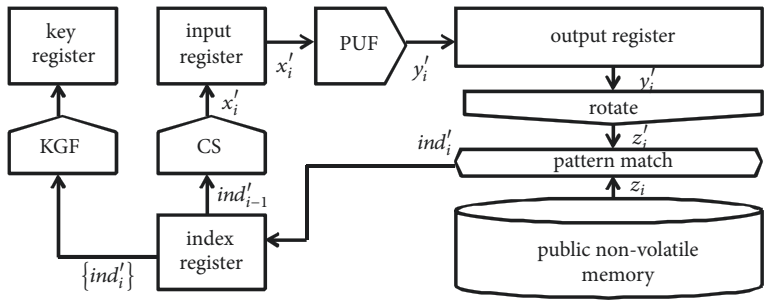

Figure 4: Building blocks for reconstruction phase in C-PMKG.

(5) For $i \in [1, n]$, it repeats the following steps:

  (a) It loads auxiliary data z_i' from the NVM

  (b) It looks for an index ind_i' indicating the circularity shifted string $\mathsf{rotate}(y_i', ind_i')$ near to z_i from the pattern match. If a mismatch happens, ind_i' and $flag$ are set with a constant value ℓ_w and zero, respectively

(6) If $flag = 1$, then it inputs $\{ind_i'\}_{i\in[1,n]}$ into KGF to compute key'. Otherwise, it aborts

(7) It loads a check string cs' from the NVM

(8) If $\mathsf{hash}(ind_0', \{(z_i', ind_i')\}_{i\in[1,n]}, key') = cs'$ holds, it regards key' as the reconstructed key. Otherwise, it aborts

There are two advantages in the SC-PMKG scheme. The first one is that, from its simple construction with a single-round, we do not need to pay attention to the intersections of auxiliary data over rounds. It simplifies the design of CS and the security consideration. The second one is that, by introducing the check string, the security against manipulation attacks of the NVM is ensured, if hash is an ideal hash function (random oracle [12]). We discuss the security in the following subsection.

4.3. Security Consideration. As in the C-PMKG scheme, since the auxiliary data is obtained from the circularly shift of the PUF output, the snake attacks are invalid for the SC-PMKG scheme. In addition, we can prove that the SC-PMKG scheme is an indistinguishable PBKG scheme in the random oracle model [12]. As for the security, the following theorem holds.

Theorem 5. *Assume that* PUF *is an indistinguishable PUF in* $(\tau, 0, \epsilon)$ *and that* hash *is the random oracle. Then, the SC-PMKG scheme is an indistinguishable PBKG in* (τ', q_K, ϵ') *where*

$$\epsilon' = \epsilon - \frac{1}{\ell_w^n}$$
$$\text{and } \tau' = \tau + q_K O(\mathsf{T}_K). \tag{1}$$

Here, T_K *denotes an execution time of the SC-PMKG scheme.*

The proof is done by the contradiction. Namely, we show that if there exists an adversary $\mathscr{A}$ against the SC-PMKG scheme, we can construct an algorithm $\mathscr{B}$, which uses $\mathscr{A}$ as a subroutine, to break the indistinguishability of PUF. The construction of $\mathscr{B}$ is simple and we give a proof in the Appendix.

Note that, in Theorem 5, we assume that $(\tau, 0, \epsilon)$ for the underlying PUF. The condition of $q = 0$ makes the PUF strongly protected within the system, lest no PUF output for an adversarial input is observable. However, there is a trade-off between the assumptions for the secure system ($q = 0$) and for a secure PUF ($q > 0$, see next paragraph). The condition we assume relaxes the choice of PUF because we can neglect the fear of the machine learning attacks [18], and because we can use a weak PUF with a small input-output space, such as SRAM-PUF.

We can extend the security model of Definition 4 so that an adversary against the PBKG scheme is allowed to

collect input-output pairs of the underlying PUF; that is, we can consider the security model with a powerful adversary. In this case, the indistinguishability of the PBKG scheme is defined with four-tuple $(\tau', q_K, q_P, \epsilon')$, instead of three-tuple (τ', q_K, ϵ'), where the adversary accesses the PBKG and PUF oracles at most q_K and q_P times, respectively. With this extension, we can prove another theorem, if the underlying PUF is indistinguishable in (τ, q_P, ϵ), instead of $(\tau, 0, \epsilon)$. If q_K is so large, a weak PUF may not be a possible choice for our system. Note that, even with this extension, the adversary of the SC-PMKG scheme should be restricted not to access the PUF oracle on x corresponding to the target key. To simplify the security proof, this paper discusses the security without the extension.

Regardless of whether the model is extended or not, we require the PUF so that its output is unobservable by invalid interface other than the interfaces in Definition 2. As for the SRAM-PUF, for example, to avoid the physical probing attack [21, 22], we should choose an appropriate PUF, with a fine process [23], etc.

4.4. Notes on Parameters. The SC-PMKG scheme is parameterized with the following parameters: ℓ_w, n, $\ell_p = n \cdot \ell_w$, $\ell_i = \log_2 \ell_w$, and ℓ_{cs}. If the pattern match in the reconstruction phase looks for the index with a threshold, the threshold th is also required.

Among them, n and ℓ_i (and, therefore, ℓ_w and ℓ_p) relate to the security. The (Shannon) entropy of key source is estimated by $n \cdot \ell_i$. To ensure the 160-bit entropy for the source, we set them so that $n \cdot \ell_i$ exceeds 160. In addition, ℓ_{cs} also relates to the security. The larger it is, up to $n \cdot \ell_i$, the harder the manipulation attack is.

On the other hand, ℓ_w and th relate to the correctness. There are two failure scenarios in reconstructing the key: the pattern match fails at Step (5)(b) or the hash value does not match the check string at Step (8). Note that if the indexes are correctly recovered at Step (5)(b), the verification at Step (8) should succeed. Hence, let us discuss the failure at Step (5)(b).

The pattern match at Step (5)(b) looks for an index with which the distance between a substring candidate and the auxiliary data is less than th and/or the smallest, as stated in Section 3.1. In order for the pattern match to be performed with only the former criterion, th should be set with adequate value (and additional criterion to narrow index candidates, if necessary) so that only the correct index is detected. With the latter criterion, it is easy to see that the index can be correctly recovered if ℓ_w is large. Note that, under the condition that each bit of PUF output is independent of other bits, the distances for correct and incorrect indexes are expected about $p_e \cdot \ell_w$ and $\ell_w/2$, where p_e is a bit error rate, respectively. Namely, if ℓ_w increases, the gap between these distances also increases and the correct index is detectable. In the next section, we check the relation between ℓ_w and the correctness of the recovered index (key). The combination of the first and second criteria, first finding index candidates with a loose threshold and then detecting the index with the smallest distance, enables not only the recovery of correct index but also the detection of system errors from faults and manipulation attacks.

5. Feasibility Tests

In this section, we test the feasibility of the SC-PMKG scheme by simulations and experiments, respectively.

5.1. Simulation. We first check the feasibility of the SC-PMKG scheme by a simulation with an approximation analysis. In this simulation, we estimate the failure probability in the key reconstruction by changing parameters ℓ_w, n, and p_e where p_e is a bit error rate in the PUF output. Following Paral and Devadas [9], for each ℓ_w, we set n so that the entropy of input for KDF is 160-bit. For example, if we use $\ell_w = 32$, we set $n = 32$ as a minimum integer so that $n \log_2 \ell_w \geq 160$.

Table 1 summarizes the failure probability for $p_e \in \{0.15, 0.035\}$ and ℓ_w from 32 to 160. Here, we assume 0.15 and 0.035 for p_e because $p_e = 0.15$ is a well discussed parameter for PUF instantiations and $p_e = 0.035$ is an average of bit error rate in our experiments below. As for n, we set n for each ℓ_w by a minimum integer as in the above case for $\ell_w = 32$. With ℓ_w and n, $\ell_p = \ell_w \times n$ is a length of the PUF output. For each p_e and ℓ_w, the "prob. (sim)" shows the failure probability from the simulation, which is estimated as shown in Algorithm 1.

For each ℓ_w, we tried $N = 5,000,000$ reconstructions, in total. (In Table 1, "0" means that there is no error for $N = 5,000,000$, which means that prob. (sim) is less that $1/N = 2 \times 10^{-7}$.) From our simulation, $\ell_w = 160$ ($p_e = 0.15$) or $\ell_w = 48$ ($p_e = 0.15$) should be enough for the key reconstruction with the failure probability less than 10^{-6}.

We also estimate the failure probability by an approximation analysis, as shown in "prob. (approx)" in Table 1.

Let us discuss the distance between a substring stored in the NVM and one circularly shifted by the *correct* index. The distance follows the binomial distribution $B(\ell_w, p_e)$. We approximate it with the normal distribution $N(\ell_w p_e, \ell_w p_e(1-p_e))$. Similarly, we approximate the distribution of the distance related to an *incorrect* index with the normal distribution $N(\ell_w/2, \ell_w/4)$.

In case the distance for the *correct* index is more than one for the *incorrect* index, the wrong index is recovered. Let us denote the probability by q. The difference between the distances for correct and incorrect indexes follows the normal distribution $N(mean, var)$ where $mean = \ell_w p_e - \ell_w/2$ and $var = \ell_w p_e(1-p_e) + \ell_w/4$, which is a composite (difference) of two normal distributions. With this notation, q can be estimated by

$$q \approx \int_{mean/\sqrt{var}}^{\infty} \frac{1}{\sqrt{2\pi}} \exp\left(\frac{x^2}{2}\right) dx. \tag{2}$$

Note that there are $\ell_w - 1$ wrong candidates for one index. If q is small enough, the probability where one of wrong index is recovered is estimated by $1-(1-q)^{\ell_w - 1} \approx 1-\{1-(\ell_w-1)q\} = (\ell_w - 1)q$. Also note that there are n indexes to be recovered. Similarly, the probability where one of index is incorrectly recovered among n substrings is approximated by $(\ell_w - 1)nq$. "prob. (approx)" in Table 1 summarizes the failure probability for each p_e, ℓ_w (and q), and n.

Table 1: Failure probability in SC-PMKG ((a) $p_e = 0.15$, (b) $p_e = 0.035$).

(a)

ℓ_w	n	ℓ_p	prob. (sim)	prob. (approx)
32	32	1024	0.538	0.630
48	29	1392	8.02×10^{-2}	5.40×10^{-2}
64	27	1728	8.29×10^{-3}	4.41×10^{-3}
96	25	2400	9.02×10^{-5}	2.83×10^{-5}
128	23	2944	1.00×10^{-6}	1.69×10^{-7}
160	22	3520	0	1.01×10^{-9}

(b)

ℓ_w	n	ℓ_p	prob. (sim)	prob. (approx)
8	53	424	0.682	0.242
16	40	640	6.86×10^{-2}	7.51×10^{-3}
24	34	816	2.73×10^{-3}	1.45×10^{-4}
32	32	1024	1.29×10^{-4}	2.54×10^{-6}
48	29	1392	2.00×10^{-7}	6.27×10^{-10}
64	27	1728	0	1.36×10^{-13}

Enrollment phase:
(1) Generate an ℓ_p-bit string at random.
(2) Divide the string into n substrings $\{y_i\}_{i\in[1,n]}$ with ℓ_w-bit each.
(3) For $i \in [1, n]$, choose $ind_i \in [0, \ell_w - 1]$ at random.
(4) For $i \in [1, n]$, compute $z_i = \mathsf{rotate}(y_i, ind_i)$.
(5) Store $\{(y_i, ind_i, z_i)\}_{i\in[1,n]}$.
Reconstruction phase:
(1) Set *cnt* to 0.
(2) Repeat the following steps N times:
  (a) For $i \in [1, n]$, generate an ℓ_w-bit string e_i where each bit of e_i is 1 with probability p_e, independently.
  (b) For $i \in [1, n]$, set $y'_i = y_i \oplus e_i$ and look for ind'_i where $\mathsf{rotate}(y'_i, ind'_i)$ is near to z_i.
  (c) Increment *cnt* if there exists $i \in [1, n]$ such that $ind_i \neq ind_i$.
(3) Compute the failure probability by cnt/N.

Algorithm 1

Since the approximations are inappropriate for small parameters, there are gaps between the failure probabilities from simulation and our approximation analysis. However, our analysis seems valid for large ℓ_w; namely, $\ell_w = 160$ and $\ell_w = 64$ are large enough, when p_e is 0.15 and 0.035, respectively, so that the failure probability is negligible.

5.2. Experiments with Nucleo Boards. We then test the feasibility with two Nucleo-F401RE boards (http://www.st.com/en/evaluation-tools/nucleo-f401re.html (accessible on Nov. 9, 2018)) by STMicroelectronics. We load $\ell_p = n \cdot \ell_w$ initial SRAM bits on each board and regard them as an output of SRAM-PUF. The experiments are performed in the room temperature and the power is supplied by the USB interface. (The characteristics of PUF would change by environments such as temperature and supplied power voltage. Refer to Chapter 4 of [24], for instance. The success rate of the key reconstruction depends on the bit error rate of PUF. We should select adequate parameters, such as ℓ_w and n, with the worst error rate among the different environments.) The averages of bit error rate for 4,096 SRAM bits on these boards are 3.30% and 3.57%, respectively. Here, the bit error rate is evaluated, in a simple manner, by a difference between the bit for the first read and bits for the following 10,000 reads.

Similar to the above simulation, we estimate the failure probability by changing ℓ_w. For each board, we try the reconstruction 10,000 times. Table 2 summarizes the failure probabilities for two boards, respectively. (Similar to Table 1, "0" means that there is no error for $N = 10,000$ and that the failure probability is less than 10^{-4}.)

For small ℓ_w, the probabilities differ from those of Table 1. This is because, different from the simulation, the initial values of SRAM cells are not uniformly random, and their stabilities are not constant for each bit. With larger ℓ_w, the probabilities in our experiment are close to those in the simulation. Note that since the initial values of SRAM cells are not uniform, if we use this SRAM as SRAM-PUF directly, the resulting SC-PMKG scheme might be insecure. From our observations, the initial values of SRAM cells have certain

Table 2: Failure probability in SC-PMKG with Nucleo-F401RE boards.

ℓ_w	n	ℓ_p	board 1	board 2
8	53	424	1	1
16	40	640	0.137	0.434
24	34	816	8.00×10^{-4}	4.20×10^{-3}
32	32	1024	1.00×10^{-4}	1.00×10^{-4}
48	29	1392	0	0
64	27	1728	0	0

bit patterns; and therefore regarding either a concatenation of first bits of each byte or an XORed value of different cells like 4-XOR PUF as an output of the SRAM-PUF may be a solution to enhance the security of the SC-PMKG scheme.

5.3. Experiments with Open Dataset. In order to check the feasibility with a different type of PUF, we use the ring oscillators' (ROs') frequencies which are available at the Secure Embedded Systems (SES) Lab at Virginia Tech (http://rijndael.ece.vt.edu/puf/download.html (accessible on Nov. 9, 2018)). We select "Full Standard Dataset Oscillator Counts." The dataset includes 100 oscillator counts for 512 ROs of 193 Xilinx Spartan-3E FPGA boards (XC3S500E) under the standard temperature/voltage. Using these ROs' counts, up to $\binom{512}{2} = 130816$ bits of RO-PUF can be derived. We implement the SC-PMKG with some of them. The average bit error rate for the first 1728 RO-PUF bits is 1.21%.

In the dataset, there are only 100 samples for each board. In the feasibility test, we evaluate the average and maximum of failure probabilities over 193 boards. Table 3 summarizes them. (Similar to Table 2, "0" means that the failure probability is less than $1/(99 \times 193) < 5.23 \times 10^{-5}$.)

Since the bit error rate (1.21%) is smaller than that of Nucleo boards (approx. 3%), the failure probabilities are also small. That is, for the correctness, fewer PUF output bits are required, for example, $\ell_p = 1024$.

6. Discussion

Table 4 summarizes the comparison among the PMKG, C-PMKG, and SC-PMKG schemes. Let us discuss the detail of each item below. We refer to [9, 25] for parameters of the PMKG and C-PMKG schemes, respectively. As for C-PMKG, we also add parameters, in parentheses, which we estimate by the similar theoretical approximation of Section 5.1. Note that the authors of [9, 25] decided the parameters so that the failure in key reconstruction should be negligible (with probability less that 10^{-6}).

"|Key|" is a bit length of an output of KGF. References [9, 25] set it to 128, and we follow them for the SC-PMKG scheme. In order to generate the 128-bit key, the previous works [9, 25] assumed that the source of the key had the 160-bit entropy in preparation for the entropy loss of PUF output and the randomness of indexes; i.e., the sum of index lengths was supposed to be 160-bit. We also follow them for the SC-PMKG scheme and give it in the second line "H(Source)" as the entropy of the source.

"H(Index)" shows the entropy of an index. In the PMKG scheme, the index is an address of the first bit of substring. Paral and Devadas [9] set it with 10. In the C-PMKG and SC-PMKG schemes, on the other hand, the index is the amount of circular shift which is at most the length of the substring. Reference [25] followed [9] to set the length of the substring with 256, and therefore H(Index) was set with 8. On the SC-PMKG scheme, from our simulation and experiments, the length of substring seems enough with 160 and we set H(Index) with $\log_2 160$. As in parentheses, the length of substring for the C-PMKG scheme can be similarly estimated with our approximation estimation, so that the failure probability is less than 10^{-6}.

"♯rounds" is a number of rounds. On the PMKG scheme, to generate (ℓ_k =)128-bit key with (ℓ_i =)10-bit index per round, 13 rounds may be enough. However, Paral and Devadas [9] used 16 to make the entropy of source 160-bit. Following them, we set the number of rounds to make the entropy of source 160-bit or more. Similarly, in the C-PMKG scheme, since the length of each index is eight, it is $160/8 = 20$ (or $\lceil 160/\log_2 160 \rceil = 22$ with our approximation). As for the SC-PMKG scheme, the number of rounds is one.

"♯indexes/rounds" are a number of indexes per a round. In the PMKG and C-PMKG schemes, they are one. On the other hand, in the SC-PMKG scheme, it is $\lceil 160/\log_2 160 \rceil = 22$ to achieve the 160-bit entropy.

"|Pattern|" is a bit length of a (sub)string of a PUF output to be pattern-matched. It is the same as the bit length of each auxiliary data. They set it to 256 which is expected to avoid the mismatch. As for the SC-PMKG scheme, we set it to 160 from our simulation. As in parentheses, the length of |Pattern| (and ones of items below) for the C-PMKG scheme can be similarly estimated as ours, with the approximation estimation.

"|PUF output|/round" is a bit length of PUF output in each round. They set it with 1279(= 1024 + 256 − 1) and 256, respectively. As for the SC-PMKG scheme, since it requires 22(= n) substrings of 160-bit in a round as we discuss, the length is $22 \times 160 = 3520$.

"|Total PUF output|" is a bit length of PUF output in total, which is $\ell_p \times r \times n$.

"|Storage|" means the total length of stored data in the NVM, except ind_0. The total length of auxiliary data is estimated with $\ell_w \cdot r \cdot n$. In the SC-PMKG scheme, beside

Table 3: Failure probability in SC-PMKG with ring oscillator (open dataset).

ℓ_w	n	ℓ_p	prob. (ave)	prob. (max)
8	53	424	0.592	1
16	40	640	0.461	1
24	34	816	0	0
32	32	1024	0	0

Table 4: Comparison of PMKG schemes.

Item	PMKG [9]	C-PMKG [25]	SC-PMKG
|Key|, ℓ_k	128	128	128
H(Source)	160	160	160
H(Index)	10	8 ($\log_2 160$)	$\log_2 160$
♯rounds, r	16	20 (22)	1
♯indexes/round, n	1	1	22
|Pattern|, l_w	256	256 (160)	160
|PUF output|/round, ℓ_p	1279	256 (160)	3520
|Total PUF output|	20464	5120 (3520)	3520
|Storage|	4096	5120 (3520)	3776
Additional functions over PMKG	–	rotate	rotate, hash
CS design	Complicated	Complicated	Simple
Snake attacks	Vulnerable	Secure	Secure
Provable security	Unknown	Unknown	Yes

the auxiliary data, the check string is stored in the NVM. We assume that SHA-256 is used as hash and the length is 3520 + 256 = 3776.

Although the C-PMKG and SC-PMKG schemes require fewer PUF outputs in total, they require additional functions, which derive an implementation and computation overhead, compared to the PMKG scheme. In the PMKG scheme, the index search can be performed sequentially. On the other hand, the C-PMKG and SC-PMKG schemes require the additional function rotate. In addition, the SC-PMKG scheme requires the hash function to compute the check string.

The challenge sequencers CS for the PMKG and C-PMKG schemes are carefully designed to avoid an intersection of auxiliary data over rounds as we discussed; however, its design for the SC-PMKG scheme can be simple.

As for the security, the PMKG scheme is vulnerable against the snake attacks. The C-PMKG and SC-PMKG schemes are secure against the snake attacks. Let us discuss the provable security. Although [20] discussed the provable security of the C-PMKG scheme, the security model did not take the manipulation attacks into consideration. It remains an open problem whether the C-PMKG scheme is provably secure without a hash function which, in the SC-PMKG, plays an important role in the provable security. On the other hand, the SC-PMKG scheme is provably secure as in Theorem 5.

To summarize, compared to the PMKG scheme, the C-PMKG and SC-PMKG schemes are not only secure but also efficient; namely, they require less PUF output. In addition, the simple CS design to generate a challenge in the single-round of the SC-PMKG scheme allows us to use even a weak PUF, like the SRAM-PUF, as a building block. It is an advantage for applications with tiny devices, such as the IoT systems.

7. Conclusions

This paper proposes a secure and efficient PMKG scheme, the SC-PMKG scheme, which saves the PUF output and allows a simple construction. It can be realized with even a weak PUF, such as the SRAM-PUF, and is suitable for tiny devices in the IoT systems. Feasibility test with other types of PUF changing the environments is one of our future works.

Appendix

A. Proof of Theorem 5

Proof sketch: We construct $\mathscr{B}$ as follows.

(1) $\mathscr{B}$ randomly generates $ind_0, ind_1, \ldots, ind_n$, where ind_0 is with prefixed length and $ind_i \in [0, \ell_w - 1]$ for $i \in [1, n]$.

(2) $\mathscr{B}$ computes an input $x = \mathsf{CS}(ind_0)$.

(3) $\mathscr{B}$ outputs x as a challenge to receive $y \in \{0,1\}^{\ell_p}$ as a target. If $\mathscr{B}$ is in the real game (the real game is a challenge to $\mathscr{B}$ whether $\mathscr{B}$ can guess that the key is correctly generated; it corresponds to $p_1 = \Pr[\mathscr{A}(y) = 1 \mid y \longleftarrow \mathsf{PBKG}(1^{\ell_k})]$ in Definition 4), y is an output $\mathsf{PUF}(x)$; otherwise, if $\mathscr{B}$ is in the random game (the random game is a challenge to $\mathscr{B}$ whether

$\mathscr{B}$ fails to guess that the key is randomly generated; it corresponds to $p_2 = \Pr[\mathscr{A}(y) = 1 \mid y \longleftarrow \{0,1\}^{\ell_k}]$ in Definition 4), y is a random string over $\{0,1\}^{\ell_P}$. The goal of $\mathscr{B}$ is to distinguish these games.

(4) $\mathscr{B}$ proceeds to steps from (3) to (6) of the enrollment phase in Protocol 3, by regarding y as an output of PUF. Note that, in the random oracle model, $cs = \mathsf{hash}(ind_0, \{(z_i, ind_i)\}_{i\in[1,n]}, key)$ is defined by the random oracle through an oracle query.

(5) $\mathscr{B}$ gives $\mathscr{A}$ key as a challenge.

(6) For a query from $\mathscr{A}$ to the random oracle, $\mathscr{B}$ answers it as follows.

  (a) If the query includes $\{ind_i\}_{i\in[1,n]}$ which $\mathscr{B}$ chose at Step (1) above, $\mathscr{B}$ aborts and fails the proof.
  (b) If the query does not include $\{ind_i\}_{i\in[1,n]}$ which $\mathscr{B}$ chose at Step (1) above, $\mathscr{B}$ passes the query/response between $\mathscr{A}$ and the oracle.

(7) For a query from $\mathscr{A}$ to the SC-PMKG scheme as a PBKG scheme, $\mathscr{B}$ answers it as follows.

  (a) If ind_0, $\{z_i\}_{i\in[1,n]}$, or cs are not modified, $\mathscr{B}$ returns key to $\mathscr{A}$.
  (b) If at least one of ind_0, $\{z_i\}_{i\in[1,n]}$, and cs is modified with different values, $\mathscr{B}$ returns nothing to $\mathscr{A}$.

(8) If $\mathscr{A}$ returns $b \in \{0,1\}$, $\mathscr{B}$ returns b.

In the above construction, $\mathscr{B}$ fails the proof at Step (6)(a), with probability $1/\ell_w^n$. Otherwise, if $\mathscr{A}$ succeeds in distinguishing the games for the SC-PBKG scheme, $\mathscr{B}$ also succeeds in distinguishing the games for the PUF. Therefore, we have the theorem.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Number JP18H05289. We sincerely thank Dr. Jeroen Delvaux for his invaluable comments.

References

[1] J. Guajardo, S. S. Kumar, G. J. Schrijen, and P. Tuyls, "FPGA intrinsic PUFs and their use for IP protection," in *Cryptographic Hardware and Embedded Systems - CHES, 2007., Volume 4727 of Lecture Notes in Computer Science*, P. Paillier and I. Verbauwhede, Eds., pp. 63–80, Springer-Verlag, Berlin, Germany, 2007.

[2] D. E. Holcomb, W. P. Burleson, and K. Fu, "Initial SRAM state as a fingerprint and source of true random numbers for RFID tags," in *Proceedings of the Conference on RFID Security 2007, IEEE*, 1210 pages, 2007.

[3] D. E. Holcomb, W. P. Burleson, and K. Fu, "Power-up SRAM state as an identifying fingerprint and source of true random numbers," *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 58, no. 9, pp. 1198–1210, 2009.

[4] B. Gassend, D. Lim, D. Clarke, M. van Dijk, and S. Devadas, "Identification and authentication of integrated circuits," *Concurrency and Computation: Practice and Experience*, vol. 16, no. 11, pp. 1077–1098, 2004.

[5] J. W. Lee, D. Lim, B. Gassend, G. E. Suh, M. Van Dijk, and S. Devadas, "A technique to build a secret key in integrated circuits for identification and authentication applications," in *Proceedings of the Symposium on VLSI Circuits (VLSI '04)*, pp. 176–179, Honolulu, Hawaii, USA, June 2004.

[6] D. Lim, *Extracting secret keys from integrated circuits [Msc. thesis]*, Institute of Technology (MIT), Massachusetts, Mass, USA, 2004.

[7] Y. Dodis, R. Ostrovsky, L. Reyzin, and A. Smith, "Fuzzy extractors: how to generate strong keys from biometrics and other noisy data," *SIAM Journal on Computing*, vol. 38, no. 1, pp. 97–139, 2008.

[8] C. Herder, L. Ren, M. Van Dijk, M.-D. Yu, and S. Devadas, "Trapdoor Computational Fuzzy Extractors and Stateless Cryptographically-Secure Physical Unclonable Functions," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 1, pp. 65–82, 2017.

[9] Z. Paral and S. Devadas, "Reliable and efficient PUF-based key generation using pattern matching," in *Proceedings of the 2011 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2011*, pp. 128–133, USA, June 2011.

[10] J. Delvaux and I. Verbauwhede, "Attacking PUF-based pattern matching key generators via helper data manipulation," in *Topics in Cryptology – CT-RSA 2014*, vol. 8366 of *Lecture Notes in Computer Science*, pp. 106–131, Springer International Publishing, Cham, 2014.

[11] J. Delvaux and I. Verbauwhede, "Attacking PUF-based pattern matching key generators via helper data manipulation," in *Topics in cryptology—CT-RSA 2014*, vol. 8366 of *Lecture Notes in Comput. Sci.*, pp. 106–131, Springer, Cham, 2014.

[12] M. Bellare and P. Rogaway, "Random oracles are practical," in *Proceedings of the the 1st ACM conference*, pp. 62–73, Fairfax, Virginia, United States, November 1993.

[13] R. Maes and I. Verbauwhede, "A discussion on the properties of. physically unclonable functions," in *Proceedings of the 3rd International Conference on Trust and Trustworthy Computing (TRUST 2010)*, 2010.

[14] A. R. Sadeghi, *Towards Hardware-Intrinsic Security*, 2010.

[15] A. Datta, A. Derek, J. C. Mitchell, and B. Warinschi, "Key exchange protocols: Security definition, proof method and applications," https://eprint.iacr.org/2006/056.pdf.

[16] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Advances in Cryptology—(CRYPTO '93)*, vol. 773 of *Lecture Notes in Computer Science*, pp. 232–249, Springer, Berlin, Germany, 1994.

[17] M. Bellare, R. Canetti, and H. Krawczyk, "Modular approach to the design and analysis of authentication and key exchange protocols," in *Proceedings of the 1998 30th Annual ACM Symposium on Theory of Computing*, pp. 419–428, May 1998.

[18] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber, "Modeling attacks on physical unclonable functions," in *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10)*, pp. 237–249, Chicago, Ill, USA, October 2010.

[19] Y. Komano, K. Ohta, and K. Sakiyama, "Encryption key generating apparatus and computer program product (Jan. 2017) Original Japanese patent(5,710,460) was filed on Dec. 16, 2011".

[20] Y. Komano, K. Ohta, K. Sakiyama, and M. Iwamoto, "Provably secure pattern matching key generation using PUF," in *Proceedings of the SCIS 2012, The 2012 Symposium on Cryptography and Information Security*, 2012 (Japanese).

[21] C. Helfmeier, C. Boit, D. Nedospasov, and J.-P. Seifert, "Cloning physically unclonable functions," in *Proceedings of the 2013 6th IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2013*, pp. 1–6, USA, June 2013.

[22] D. Nedospasov, J. Seifert, C. Helfmeier, and C. Boit, "Invasive PUF Analysis," in *Proceedings of the 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pp. 30–38, Los Alamitos, CA, USA, August 2013.

[23] H. Mori, T. Nakagawa, Y. Kitahara et al., "An low-energy 8T dual-port SRAM for image processor with selective sourceline drive scheme in 28-nm FD-SOI process technology," in *Proceedings of the 23rd IEEE International Conference on Electronics, Circuits and Systems, ICECS 2016*, pp. 532–535, Monaco, December 2016.

[24] R. Maes, *Physically Unclonable Functions: Constructions, Properties and Applications*, Springer, New York, NY, USA, 2013.

[25] Y. Iwai, T. Fukushima, D. Moriyama et al., "Implementation and evaluation of PUF-based pattern matching key generation using circular shift," in *Proceedings of the SCIS 2013, The 2013 Symposium on Cryptography and Information Security*, 2013.