Hindawi
Security and Communication Networks
Volume 2019, Article ID 5935292, 15 pages
https://doi.org/10.1155/2019/5935292

Research Article

A Generic Multifactor Authenticated Key Exchange with Physical Unclonable Function

Jin Wook Byun

Department of Information and Communication, Pyeongtaek University, Yongi-dong 111, Pyeongtaek-si, Gyeonggi-do, Republic of Korea

Correspondence should be addressed to Jin Wook Byun; jwbyun@ptu.ac.kr

Received 31 October 2018; Revised 4 January 2019; Accepted 8 January 2019; Published 4 March 2019

Academic Editor: Bela Genge

Copyright © 2019 Jin Wook Byun. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

We study how to generally construct a PUF-based multifactor authenticated key exchange (MAKE) protocol from any secure password-based authenticated key exchange protocol. We newly consider a new setting in which a user holding a PUF-embedded device desires to perform an authenticate key exchange through multifactor authentication factors (password, biometrics, secret). Our construction is the first PUF-based general MAKE construction. By applying a PUF in a device, our MAKE construction is still secure even if all authentication factors are totally compromised. Our construction is the first PUF-based generic MAKE protocol requiring additional three communication flows without any public key based primitives such as signature.

1. Introduction

To establish a secure communication over public network, a secure exchange of secret key with authentication is an absolute prerequisite. Thus, authenticated key exchange (for short, AKE) has been one of the most important cryptographic primitives. Basically, for practical and secure AKE, a memory requirable long-term secret key, a human memorable password, and human-dependent biometric information have been utilized as authentication factors in the AKE. If the AKE protocol is designed with only one factor (e.g., secret key or password) for authentication, it is considered as a one-factor AKE. Bellare and Rogaway first formalized security notions and definitions for the one-factor AKE based on symmetric secret key [1]. Subsequently, Bellare et al. have established a new security model in case of memorable password [2] (for short, PAKE). In a span of twenty years, these two models not only have been the foundation for subsequent PAKE results achieving augmented provable security with efficiency [3, 4], but also have been bases for the extended model such as group, three-party, and multifactor (secret, password, biometrics) authentication settings [5–9]. In this paper, we aim to design a generic framework to transform any secure PAKE into a secure MAKE with physical unclonable function. Before discussing our motivations, let us briefly summarize relevant results on provably secure AKE and MAKE constructions.

If two authentication factors (secret key and password) are simultaneously used into an AKE protocol, we consider it as a two-factor AKE (for short, TAKE) protocol. With the rapid popularization of smartcard, there have been huge amounts of research on TAKE in a practical scenario where a user memorizes own password and possesses a secret key in each own smartcard [10–16]. In a smartcard setting, it is impossible to list all (or even some specific related works and topics) works here. One main research stream is that many protocols have been found to be flawed and then finally redesigned repeatedly. Furthermore, our paper does focus on not the TAKE, but the MAKE that handles multifactor AKE. However, in the TAKE, there have been good relevant works on generic construction. First, Yang et al. [17] have designed a framework to transform a secure password-based PAKE [18] suggested by Halevi and Krawczyk into the TAKE protocol. The main idea is to combine the HK password-based PAKE with signature-based authenticator, which finally enables achieving a general TAKE scheme. However, it is not general construction since the scheme is only able to transform

the HK scheme into two-factor scheme, and is not be able to handle all kinds of secure PAKE for building TAKE. In 2010, Stebila et al. suggested multifactor password-based AKE [19]. It has been designed to allow multiple kinds of passwords. More precisely, this protocol deals with multiple passwords for authentication (note that biometrics is not included at all). Hence it is fair to say that it falls into the TAKE protocol. Besides, the protocol relies on the PKI environment and random oracle model for security proof. In 2014, Huang et al. have presented new types of adversaries with precomputed and different data on the smart card in a TAKE setting [20]. Two TAKE schemes [21, 22] have been analyzed and securely revised under the new adversaries [20]. In 2018, Wang and Wang have syntagmatically and universally defined an explicit and new security model with practical adversaries' capabilities for a TAKE setting. Also, a new and secure TAKE scheme, satisfying all security criteria, has been suggested. Very recently, Wang et al. have also proposed a new framework for evaluating security, efficiency, and scalability for a TAKE wireless sensor network [23]. Under the framework, two existing schemes [24, 25] have been found to be flawed [23]. Furthermore, 44 TAKE schemes have been universally compared and analyzed. It is one of the systematic comparisons over such vast amounts of research.

A multifactor AKE (for short, MAKE) deals with three types of authentication factors (secret, password, and biometrics). The main advantage of MAKE over TAKE is its ability to preserve the security even if one or two authentication factors are compromised. Despite its merits, relatively, the MAKE has been received less attention in the literature than the TAKE. In 2008, Pointcheval and Zimmer have designed a provably secure MAKE protocol (for short, PZ) under computational Diffie-Hellman assumption [26]. It uses an ElGamal public key encryption [27] for securing biometric information. However, in 2012, it has been found to be insecure by Hao and Clarke [28]. The insecurity of PZ is that an attacker, who steals a password, is able to deduce biometric information. After the PZ, there have been attempts to design a MAKE in smartcard, cloud computing, three-party, and multiserver environments [29–32]. In 2009, Fan et al. have proposed a public key based MAKE in a smartcard setting [29]. It protects users' biometric data from anyone else including a server while allowing secure authentication. Later, the PZ has been extended into three-party setting by Liu et al. [30]. It has been the first work on addressing the three-party setting that assumes an intervention of trusty party between a client and a server. A new MAKE based on elliptic curve has been suggested in a multiserver setting by He et al. in 2015 [31]. Very recently, Wei et al. have presented a privacy-preserving multifactor authenticated key exchange for cloud computing [32]. It has achieved better efficiency than the PZ in terms of computation and bandwidth costs.

1.1. Related Works. Our goal is to construct a secure generic PUF-based MAKE protocol from any secure PAKE protocol. To the best of our knowledge, a PUF-based MAKE protocol has not been proposed yet in the literature. However, regarding generic MAKE constructions, which are not based on PUF, there have been few results in recent years. First, in 2011, Huang et al. have designed a generic transformation that combines any secure PAKE in the smartcard setting with biometrics for building a MAKE [33]. Their main idea is to execute a TAKE twice. They use a fuzzy extractor to handle user's biometrics. Later, it has been enhanced by Yu et al. in 2014 [34]. Its enhanced version applied a fuzzy vault based on euclidian distance for handling biometric features. And also, it used a TAKE just once. Despite its efficiency and novelty, these two constructions have been built under a PKI setting where participants must be allowed to sign (verify) and encrypt (decrypt) based on public key cryptography. Moreover, one critical problem is that these two constructions [33, 34] have not been built in a generic way. For example, in [34], they first make $pw = H(pw_1 \parallel bio)$ and apply pw into a TAKE where pw_1 is a password, bio is a biometric information, and H is a cryptographic secure hash function. Depending on what kind of TAKE is being used, an input of $pw(= H(pw_1 \parallel bio))$ is not always an input for TAKE. For instance, if TAKE uses a structure g^{pw} (or $E_{pw}(\cdot)$) inside, then, before application, $pw(= H(pw_1 \parallel bio))$ should be customized as $f(pw)$ where f is a function that converts an input to a cyclic group element output (or fixed length output, e.g., 128 bits, 256 bits). In [34], they select a good but specific TAKE [17] and demonstrate how it can be initiated to build a final MAKE. It does not mean that their generic MAKE is able to cover all cases of TAKE. One important observation from this issue is that a generic MAKE should execute a TAKE first and then use its session key for integrating multifactor authenticators rather than executing TAKE with an integration of multifactor authenticators.

Fleischhacker et al. have presented a generic MAKE construction (for short, FMA) by combining subprotocols such as unauthenticated key exchange, password-based authentication (password), public key authentication (secret key), and biometric authentication protocols (biometric) [35]. The main idea is that it uses subprotocols independently for building a general MAKE. This approach guarantees not only better security but also simplicity for implementation. However it does not allow two authentication factors to be used together for one subprotocol; hence it has a limitation to enhance the total communication cost.

1.2. Motivation. In the paper, we firstly attempt to use a physical unclonable function (PUF) for generally building a secure MAKE. The PUF is basically embedded in a device at manufacturing stage. It always produces an unpredictable output depending on device's physical status, and it cannot be duplicated or cloned over a new device. This unpredictability has been one of the good properties for enhancing security in cryptology. However, if the PUF always outputs different unpredictable result, then its output itself cannot become a secret key. A secret key in cryptology should be not only unpredictable, but also the same value when it is used for a secret key of encryption, decryption, and authentication. For this reason, generally, a fuzzy extractor (FE) has been used with the PUF for making a secure fixed secret from the unpredictable PUF's outputs. In fact, much research on how the PUF can be securely combined with the existing

cryptographic primitives, such as PUF-based authentication, bit commitment, and PUF-based encryption, has been conducted in recent decades. From the short histories of the PAKE, MAKE, and PUF, one question naturally arises; can we design a MAKE protocol by using the existing and well-proved PAKE protocol? More precisely, is it possible to generally construct MAKE if we are given any secure PAKE and hardware dependent PUF? Generally, the MAKE needs a storage device for maintaining a long-term secret. That is why a huge amount of MAKE has been presented in a smartcard setting so far. Besides, the PUF is hardware dependent function and it is well suited for a storage device. Therefore, it is quite natural for a MAKE to assume a PUF-embedded device such as smartcard, DRAM, SRAM. This setting has one important benefit that the previous MAKE inherently has not satisfied. That is, even if three authentication factors (password, secret, biometrics) are all compromised, our MAKE with PUF-based device can be still secure.

1.3. Our Result. In the paper, we first study a topic on how to systematically construct a MAKE protocol with any secure PAKE protocol in a PUF-embedded device setting. In our setting, a user holds a PUF-embedded device and desires to mutually authenticate it with a server by multifactor authenticators. The design difficulties come from how common symmetric secrets are securely established from multiple authenticators. As illustrated in Figure 1, our main idea is to apply user's strong secrets (a long-term secret, biometrics) into an input of PUF in a device and obtain an output secret r. Our PUF in a MAKE actually does not need a password as an input;, instead, our MAKE just needs to execute a secure PAKE first and obtain PAKE's session key. Finally, PUF's r and PAKE's session key are hashed (random oracle) in order to produce two symmetric authentication keys. These keys perform a critical role for securely making message authentication codes for Diffie-Hellman values and a final session key of MAKE. Executing a secure PAKE first can avoid the security problem on the password such as off-line or on-line dictionary attacks. For instance, in case of using a long-term secret or biometric based AKE, we should first go through a problem on how a weak secret, password, can be securely combined with a secret or biometrics based AKE, which would be much more harder than the cases of combining the PAKE with a long-term secret and biometrics.

Most of all, our construction is the first PUF-based general MAKE constructions. By applying a PUF in a device, our MAKE construction can be still secure even if all authentication factors (password, secret, biometrics) are totally revealed. Moreover, our construction is relatively efficient in terms of communication cost. That is, our construction needs at most 6 communication flows (one execution of PAKE and three additional communication flows) that no public key based primitives such as signature are using. Unless the underlying PAKE is asymmetric-based setting, our MAKE construction is comprised of only symmetric primitives such as symmetric encryption, MAC function, which are relatively efficient than the works [35], as analyzed in Table 2. While the total communication round is greatly enhanced, our communication cost is not better than the FMA [35]. This slight inefficiency is due to the use of PUF and Diffie-Hellman keying materials. However, enhanced security and the reduced total communication flow balance up these little computation loss.

2. Preliminaries

2.1. Fuzzy Extractor and PUF

Definition 1 (fuzzy extractor[36]). A fuzzy extractor consists of two procedures, $\mathsf{Gen}(\cdot)$, $\mathsf{Rep}(\cdot)$:

(1) $\mathsf{Gen}(\cdot)$ outputs a string $R \in \{0,1\}^{l_r}$ and a helper string $P \in \{0,1\}^*$, on input $W \in M$, guaranteeing that for any distribution W with min-entropy m, if $(R,P) \longleftarrow \mathsf{Gen}(W)$, then $\mathsf{SD}((R,P),(U_l,P)) \leq \epsilon$, where the SD means statistical distance.

(2) $\mathsf{Rep}(\cdot)$ takes (W',P) on inputs and it outputs $\mathsf{Rep}(W',P) = R$ if $\mathsf{dist}(W,W') \leq t$ and $(R,P) \longleftarrow \mathsf{Gen}(W)$.

Definition 2 (PUF[37, 38]). A physically unclonable function $(t,\epsilon_1,\epsilon_2,\epsilon_{puf})$-PUF is a function $\{0,1\}^{l_p} \longrightarrow \{0,1\}^{l_o}$ satisfying the following.

(1) A PUF is bound to a device and its evaluation is efficient.

(2) For any PPT adversary $\mathscr{A}_{puf}$, PUF's output is impossible to characterize. Let $\mathsf{PUF}:\{0,1\}^{l_p} \longrightarrow \{0,1\}^{l_o}$ be an ideal PUF. An experiment $\mathsf{Exp}^{puf_b}_{\mathscr{A}_{puf}}(k)$ is defined for $b \xleftarrow{R} \{0,1\}$ where $k \in \mathbb{N}$ is a security parameter. $\mathsf{Exp}^{puf_0}_{\mathscr{A}_{puf}}(k)$ means an experiment in which $\mathscr{A}_{puf}$ can access an ideal PUF in polynomial number of times. That is, in this experiment, an oracle $\mathcal{O}^{\mathsf{PUF}}(\cdot)$ returns $y \longleftarrow \mathsf{P}(x)$ on input $x \in \{0,1\}^{l_p}$. Otherwise if $b = 0$, in $\mathsf{Exp}^{puf_1}_{\mathscr{A}_{puf}}(k)$, $\mathcal{O}^{\mathsf{PUF}}(\cdot)$ returns a random $y \xleftarrow{R} \{0,1\}^{l_o}$ as an answer. $\mathscr{A}_{puf}$'s goal is to finally output a guessing bit b' for b. To be precise, the following puf-advantage, ϵ_{puf}, should be negligible.

$$\epsilon_{puf} = \left|\Pr\left[\mathsf{Exp}^{puf_1}_{\mathscr{A}_{puf}}(k) = 1\right] - \Pr\left[\mathsf{Exp}^{puf_0}_{\mathscr{A}_{puf}}(k) = 1\right]\right| \quad (1)$$

(3) The distance between two PUF's outputs on the same challenge x is at most t; e.g., the following probability is negligible ϵ_1.

$$\Pr\left[\mathsf{dist}(y,z) > t \mid x \longleftarrow \{0,1\}^{l_p}, y \longleftarrow \mathsf{PUF}(x), z \longleftarrow \mathsf{PUF}(x)\right] \leq \epsilon_1 \quad (2)$$

(4) The distance between two outputs $\mathsf{PUF}_A(x)$, $\mathsf{PUF}_B(x)$ from different devices A, B, on the same

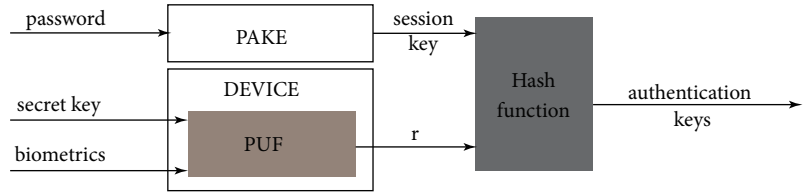


Figure 1: A generation of authentication keys from multifactor authenticators.

challenge x, is at least t; e.g., the following probability is negligible ϵ_2.

$$\Pr\left[\mathsf{dist}\left(y,z\right) < t \mid x \longleftarrow \{0,1\}^{l_p}, y \longleftarrow \mathsf{PUF}_A\left(x\right), z \longleftarrow \mathsf{PUF}_B\left(x\right)\right] \leq \epsilon_2 \tag{3}$$

Our PUF is based on an ideal PUF model where a PUF is considered as a random permutation oracle. In the same way as a random oracle, an adversary $\mathscr{A}_{puf}$ is adaptively able to query the PUF within polynomial time [37–41], but $\mathscr{A}_{puf}$ is never able to tell apart which one is real PUF or simulated PUF. This second property basically follows the definition in [37]. The third property says that the PUF's output ($r = \mathsf{PUF}(x)$) is always different within a certain distance d on the same input x. It is one of the good properties as any adversary cannot predict PUF's outputs, but its uncertainty makes it hard to directly apply $r(= \mathsf{PUF}(x))$ itself for making a cryptographic key. Thus, a PUF is usually combined with a fuzzy extractor which consists of Gen and Rep for making and recovering a fixed cryptographic secret k. That is, even if any adversary corrupts the input x, it cannot predict output $r(= \mathsf{PUF})$ or k without PUF while only valid user who holds PUF can recover k. The last property means that each PUF in different device should have own unique output on the same input. These third and fourth properties follow definitions in [38].

2.2. PRF and DDH

Definition 3 (pseudorandom function family (PRF)). A pseudorandom function family is a collection of functions $F_K : \mathsf{K}(F) \times \mathsf{D}(F) \longrightarrow \mathsf{R}(F)$ indexed by a key $K \in \mathsf{K}(F)$, where $\mathsf{K}(F)$ is the set of keys of F_K and $\mathsf{D}(F)$ is the set of domains of F_K, such that for any PPT algorithm $\mathscr{B}$, it holds the following:

(i) Given $x \in \mathsf{D}(F)$ and $K \in \mathsf{K}(F)$, there is a PPT algorithm to compute $\mathsf{F}_K(x)$.

(ii) For any PPT algorithm $\mathscr{B}$, the following *prf*-advantage, ϵ_{prf}, is negligible:

$$\left|\Pr\left[\mathscr{B}^{F_K} = 1 : K \xleftarrow{R} \mathsf{K}\left(F\right)\right] - \Pr\left[B^F = 1 : F \xleftarrow{R} U_{F_K}\right]\right| \leq \epsilon_{prf} \tag{4}$$

where U_{F_K} is the set of all functions mapping from $\mathsf{D}(F)$ to $\mathsf{R}(F)$ and $\xleftarrow{R}$ denotes an element chosen randomly.

Definition 4 (decisional Diffie-Hellman assumption (DDH)). The DDH problem is defined as follows: Let $\mathbb{G}$ be a cyclic group that has a prime order q and a generator g. The advantage of $\mathscr{A}$ is

$$\mathsf{Adv}_{\mathscr{A}}^{ddh}\left(T_D, \mathbb{G}\right) = \left|\Pr\left[\mathscr{A}\left(g^u, g^v, g^{uv}\right) = 1; u, v \longleftarrow \mathbb{Z}_q^*\right] - \Pr\left[\mathscr{A}\left(g^u, g^v, R\right) = 1; u, v \longleftarrow \mathbb{Z}_q^*, R \longleftarrow \mathbb{G}\right]\right| \tag{5}$$

where $\mathscr{A}$ is PPT adversary with time complexity T_D. The DDH assumption is that the advantage of $\mathscr{A}$ is negligible.

2.3. Secure MAC and Secure Symmetric Encryption under CCA

Definition 5 (secure message authentication code under chosen message attack). A message authentication code is a pair of algorithms MAC = $(\mathscr{T}, \mathscr{V})$. We define a MAC key set $M_k = \{0,1\}^{l_m}$ where l_m is the size of MAC key. An algorithm $\mathscr{T}$ is a MAC generation algorithm which takes as inputs a MAC key $K \in M_k$ and message x and outputs a string $\mathscr{T}(K, x)$ called a tag. An algorithm $\mathscr{V}$ is a MAC verification algorithm which takes key K, message x, and tag $\mathscr{T}(K, x)$ and outputs a bit 1 (authentic) or 0 (unauthentic). Let us consider the following experiment. A notation of $\mathscr{A}_{mac}^{O(\cdot)}$ indicates that the adversary $\mathscr{A}_{mac}$ can access an oracle $O(\cdot)$.

Experiment $\mathbf{Exp}_{\mathscr{A}_{mac}}^{cma}$

$k \xleftarrow{R} \mathsf{MKey}$

$(x^*, v^*) \longleftarrow \mathscr{A}_{mac}^{\mathscr{T}(K,\cdot),\mathscr{V}(K,\cdot,\cdot)}$: v^* is a tag value for a new message x^*

If $\mathscr{V}(K, x^*, v^*) = 1$ and v^* was never outputted by $\mathscr{T}(K, \cdot)$ in response to query x^*, then return 1; else return 0.

We define *cma* (chosen message attack) advantage of $\mathscr{A}_{mac}$ as follows.

$$\mathbf{Adv}_{\mathscr{A}_{mac}}^{cma}\left(T_m, k\right) = \Pr\left[\mathbf{Exp}_{\mathscr{A}_{mac}}^{cma}\left(T_m, k\right) = 1\right]. \tag{6}$$

If cma-advantage of **MAC** is negligible, the **MAC** scheme is secure under chosen message attack.

Definition 6 (secure symmetric encryption under chosen cipher attack). Let **SE** = (E, D) be a symmetric encryption scheme and $\mathscr{A}_{se}$ be an adversary of **SE**. We define a symmetric key set $\mathsf{Key} = \{0,1\}^{l_e}$ where l_e is the size of encryption key. An algorithm E is a probabilistic symmetric encryption algorithm which takes a symmetric key $K \in \mathsf{Key}$ and

plaintext m and outputs ciphertext $E_K(m)$. An algorithm D is a deterministic decryption algorithm which takes key K, plaintext m, and $E_K(m)$ as inputs and outputs message m. Let us consider the following experiment. A notation of $\mathcal{A}_{se}^{O(\cdot)}$ indicates that the adversary $\mathcal{A}_{se}$ can access an oracle $O(\cdot)$.

(1) $K \xleftarrow{R} \mathsf{Key}$

(2) **(Find Stage)** $(x_0, x_1, s) \longleftarrow \mathcal{A}_{se}^{E_K(\cdot)D_K(\cdot)}$

(3) $b \xleftarrow{R} \{0,1\}, c \longleftarrow E_K(x_b)$

(4) **(Guess Stage)** $b' \longleftarrow \mathcal{A}_{se}^{E_K(\cdot)D_K(\cdot)}(c, s)$

(5) If $b' = b$ then return 1 else return 0

For any T_{se}, q_e, q_d, we define *cca*-advantage of **SE** as

$$\mathsf{Adv}_{\mathcal{A}_{se}}^{cca}(k) = 2\Pr\left[b = b'\right] - 1. \tag{7}$$

If *cca*-advantage of **SE** is negligible, the **SE** scheme *is* secure under chosen ciphertext attack.

3. Security Model for Multifactor Authenticated Key Exchange

Based on the BPR model [2], in this section, a new security model is presented for PUF-based MAKE. We follow the basic framework and its notations of BPR model [2]. As our compiler converts any secure PAKE into secure PUF-based MAKE, two security models (PAKE and MAKE) are required for security proof. Two security models are the same except the number of authentication factors, session freshness.

3.1. Communication Model. Participants, long-lived secrets in the MAKE. Two types of protocol participants, a set of users $\mathbb{U}$ and a set of servers $\mathbb{S}$, are assumed in a MAKE protocol with multiple long-lived secrets. That is, we suppose that each user $\mathsf{U} \in \mathbb{U}$ owns a tuple $M_c = \{\tilde{\delta}, \mathsf{pw}, \mathbf{s}, \mathsf{PUF}\}$. U memorizes a secret password pw and possesses a physical unclonable function, PUF, which is usually embedded into a smartcard or memory card. U holds a long-term secret **s**. U itself owns biometric template $\tilde{\delta}$ such as iris or fingerprint. $\mathcal{S}$ securely keeps $M_s = \{ID, T[M_c]\}$ in its database, where ID is an identifier and $T[M_c]$ is a transformation of M_c. A user U (or a server S) may execute MAKE protocol multiple times and we also denote the i-th instance executed by entity U (resp., S) as U_m^i (resp. S_m^i).

Participants, long-lived secret in the PAKE. We assume two types of protocol participants, a user $\mathsf{U} \in \mathbb{U}$ and a server $\mathsf{S} \in \mathbb{S}$. The set of all participants is the union $\mathbb{U} \cup \mathbb{S}$. We suppose that each user $\mathsf{U} \in \mathbb{U}$ memorizes a secret password pw and the server S keeps a vector $\mathsf{pw}_s = \langle \mathsf{pw}_u \rangle_{\mathsf{u} \in \mathbb{U}}$ that contains entry for each client. The passwords $\mathsf{pw}, \mathsf{pw}_s$ are called the long-lived key of user U and server S. A user U (or a server S) may execute the 2-party PAKE protocol multiple times. The i-th instance is executed by entity U (resp., S) as U_t^i (resp., S_t^i).

Queries of Adversary in the MAKE. Queries are defined to model malicious behaviors of an adversary. We assume that there is an adversary $\mathcal{A}$ that is able to control communications over the networks and interact with the oracles by asking queries defined below.

(i) $\mathsf{Send}_m(U, i, M)$. It sends the message M to an oracle U_m^i. This query models that $\mathcal{A}$ sends a message M to an instance of U_m^i in the real protocol. The oracle U_m^i will return the next message that an honest U_m^i should be sent according to the protocol.

(ii) $\mathsf{Reveal}_m(U, t)$. If the oracle U_m^t has a session key, then this query outputs a session key to $\mathcal{A}$. This query models the possibility that $\mathcal{A}$ gets session keys.

(iii) $\mathsf{Corrupt}_m(U, \mathsf{a})$. This query models the possibility that $\mathcal{A}$ corrupts a client U. We assume that $\mathcal{A}$ can just obtain long-lived keys, but cannot obtain ephemeral states of instances. This query is classified into three types according to a.

(a) If $\mathsf{a} = 1$, then the password pw of U is returned as an answer.

(b) If $\mathsf{a} = 2$, then the secret PUF of U is returned as an answer.

(c) If $\mathsf{a} = 3$, then the secret $\tilde{\delta}$ of U is returned as an answer.

(d) If $\mathsf{a} = 4$, then the secret **s** of U is returned as an answer.

In case of PAKE, the cases of $a = 1$ are only considered.

(iv) $\mathsf{Execute}_m(U, i, S, j)$. This query returns transcripts of an honest execution between user instance U_m^i and server instance S_m^j. This query models a passive adversary who simply eavesdrops transcripts on an honest execution.

(v) $\mathsf{Test}_m(U_m^i)$. This query is only used to measure $\mathcal{A}$'s knowledge on a session key. When $\mathcal{A}$ asks a Test query to the *fresh* U_m^i, a coin is flipped to generate a random bit b. If $b = 1$, the real session key is returned to $\mathcal{A}$. Otherwise a random value is returned. $\mathcal{A}$ is allowed to make a Test query only once at any time during the experiment.

Queries of Adversary in the PAKE. All queries and their responses are the same as those of MAKE except a corrupt query with notions. A corrupt query in the PAKE is defined only in case of a=1. A notion on the instance, U_m^i, is replaced with U_t^i (S_t^j). And also, to distinguish notations in the security proof, we define notations of queries for the PAKE without subscript m (e.g., Send, Corrupt, Execute, Reveal, Test).

3.2. Security Definition

Definition 7 (session identifier, session key, partner identifier, and partnering 2-party **MAKE**). We define a session identifier (SID) for U_m^i as $\mathsf{sid}_{U_m^i}$ which is the concatenation of all messages sent and received by U_m^i. A session key

is denoted as $\mathsf{sk}_{U,S}$. A partner ID is a corresponding participant with whom U_m^i is interacting, which is denoted as $\mathsf{pid}_{U_m^i}$. Instances U_m^i and S_m^j are partnering if the following conditions are satisfied: (1) Both instances have been accepted with the same session key, session ID, and partner ID. That is, if both oracles have been accepted with $(\mathsf{sk}_{U,S}, \mathsf{sid}_{U_m^i}, \mathsf{pid}_{U_m^j})$, $(\mathsf{sk}_{S,U}, \mathsf{sid}_{S_m^j}, \mathsf{pid}_{S_m^j})$, then $(\mathsf{sk}_{U,S} = \mathsf{sk}_{S,U}, \mathsf{sid}_{U_m^i} = \mathsf{sid}_{S_m^j} \neq \mathsf{NULL})$. (2) No oracle other than U_m^i and S_m^j is accepted with the pid of U_m^i and S_m^j. $(\mathsf{pid}_{U_m^i} = S_m^j, \mathsf{pid}_{S_m^j} = U_m^i)$.

Regarding the PAKE, we use essentially the same definitions, notations for SID, a session key, and PID with the ones of MAKE model. Thus we do not define them again here. Next, we consider freshness, forward secure freshness for MAKE and PAKE, respectively.

Definition 8 (freshness for 2-party **MAKE**). An instance U_m^i is *fresh* in terms of multifactor AKE if (1) neither U_m^i nor its partner has been asked for a Reveal query; (2) neither U_m^i nor its partner has been asked for a $\mathsf{Corrupt}(U_m^i, 1)$, $\mathsf{Corrupt}(U_m^i, 2)$, $\mathsf{Corrupt}(U_m^i, 3)$, $\mathsf{Corrupt}(U_m^i, 4)$ queries. The second condition means that U_m^i and its partner have been asked for less than four corrupt queries. That is, $\mathscr{A}$ can ask Corrupt queries satisfying the following equation.

$$(\mathsf{Corrupt}(1) \wedge \mathsf{Corrupt}(2) \wedge \mathsf{Corrupt}(3) \wedge \mathsf{Corrupt}(4)) = \mathsf{FALSE} \tag{8}$$

If $\mathscr{A}$ has asked query of $\mathsf{Corrupt}(U, \mathsf{a})$ where $1 \leq \mathsf{a} \leq 4$, then it is $\mathsf{Corrupt}(\mathsf{a}) = \mathsf{TRUE}$. Otherwise, it is $\mathsf{Corrupt}(\mathsf{a}) = \mathsf{FALSE}$.

Definition 9 (forward secure freshness for 2-party **MAKE**). Regarding forward secrecy, a new freshness, *fs*-fresh, is defined. An instance U_m^i is *fs*-fresh in terms of multiple authentication factors if (1) neither U_m^i nor its partner has been asked for a Reveal query; (2) before $\mathsf{Test}(U_m^i)$, neither U_m^i nor its partner has been asked for all three $\mathsf{Corrupt}(U_m^i, 1)$, $\mathsf{Corrupt}(U_m^i, 2)$, $\mathsf{Corrupt}(U_m^i, 3)$, $\mathsf{Corrupt}(U_m^i, 4)$ queries and $\mathsf{Send}_m(U, i, M)$ queries.

Remark 10. The above first property (1) is for Reveal case for both PAKE and MAKE. However, the second property (2) has been modified by multiple authentication factors. For example, let us consider some cases of *fs*-fresh. If $\mathscr{A}$ asks $\mathsf{Corrupt}(U_m^i, 1)$, $\mathsf{Corrupt}(U_m^i, 2)$ queries (less than all four Corrupt queries) before $\mathsf{Test}(U_m^i)$ and does ask $\mathsf{Send}_m(U, i, M)$ query, then it is *fs*-fresh. Also, if $\mathscr{A}$ asks $\mathsf{Corrupt}(U_m^i, 1)$, $\mathsf{Corrupt}(U_m^i, 2)$, and $\mathsf{Corrupt}(U_m^i, 3)$ queries before $\mathsf{Test}(U_m^i)$ and does not ask $\mathsf{Send}_m(U, i, M)$ query, then it is *fs*-fresh. If all Corrupt queries have been asked after Test query, then it is also *fs*-fresh.

Definition 11 (freshness for 2-party **PAKE**). It is defined in the same way as MAKE except the second condition. That is, the first condition is the same as the MAKE, but the second condition is that (2) neither U_m^i nor its partner has been asked for a $\mathsf{Corrupt}(U_m^i, 1)$ query.

Definition 12 (forward secure freshness for 2-party **PAKE**). It is defined in the same way as the above MAKE except the second condition. That is, (2) before $\mathsf{Test}(U_m^i)$, neither U_m^i nor its partner has been asked for a $\mathsf{Corrupt}(U_m^i, 1)$ query and $\mathsf{Send}_m(U, i, M)$ query.

Definition 13 (session key security for 2-party **MAKE**). Let us consider the following experiment. We assume that an adversary $\mathscr{A}$ is run within a polynomial time bound T. $\mathscr{A}$ interacts with a finite number of oracles by asking queries defined above during the experiment. $\mathscr{A}$ can ask a Test query to a fresh oracle at any time. When $\mathscr{A}$ asks a Test query to the *fresh* U_m^i, then the experiment flips a coin for bit b. If it lands $b = 1$, then the real session key is returned to the adversary. Otherwise, a random key is returned to the adversary. Eventually $\mathscr{A}$ guesses b, outputs the guessed bit b', and terminates the experiment. The *sk* advantage of $\mathscr{A}$ in attacking the MAKE protocol P is defined as $\mathsf{Adv}_{P,sk}^{make}(\mathscr{A}) = 2\Pr[b = b'] - 1$. It should be negligible as follows.

$$\mathsf{Adv}_{P,sk}^{make}(\mathscr{A}, T, k) \leq C' \cdot q_s^{s'} + \varepsilon(k), \tag{9}$$

where $\varepsilon(k)$ is a negligible function, $|\mathbb{D}|(e.g., 12, 898, 437)$ is a password space for a Zipf's parameters C'(e.g., 0062239), s'(e.g., 0.155478), which follows Zipf's law for password [42, 43], q_s is the number of on-line Send queries asked, and k is a security parameter.

Remark 14. Since a memorable password is chosen by a user, it is not uniformly distributed. That is, due to the memorable property, it is natural for the user not to be able to make full use of the whole distribution of $\|\mathscr{D}\|$ at password selection stage. Thus, regarding the formalization term on an on-line-password guessing attack in all of our definitions here, we follow the Zipf assumption for password distributions [42, 43], which formalizes user own memorable password (not uniform from $\mathscr{D}$), $C' \cdot q_s^{s'}$, rather than a uniformly modeled term $q_s/\|\mathscr{D}\|$, where C', s' are Zipf's parameters [42, 43] depending on dataset, policy, language.

Definition 15 (session key security for 2-party **PAKE**). The definition for PAKE is the same as the MAKE security except that a notion $\mathsf{Adv}_{P,sk}^{make}(\mathscr{A}, T, R)$ is replaced with a notion $\mathsf{Adv}_{P,sk}^{pake}(\mathscr{A}, T, k)$. It is defined as $\mathsf{Adv}_{P,sk}^{pake}(\mathscr{A}, T, k) = 2\Pr[b = b'] - 1$ and it should be as follows. $\mathsf{Adv}_{P,sk}^{pake}(\mathscr{A}, T, k) \leq C' \cdot q_s^{s'} + \varepsilon(k)$ where $\varepsilon(k)$ is a negligible function, $|\mathbb{D}|$(e.g., 12, 898, 437) is a password space for a Zipf's parameters C'(e.g., 0062239), s'(e.g., 0.155478), which follows Zipf's law for password [42, 43], q_s is the number of on-line Send queries asked, and k is a security parameter.

Definition 16 (forward secure session key security for 2-party **PAKE**). A forward secure 2-party PAKE protocol is defined with a condition that the *fs*-fresh instance should

be asked for a Test query. Basically its definition is the same as the definition of session key secure PAKE except that a notion $\mathsf{Adv}^{pake}_{P,sk}(\mathscr{A},T,k)$ is replaced with a notion $\mathsf{Adv}^{fs-pake}_{P,sk}(\mathscr{A},T,k) = 2\Pr[b=b'] - 1$. It should be as follows: $\mathsf{Adv}^{fs-pake}_{P,sk}(\mathscr{A},T,k) \le C' \cdot q_s^{s'} + \varepsilon(k)$ where $\varepsilon(k)$ is a negligible function, $|\mathbb{D}|$(e.g., 12, 898, 437) is a password space for a Zipf's parameters C'(e.g., 0062239), s'(e.g., 0.155478), which follows Zipf's law for password [42, 43], q_s is the number of on-line Send queries asked, and k is a security parameter.

The mutual authentication for MAKE and PAKE is defined by an event, Succ, as follows.

Definition 17 (mutual Authentication [2]). We follow a definition of mutual authentication defined in [2]. For client authentication, we consider sessions where the server accepts with no partner instance of the client. With regard to server authentication, we consider sessions where the client accepts with no partner instance of the server. We say that an adversary breaks mutual authentication if either server or client instance is accepted, but has no partner oracle. We denote this event by Succ and its probability is defined as $\mathsf{Succ}^{ma}_P(\mathscr{A},T,k) = \Pr[\mathsf{Succ}]$. We say that a given MAKE protocol P provides mutual authentication if $\mathsf{Succ}^{ma}_P(\mathscr{A},T,k)$ is negligible as follows:

$$\mathsf{Succ}^{ma}_P(\mathscr{A},T,k) \le \varepsilon(k), \tag{10}$$

where $\varepsilon(k)$ is a negligible function and k is a security parameter.

4. A Compiler from PAKE to MAKE with PUF

The construction of MAKE consists of two phases; one is for the PAKE phase to derive a common key, and the other is for generic authentication phase with the derived common key, as illustrated in Figure 2 and Figure 3, respectively. We use the following notations in the protocol.

4.1. Notation

(i) c: It is a set of messages transferred between the user U and the server S in the PAKE. We assume that it has l_c bits. We can assume a function (pseudo-random or hash function) that maps a set of messages generated by any PAKE into the fixed size l_c bits.

(ii) ID, $\widetilde{\delta}$, pw, sk_p: ID is an identifier of U and it has l_{id} bits. $\widetilde{\delta}$ is U's biometric template, pw is U's memorable password, and $\mathbf{s}$ is a long-term secret key. sk_p is an agreed-upon common key between U and S in the PAKE, which are assumed to have l_p bits.

(iii) H_0, H_1, H_2, H_3: They are all cryptographic secure hash functions, which are simulated as random oracles in the security proof. They are, respectively, defined as $H_0 : \{0,1\}^{n_0} \longrightarrow \{0,1\}^{l_0}$, $H_1 : \{0,1\}^{n_1} \longrightarrow \{0,1\}^{l_1}$, $H_2 : \{0,1\}^{n_2} \longrightarrow \{0,1\}^{l_2}$, $H_3 : \{0,1\}^{n_3} \longrightarrow \{0,1\}^{l_3}$.

4.2. A Setup Description

(i) A setup is executed over a secure channel. U first computes a strong secret $\delta(=\mathsf{Gen}(\widetilde{\delta}))$ from its own biometric template $\widetilde{\delta}$. And, U evaluates PUF and generates (r,h) from Gen (PUF($\delta \parallel \mathbf{s}$)). The values $ID, r,$ pw are sent to S.

(ii) U and S securely maintain (pw, PUF, $\delta, \mathbf{s}$)($ID, r,$ pw), respectively.

We note that the setup phase is a necessary step to share pw in the PAKE setting. Our MAKE setup phase is also unavoidable for securely delivering multiple authentication factors between U and S.

4.3. A Description of the Compiler

(1) A user U and a server S execute any efficient and provably secure PAKE protocol with c (a set of all transferred messages) and a common key sk_p. If the agreement of sk_p or authentication is failed in the PAKE, then our compiler outputs a failure and stops the protocol. Otherwise, it continues to run the next step (2).

(2) The user U inputs own biometric template $\widetilde{\delta}$ and then computes a strong secret δ from $\mathsf{Rep}(\widetilde{\delta}, h_\delta)$. U evaluates PUF and executes Gen with h, then computes $r(=\mathsf{PUF}(H_0(\delta \parallel \mathbf{s}), h_r))$, and derives common keys $k_1(=H_1(c \parallel r \parallel sk_p)), k_2(=H_2(c \parallel r \parallel sk_p))$ for message authentication. U finally computes $\alpha_1(=g^x mod\, \mathrm{q})$ and $\beta_1(=MAC_{k_1}(\alpha_1))$ and sends them to S.

(3) On receiving α_1, β_1, S first searches U's list, (ID, r, pw) from database. Then S derives $k_1(=H_1(c \parallel r \parallel sk_p)), k_2(=H_2(c \parallel r \parallel sk_p))$ from c, r, sk_p. S with k_1 verifies that $\beta_1(=MAC_{k_1}(\alpha_1))$ is valid or not. If it is valid then S makes $\alpha_2 = g^y mod q$ for a random $y \in \mathbb{Z}_q^*$, its authentication tag $\beta_2 = MAC_{k_2}(\alpha_2)$, and $\gamma = (\alpha_1)^y$. S also computes keying material $sk'(=H_3(\gamma \parallel \alpha_1 \parallel \alpha_2 \parallel \beta_1 \parallel \beta_2 \parallel c))$ and its authenticator $h_1(=H_3(sk' \parallel 1))$. The messages α_2, β_2, h_1 are sent to U.

(4) Regarding α_2, β_2, h_1, U verify α_2's authenticator β_2. If it is valid then U computes keying material $sk'(=H_3(\gamma \parallel \alpha_1 \parallel \alpha_2 \parallel \beta_1 \parallel \beta_2 \parallel c))$. U also verifies $h_1(=H_3(sk' \parallel 1))$ by using sk'. If it is also valid then U makes $h_2(=H_3(sk' \parallel 2))$ and sends it to S. Then U finally computes a common session key $sk(=H_3(sk' \parallel 0))$. On receiving h_2, S verifies h_2 with sk'. If it is valid, S finally computes a common session key $sk = H_3(sk' \parallel 0)$.

In Figures 2 and 3, we illustrate a setup phase and a final general PUF-based MAKE construction, respectively.

User: U	Server: S
$(\mathsf{pw}, \mathsf{PUF}, \tilde{\delta}, \mathbf{s}, h_r, h_\delta)$	(ID, r, pw)
$(\delta, h_\delta) \leftarrow \mathsf{Gen}(\tilde{\delta})$ $(r, h_r) \leftarrow \mathsf{Gen}(\mathsf{PUF}(H_0(\delta\|\|\mathbf{s})))$	
$\xrightarrow{ID,r,pw}$	
$(\mathsf{pw}, \mathsf{PUF}, \tilde{\delta}, \mathbf{s}, h_r, h_\delta)$	(ID, r, pw)

Figure 2: A setup phase.

User: U		Server: S
$(\mathsf{pw}, \mathsf{PUF}, \tilde{\delta}, \mathbf{s}, h_r, h_\delta)$		(ID, r, pw)
	(1) $\Leftarrow$PAKE$\Rightarrow$	
Establish sk_p		Establish sk_p
$\delta \leftarrow \mathsf{Rep}(\tilde{\delta}, h_\delta)$ $r = \mathsf{Rep}(\mathsf{PUF}(H_0(\delta\|\|\mathbf{s})), h_r)$ $k_1 = H_1(c\|\|r\|\|sk_p)$ $k_2 = H_2(c\|\|r\|\|sk_p)$		
$\alpha_1 \leftarrow g^x \ mod \ q$ $\beta_1 = MAC_{k_1}(g^x \ mod \ q)$		
	(2) $\xrightarrow{\alpha_1, \beta_1}$	
		$k_1 = H_1(c\|\|r\|\|sk_p)$ $k_2 = H_2(c\|\|r\|\|sk_p)$
		Verify β_1 If β_1 is invalid, output FAIL
		Otherwise if β_1 is valid, compute the followings
		$\alpha_2 = g^y \ mod \ q$ $\beta_2 = MAC_{k_2}(g^y \ mod \ q)$ $\gamma = (\alpha_1)^y$ $sk' = H_3(\gamma\|\|\alpha_1\|\|\alpha_2\|\|\beta_1\|\|\beta_2\|\|c)$ $h_1 = H_3(sk'\|\|1)$
	(3) $\xleftarrow{\alpha_2, \beta_2, h_1}$	
Verify β_2, h_1 If β_2 is invalid, output FAIL If h_1 is invalid, output FAIL		
Otherwise, compute $\gamma = (\alpha_2)^x$ $sk' = H_3(\gamma\|\|\alpha_1\|\|\alpha_2\|\|\beta_1\|\|\beta_2\|\|c)$ $h_2 = H_3(sk'\|\|2)$		
	(4) $\xrightarrow{h_2}$	
		Verify h_2 If h_2 is invalid, output FAIL
Compute $sk = H_3(sk'\|\|0)$		Otherwise, compute $sk = H_3(sk'\|\|0)$

Figure 3: A general construction of MAKE.

Our construction combines two authentication factors, biometric template, and long-term secret and makes one secret r using a PUF. However, our compiler does not directly handle a password at all, but just takes a PAKE as an input. That is, our MAKE obtains PAKE's session key sk_t through executing a secure PAKE. This approach can guarantee the password security such as off-line or on-line dictionary attacks, as explained earlier. Anyway, these two secret values r, sk_t are mainly used to establish intermediate keys k_1, k_2.

4.4. Security Proof. The following well-known lemma [44] is used for calculating difference of probability between adjacent games in the security proof.

Lemma 18. *Let E, E', and F be events defined on a probability space such that $Pr[E \wedge \neg F] = Pr[E' \wedge \neg F]$. Then, we have $|Pr[E] - Pr[E']| \leq Pr[F]$.*

Theorem 19. *Let $\mathcal{A}$ be a PPT adversary against the session key security of* MAKE *protocol P within a time bound T. $\mathcal{A}$ can ask q_{s_m} to send queries, q_{e_m} executes queries, and then*

$$\begin{aligned}
\mathsf{Adv}_{P,sk}^{make}(\mathcal{A}, T, k) &\leq 2\mathsf{Adv}_{P,sk}^{pake}\left(\mathcal{A}_p, T_a, k\right) + 2\epsilon_{puf} \\
&\quad + + \frac{q_{H_3}^2}{2^{l_3}} + \frac{\left(q_{s_m} + q_{s_e}\right)^2}{2^{2p+1}} \\
&\quad + 2\mathsf{Adv}_{\mathcal{A}_{ddh}}^{ddh}(T_D, \mathbb{G}) \\
&\quad + 4\mathsf{Adv}_{\mathcal{A}_{mac}}^{cma}(T_{mac}, k) \\
\mathsf{Succ}_P^{ma}(\mathcal{A}, T, k) &\leq \frac{q_{s_m}^2\left(2^{l_1+1} + 2^{l_2+1}\right)}{2^{l_1+l_2+2}} \\
&\quad + \mathsf{Adv}_{P,sk}^{pake}\left(\mathcal{A}_p, T_a, k\right) + \epsilon_{puf} \\
&\quad + \frac{q_{H_3}^2}{2^{l_3+1}} + \frac{\left(q_{s_m} + q_{s_e}\right)^2}{2^{2(p+1)}} \\
&\quad + \mathsf{Adv}_{\mathcal{A}_{ddh}}^{ddh}(T_D, \mathbb{G}) \\
&\quad + 2\mathsf{Adv}_{\mathcal{A}_{mac}}^{cma}(T_{mac}, k)
\end{aligned} \tag{11}$$

where $T_a, T_D \geq T + q_{s_m}(T_f + T_e + T_h), T_{mac} \geq T + T_f(q_t + q_v)$. q_{s_m}, q_t, q_v are the total numbers of Send_m *, tag, verification (for MAC) queries. T_f, T_e, T_h are the computation time for MAC, exponentiation, and hash function, respectively.*

Proof. We assume an adversary $\mathcal{A}$ that runs in games from $\mathbf{G}_0$ to $\mathbf{G}_5$. In each game, $\mathcal{A}$ asks a Test_m query, then a coin for bit b is flipped to specify a real key or a random key, and then $\mathcal{A}$ outputs a guessing bit b'. Let $\mathbf{S}_i$ be an event that $b' = b$ in $\mathbf{G}_i$ and $\Pr[\mathbf{S}_i]$ is its probability. Also, Succ_i is defined as an event that a server or a user instance accepts with no partner instance of the client in $\mathbf{G}_i$ as in Definition 17.

Game G_0. The game $\mathbf{G}_0$ provides a real environment for the MAKE protocol. By the security definition, the session key advantage of $\mathcal{A}$ is defined as follows.

$$\begin{aligned}
\mathsf{Adv}_{P,sk}^{make}(\mathcal{A}, T, k) &= 2\Pr[\mathbf{S}_0] - 1, \\
\mathsf{Succ}_P^{ma}(\mathcal{A}, T, k) &= \Pr[\mathsf{Succ}_0]
\end{aligned} \tag{12}$$

Game G_1. In this game, a random key sk_r is selected randomly and it is replaced with the established key sk_p of the PAKE. Through the replacement, we show that if there exists a PPT adversary A_D that distinguishes two games, then one can build an adversary A_p that breaks the security of PAKE, which is shown in Lemma 20.

Lemma 20. *The difference of success probability between $\mathbf{G}_0$ and $\mathbf{G}_1$ is at most the advantage of breaking 2-party* PAKE *protocol.*

$$\left|\Pr[\mathbf{S}_0] - \Pr[\mathbf{S}_1]\right| = \mathsf{Adv}_{P,sk}^{pake}\left(\mathcal{A}_p, T_a, k\right) \tag{13}$$

where $T_a \geq T + q_{s_m}(T_f + T_e + T_h)$. q_{s_m} is the total number of Send_m *queries. T_f, T_e, T_h are the computation time for MAC, exponentiation, and hash function.*

Proof. A_p exploits A_D to guess a hidden bit b defined in the experiment of 2-party PAKE. A_p provides simulations on queries asked by A_D. All queries are simulated as follows.

(i) Hash queries of H_1, H_2, H_3: If $\mathcal{A}_D$ asks a hash query of m to H_1 (resp. H_2, H_3) where a list (m, λ) appears in the hash table δ_{H_1} (resp. $\delta_{H_2}, \delta_{H_3}$), then λ is returned as an answer for $H_1(m)$ (resp. $H_1(m), H_2(m)$). Otherwise, a new $\lambda \in \{0,1\}_1^l$ (resp. l_2, l_3) is chosen at random and returned. Then the list (m, λ) is added to the table δ_{H_1} (resp. $\delta_{H_2}, \delta_{H_2}$).

(ii) $\mathsf{Send}_m(U, i, M)$: It is classified into two types as follows.

(a) If Send query is asked to the parts of PAKE, A_p first asks $\mathsf{Send}(U, i, m)$ query to the PAKE and then returns its output as an answer. At the end of PAKE protocol, there exists a Send query to let the instance be accepted in the PAKE. For this Send query, A_p uses the random key sk_r to simulate an answer α_1, β_1. That is, first, A_p evaluates PUF computes r, and then decides $k_1 = H_1(c \parallel r \parallel \boxed{sk_r})$, $k_2 = H_2(c \parallel r \parallel \boxed{sk_r})$ from sk_r and H_1, H_2 oracles. A_p finally produces an answer, $\alpha_1 = g^x \bmod q$, $\beta_1 = \mathrm{MAC}_{k_1}(\alpha_1)$ for a random x from $\mathbb{Z}_q^*$. A_p keeps r, α_1, β_1 for later simulations.

(b) Otherwise if $M = \alpha_2, \beta_2, h_1$, then $\mathcal{A}_p$ verifies h_1. If it is valid, then it is accepted with $sk' = H_3(\gamma \parallel \alpha_1 \parallel \alpha_2 \parallel \beta_1 \parallel \beta_2 \parallel c)$ and $h_2 = H_3(sk' \parallel 2)$. $\mathcal{A}_p$ finally outputs h_2 as an answer.

(iii) $\mathsf{Send}_m(S, j, M)$: There are also three types of queries as follows.

(a) If Send query belongs to the query in PAKE, A_p asks $\mathsf{Send}(S, i, M)$ query to the PAKE and then uses its output as an answer.

(b) If $M = \alpha_1 \| \beta_1$, then A_p verifies β_1. If it is valid, then A_p makes $k_1 = H_1(c \| r \| \boxed{sk_r})$, $k_2 = H_2(c \| r \| \boxed{sk_r})$ with previously made value c, r. A_p finally produces an answer, $\alpha_2 = g^y mod q$, $\beta_2 = \mathrm{MAC}_{k_1}(\alpha_2)$ for a random y from $\mathbb{Z}_q^*$. A_p keeps α_2, β_2 for later simulations.

(c) Otherwise if $M = h_2$, then A_p first verifies h_2. If it is valid, then A_p accepts it with $sk = H_3(sk' \| 0)$.

(iv) $\mathsf{Execute}_m(U, i, \mathsf{S}, j)$: To return transcripts of an honest execution between U_m^i and S_m^j, $\mathscr{A}_p$ collects all outputs of Send_m queries and outputs them as an answer

(v) $\mathsf{Reveal}_m(U, t)$: For the instance that has been accepted, A_p returns $sk = H_3(sk' \| 0)$ as an output where $sk' = H_3(\gamma \| \alpha_1 \| \alpha_2 \| \beta_1 \| \beta_2 \| c)$.

(vi) $\mathsf{Corrupt}_m(U, a)$: According to a, it outputs registered password pw $(a = 1)$, PUF $(a = 2)$, biometric template $\widetilde{\delta}$ $(a = 3)$, and $\mathbf{s}$ $(a = 4)$ for U.

(vii) $\mathsf{Test}_m(U, t)$: When A_D asks $\mathrm{Test}_m(U, i)$ query, A_p chooses a random bit $b \in \{0, 1\}$ and selects a *fs-fresh* instance in aspect of MAKE and asks a Test (U, i) query. For the hidden bit b (real or session key), A_D outputs b'. If $b = b'$, then A_p outputs 1; otherwise, A_{pake} outputs 0.

Let us consider the advantage of A_p over the above simulation. We define an event real as a real session key, sk_p $(b = 1)$ is given by experiment of MAKE security while an event rand is a random key, and sk_r $(b = 0)$ is given to A_p.

$$\begin{aligned}\mathsf{Adv}_{P,sk}^{pake}\left(T', \mathscr{A}_p, k\right) &= 2\Pr\left[b = b'\right] - 1 \\ &= \left|\Pr\left[A_p = 1 \mid \mathsf{real}\right] - \Pr\left[A_p = 1 \mid \mathsf{rand}\right]\right| \\ &= \left|\Pr\left[b = b' \mid \mathbf{G}_0\right] - \Pr\left[b = b' \mid \mathbf{G}_1\right]\right| \\ &= \left|\Pr\left[\mathbf{S}_0\right] - \Pr\left[\mathbf{S}_1\right]\right|\end{aligned} \tag{14}$$

If we suppose A_D runs with polynomial time T, then A_p's running time T_a is dependent on T. Additionally, it requires the time of making the messages $\alpha_1, \beta_1, \alpha_2, \beta_2, h_1, h_2$. Hence the total time is bounded as $T_a \geq T + q_{s_m}(T_f + T_e + T_h)$. q_{s_m} is the total number of Send_m queries. T_f, T_e, T_h are the computation time for MAC, exponentiation, hash function.

This completes the proof of Lemma 20.

Game G_2. In this game, we consider a value r in the protocol, which is an output of ideal PUF. We replace r with a random value as a random oracle. That is, a table $\mathscr{L}_P$ is maintained for a valid simulation. Regarding each input x, if a list (x, y) appears in $\mathscr{L}_P$, then y is outputted as an answer. Otherwise, a random y is selected from $\{0, 1\}^{l_o}$ and returned as an answer for $\mathsf{PUF}(x)$. The list (x, y) is added to the list $\mathscr{L}_P$. Then, if there exists $\mathscr{A}$ that is able to distinguish two adjacent games, then it is clear to build an adversary $\mathscr{A}_{puf}$. In $\mathbf{G}_1$, an ideal PUF has been used, but in $\mathbf{G}_2$, a randomly chosen value is used. $\mathscr{A}_{puf}$ returns an output bit $d = 1$ if $b' = b$. Otherwise, $\mathscr{A}_{puf}$ returns $d = 0$. Then we have the following.

$$\begin{aligned}\epsilon_{puf} &= \left|\Pr\left[\mathsf{Exp}_{\mathscr{A}_{puf}}^{puf_1}(k) = 1\right] - \Pr\left[\mathsf{Exp}_{\mathscr{A}_{puf}}^{puf_0}(k) = 1\right]\right| \\ &= \left|\Pr\left[d = 1 \mid \mathsf{PUF}(x)\right] - \Pr\left[d = 1 \mid y \longleftarrow \{0, 1\}^{l_o}\right]\right| \\ &\geq \left|\Pr\left[b = b' \mid \mathbf{G}_1\right] - \Pr\left[\ b = b' \mid \mathbf{G}_2\right]\right| \\ &= \left|\Pr\left[\mathbf{S}_1\right] - \Pr\left[\mathbf{S}_2\right]\right|\end{aligned} \tag{15}$$

Game G_3. In this game, we analyze a collision event Col that an identical session key is accidentally generated for any two sessions. This event is due to a random oracle H_3 that produces final secrets sk', sk. It can be analyzed in two cases. The first case is that if its simulated output (randomly chosen value) is identical for two sessions, then two sessions naturally have the same sk', sk. For example, on two different input queries $x = (\gamma \| \alpha_1 \| \alpha_2 \mid \beta_1 \| \beta_2 \| c), x' = (\gamma' \| \alpha_1' \| \alpha_2' \mid \beta_1' \| \beta_2' \| c')$, each output can be simulated as an one random value $(e.g.(x, y), (x', y) \in \mathscr{L}_p)$. However, its probability is $q_{H_3}^2/(2^{l_3} + 1)$, which is negligible. The second case is that the input queries $(\gamma \| \alpha_1 \| \alpha_2 \mid \beta_1 \| \beta_2 \| c)$ themselves can be the same and then they naturally output the same output of sk', sk. The security (duplication) of c comes from the security of PAKE, which is negligible. In addition, the values α_2, β_2 are decided by α_1, α_2. Hence, for two sessions, the probability of equally selecting $\alpha_1 = g^x mod q, \beta_1 = g^y mod q$ is bound to $(q_{s_m} + q_{s_e})^2/2^{2(q+1)}$ where q_{s_m}, q_{s_e} are the total numbers of Send, $\mathsf{Execute}$ queries. We have the following.

$$\left|\Pr\left[\mathbf{S}_2\right] - \Pr\left[\mathbf{S}_3\right]\right| \leq \Pr\left[\mathsf{Col}\right] \leq \frac{q_{H_3}^2}{2^{l_3+1}} + \frac{\left(q_{s_m} + q_{s_e}\right)^2}{2^{2(q+1)}} \tag{16}$$

Game G_4. In this game, we consider an event MS that $\mathscr{A}$ makes valid $\mathsf{Send}(S, j, \widetilde{M})$ queries where $\widetilde{M} = \widetilde{\beta}_1 \| \mathrm{MAC}_{k_1}(\widetilde{\alpha}_1)$. Regarding valid messages α_1, β_1, their $\mathsf{Send}(S, j, M)$ query has been simulated in $\mathbf{G}_2$. Here, our goal is to exclude forgery events among $\mathsf{Send}(S, j, M)$ queries that have valid MAC pair $\widetilde{\beta}_1 \| \mathrm{MAC}_{k_1}(\widetilde{\alpha}_1)$. Let us assume a MAC forgery $\mathscr{A}_{mac}$. If this event happens, then $\mathscr{A}_{mac}$ halts and outputs the pair $\widetilde{\alpha}_1, \widetilde{\beta}_1$ as its forgery of MAC, which contradicts a secure MAC assumption. If there are no forgeries, then $\mathscr{A}_{mac}$ returns a failure notification. Other processes of simulation for $\mathscr{A}$ are the same as those of previous games. $\mathscr{A}_{mac}$ requires computational time T_{mac} such that $T_{mac} \geq T + T_f(q_t + q_v)$, where $\tau_{\mathscr{T}}$ is computational time for MAC and q_t, q_v are tag and verification queries that $\mathscr{A}$ makes. The same analysis applies to the case of α_2, β_2. We have the following.

$$\begin{aligned}\Pr\left[\mathbf{G}_3 \mid \neg\mathsf{MS}\right] &= \Pr\left[\mathbf{G}_4 \mid \neg\mathsf{MS}\right] \\ \left|\Pr\left[\mathbf{G}_3\right] - \Pr\left[\mathbf{G}_4\right]\right| &\leq \Pr\left[\mathsf{MS}\right] \leq 2 \cdot \mathsf{Adv}_{\mathscr{A}_{mac}}^{cma}\left(T_{mac}, k\right)\end{aligned} \tag{17}$$

Game G_5. In this game, we consider a random **DDH** triple $D_{rand} = (X, Y, Z)$ where $X = g^u \, mod \, q, Y = g^v \, mod \, q$, and

$Z = g^r \bmod q$. Using D_R, the Send queries are simulated as follows.

(i) $\mathsf{Send}_m(U, i, M)$: It is classified into two types as follows.

(a) If Send query is asked for letting the instance be accepted in the PAKE, then $\mathbf{G}_5$ uses the random key sk_r to make k_1, k_2. However, the value α_1 is replaced with $\boxed{X}$. Accordingly, the value $\beta_1 (= MAC_{k_1}(X))$ is made by k_1, X. Other simulations are the same as the previous games.

(b) Otherwise if $M = \alpha_2, \beta_2, h_1$, then $\mathscr{A}_p$ verifies h_1. If it is valid, then accepts with $sk' = H_3(\boxed{Z} \parallel \alpha_1 \parallel \alpha_2 \parallel \beta_1 \parallel \beta_2 \parallel c)$ and $h_2 = H_3(sk' \parallel 2)$. $\mathscr{A}_p$ finally outputs h_2 as an answer.

(ii) $\mathsf{Send}_m(S, j, M)$: There are also three types of queries as follows.

(a) If Send query belongs to the query in PAKE, the way of simulations is the same as the previous game.

(b) If $M = \alpha_1 \parallel \beta_1$, then A_p verifies β_1. If it is valid, then A_p makes $k_1 = H_1(c \parallel r \parallel sk_r)$, $k_2 = H_2(c \parallel r \parallel sk_r)$. However, the answer value α_2 is replaced with $\boxed{Y}$ and its MAC value is also computed by $\beta_2 = \mathrm{MAC}_{k_1}(\boxed{Y})$. Others are the same as the previous games.

By using $\mathscr{A}$, which tries to distinguish two games $\mathbf{G}_4$ and $\mathbf{G}_5$, one can build a polynomial time algorithm $\mathscr{A}_{ddh}$ that breaks the assumption of **DDH**. For a **Test** query, $\mathscr{A}$ guesses a coin and outputs b'. If $b = b'$, then $\mathscr{A}_{ddh}$ outputs 1, which means that the input triple is a real **DDH** triple. Otherwise it outputs 0 which means that the input triple is a random **DDH** triple. The success probability of $\mathscr{A}_{ddh}$ is calculated as follows. $\mathscr{A}_{ddh}$ requires computation time for simulating Send queries such that $T_D \geq T + q_{s_m}(T_f + T_e + T_h)$.

$$\begin{aligned}
&\mathsf{Adv}^{ddh}_{\mathscr{A}_{ddh}}(T_D, \mathbb{G}) \\
&= \left| \Pr\left[\mathscr{A}_{ddh}(g^x, g^y, g^{xy}) = 1; x, y \longleftarrow \mathbb{Z}_q^*\right] \right. \\
&\left. - \Pr\left[\mathscr{A}_{ddh}(X, Y, Z) = 1; u, v \longleftarrow \mathbb{Z}_q^*, Z \longleftarrow \mathbb{G}\right] \right| \\
&\leq \Pr\left[\mathscr{A}_{ddh} = 1 \mid D_{real}\right] - \Pr\left[\mathscr{A}_{ddh} = 1 \mid D_{rand}\right]. \\
&= \Pr\left[b = b' \mid \mathbf{G}_4\right] - \Pr\left[b = b' \mid \mathbf{G}_5\right]. = \Pr\left[\mathsf{S}_4\right] \\
&- \Pr\left[\mathsf{S}_5\right].
\end{aligned} \tag{18}$$

Let us consider $\Pr[\mathsf{S}_5]$ in $\mathbf{G}_5$. As simulated in Send queries, the session key sk is decided by the given random element Z. Hence sk is random and independent of elements U, V. That is, no information of b is leaked through the **Test** query. We have the following.

$$\Pr\left[\mathsf{S}_5\right] = \frac{1}{2} \tag{19}$$

Thus, we obtain the final result as follows.

$$\begin{aligned}
\Pr\left[\mathsf{S}_0\right] &\leq \mathsf{Adv}^{pake}_{P,sk}\left(\mathscr{A}_p, T_a, k\right) + \epsilon_{puf} + \frac{q_{H_3}^2}{2^{l_3+1}} \\
&+ \frac{\left(q_{s_m} + q_{s_e}\right)^2}{2^{2(p+1)}} + \mathsf{Adv}^{ddh}_{\mathscr{A}_{ddh}}(T_D, \mathbb{G}) + 2 \\
&\cdot \mathsf{Adv}^{cma}_{\mathscr{A}_{mac}}(T_{mac}, k) + \frac{1}{2}
\end{aligned} \tag{20}$$

Let us consider the event Succ_i . We have already excluded collision events on α_1, β_2 and MAC forgeries on β_1, β_2 in the previous games. The left analysis is on h_1, h_2. In this game, the secret sk' is made by a random oracle H_3 and $h_1 (= H_3(sk' \parallel 1))$ is also simulated by a random oracle H_3. Thus, for successful impersonation, $\mathscr{A}$ should correctly guess h_1, h_2. This probability is bounded as $q_{s_m}^2/2^{l_1+1}, q_{s_m}^2/2^{l_2+1}$, respectively.

$$\Pr\left[\mathsf{Succ}_5\right] \leq \frac{q_{s_m}^2}{2^{l_1+1}} + \frac{q_{s_m}^2}{2^{l_2+1}} \tag{21}$$

By the following difference, we can bound $\mathsf{Succ}^{ma}_P(\mathscr{A}, T, k)$. ($\Delta_i$ is the probability difference between $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$.)

$$\begin{aligned}
\mathsf{Succ}^{ma}_P(\mathscr{A}_M) &= \Pr\left[\mathsf{Succ}_0\right] \leq \Pr\left[\mathsf{Succ}_5\right] + \sum_{i=0}^{4} \Delta_i \\
&\leq \frac{q_{s_m}^2\left(2^{l_1+1} + 2^{l_2+1}\right)}{2^{l_1+l_2+2}} \\
&+ \mathsf{Adv}^{pake}_{P,sk}\left(\mathscr{A}_p, T_a, k\right) + \epsilon_{puf} \\
&+ \frac{q_{H_3}^2}{2^{l_3+1}} + \frac{\left(q_{s_m} + q_{s_e}\right)^2}{2^{2(p+1)}} \\
&+ \mathsf{Adv}^{ddh}_{\mathscr{A}_{ddh}}(T_D, \mathbb{G}) + 2 \\
&\cdot \mathsf{Adv}^{cma}_{\mathscr{A}_{mac}}(T_{mac}, k)
\end{aligned} \tag{22}$$

This completes the proof of Theorem 19 . □

5. Performance Analysis and Discussion

For coherent comparison and analysis, we follow a recent evaluation metrics that has been suggested by Wang and Wang [45] and been utilized in many other recent studies [20, 23, 45–47]. One important difference is that our scheme is not two-factor (smartcard and password) AKE scheme, but a generic construction that securely converts any secure PAKE into MAKE. Hence, regarding criteria on a password, our construction depends on the properties that the basis PAKE itself guarantees. And also, our scheme aims for a universal construction and thus it cannot guarantee or handle criteria on a smart card. Moreover, our scheme is the first generic MAKE using a PUF; for short, there is no scheme for comparison. However, in view of generic construction, we compare our generic construction with other recent schemes

Table 1: Security criteria comparison.

Scheme	G	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
HXCZD [33]	×	×	√	-	-	√	-	√	√	-	√	-	√
YWMG [34]	×	×	√	-	-	√	-	√	√	-	√	-	√
FMA [35]	√	×	-	-	-	√	-	√	√	-	√	-	√
Our scheme	√	×	-	-	√	√	×	√	√	-	√	-	√

G: whether it is a generic compiler or not, √: satisfied, ×: unsatisfied, -: depending on the basis instantiations or not considered.

Table 2: Efficiency comparison.

Scheme	G	TFR[1]	TCC[2]	TSS[3]
HXCZD [33]	×	$4 \le 2 \cdot ta \le 6$	$h + f$	$h + f$
YWMG [34]	×	$2 \le ta \le 3$	$2e + v + pe + f + h$	$2e + s + pd$
FMA [35]	√	$10 \le u + pw + b + pk + 2 \le 14$	$3h$	$3h$
Our scheme	√	$5 \le pw + 3 \le 6$	$5h + 2e + m + p$	$5h + 2e + m$

[1]TFR: the total flow number of communications. The notions u, pw, b, pk, ta, respectively, mean unauthenticated key exchange, PAKE, biometric-based authentication, public key authentication, and two-factor authentication protocols. [2]The total computation time for a user: h, e, m: computation time for hash, exponentiation, and MAC operations, resp.; p, f: computation time for PUF, fuzzy vault, resp.; s, v: computation time for a signature generation and verification, resp. [3]The total computation time for a server.

in Table 1 (security) and Table 2 (efficiency). As explained earlier, we remind again that recent schemes [33, 34] cannot be a generic construction, unlike their claims, even though their subjects are on general construction. However, please note that their constructions have been built up with the assumption of secure existing PAKE or TAKE. Thus, those two schemes [33, 34] have been analyzed with the FMA [35] scheme in each table. First, we briefly summarize 12 evaluation criteria introduced in [45].

(i) C1: The server never needs a database for user password or verifier.

(ii) C2: The user can freely choose and change a memorable password

(iii) C3: The server administrator should not be able to derive user's password.

(iv) C4: The scheme is strong against a smart card loss attack.

(v) C5: The scheme is strong against off-line, replay, parallel session, desynchronization, stolen verifier, impersonation, key control, unknown key share, and known key attacks.

(vi) C6: The user can revoke own card without changing its own identity.

(vii) C7: The scheme provides a key agreement between participants.

(viii) C8: The scheme is free from a clock synchronization.

(ix) C9: If a wrong password is typed by mistake, any type of notification is alarmed to the user.

(x) C10: The scheme provides a mutual authentication.

(xi) C11: The scheme protects users' identities.

(xii) C12: The scheme provides forward secrecy.

5.1. Discussion on Security Criteria. Our MAKE has been built up with a secure PAKE scheme. From the PAKE, our compiler securely adds other multiple factors to achieve an authenticated key exchange. Since the secure PAKE itself handles security of password, our construction is free from its related attacks. Thus, in some criteria, our scheme depends on how the basis PAKE has been instantiated. The same applies to other schemes [33–35]. It is because not only they have been constructed from secure initiations (general construction), but also security criteria defined in [45] (Table 1) are actually motivated from two-factor authenticators (password and smart card). For instance, a general MAKE construction just assumes multiple long-term secrets and does not take account of how these secrets are maintained. Thus, a smart card stolen attack (C4) cannot be considered in a general construction model. In our model, however, we firstly use PUF as an authenticator that is dependent on a hardware device such as a smart card, and C4 can be analyzed for comparison. Even if an adversary obtains PUF, our scheme is still secure authentication unless other secrets remain secure, which is better criterion than others. The criteria C2, C3, C4, C6, C9 are closely related to the underlying PAKE or TAKE. In particular, C1 evaluates whether a database is required or not for a password verifier. In most TAKE schemes over a smart card, actually C1 may be easily attained. It is because they assume a smart card that is capable enough to take a valid password as an input through a card reader and calculate a valid shared secret with a server if its input password is valid on a smart card. Afterwards, the server just takes a step for checking whether the shared secret is valid or not, which never requires a password or password verifier on a server side. However, in generic compiler setting, first of all, holding a smart card is not necessary condition. It just focuses on how to securely transform a secure primitive one into an integrated primitive one. One may argue that a general MAKE can be systematically constructed from a TAKE with a smart card. But, in this approach, the underlying TAKE already includes

a smart card, and it is hard (or not practical) to include a new hardware dependent PUF more. In conclusion, in PUF setting, it is more reasonable to initiate PAKE (or AKE) first and then construct MAKE by using PUF. Table 1 shows that our scheme is an accurate general MAKE construction with relatively better security performances.

5.2. Discussion on Efficiency. Table 2 shows, in general construction setting, that our scheme achieves better communication cost than the FMA [35] scheme. The FMA scheme executes many types of subprotocols for making a general MAKE. It may guarantee better security from the secure underlying subprotocols with simple implementation. However, two authentication factors are never used together for one subprotocol, and many types of subprotocols should be initiated independently. Hence, it has a fundamental limits in order to reduce the total communication cost. In Table 2, we assume that a normal PAKE or TAKE can be designed in two communication flows without an authentication. This influences upper or lower bounding of TFR. In conclusion, our scheme requires at most 6 communication flows with no public key based setting such as a signature used. While the total communication flow is more greatly enhanced than the FMA [35], our communication cost is not better than the FMA. This slight inefficiency is due to the use of PUF and Diffie-Hellman keying materials for guaranteeing forward secrecy. That is, enhanced security and the reduced total communication cost balance up this little computation loss. One interesting topic would be on how we can design MAKE with PUF at the lower communication bounding cost ($e.g., 4 \leq pw + 2$). Except the part of PAKE, our compiler additionally needs three more communication flows. But, by initiating PAKE, we already share a common key. This common secret can be used for making a final session key with slight modification. For instance, in our protocol, a user can compute a final session key after receiving messages in step (3), ignoring step (4). In the same way, upon messages of (2), a server can generate a final session key before step (3). The messages in steps (2) and (3) can be designed for just sharing ephemeral values and verifying (authentication) keying materials by using a shared key from PAKE. This construction remains one of future work including security analysis under rigorous security model.

6. Concluding Remarks

In the paper, we presented a general PUF-MAKE construction that securely transforms a PAKE into a MAKE with a physically unclonable function. Our construction especially takes a secure PAKE rather than taking the PKI-based (or non PKI-based) AKE as an input. The reason is that it naturally resolves a password issue such as off-line or on-line dictionary attacks. For instance, in case of transforming a PKI-based AKE into a MAKE, we should first go through a problem on how an easy-to-guess password can be securely combined with the PKI-based AKE and biometrics, which would be much more harder than the case of combining the PAKE with a long-term key and biometrics. Nevertheless, in view of foundation works, it is quite meaningful to show the existence of PUF-MAKE from PKI, non-PKI-based AKE, or other AKE primitives. In addition, we elaborated on two-party setting here, but a general PUF-MAKE construction for a group setting would also be a good future work.

Data Availability

The data used to support the findings of this study are included in the article.

Conflicts of Interest

The author declares that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2017R1D1A1B03032424).

References

[1] M. Bellare and P. Rogaway, "Entity authentication and key distribution," in *Proceedings of the Crypto '93*, vol. 773, pp. 232–249, LNCS, Springer-Verlag, 1993, Full version at http://www-cse.ucsd.edu/users/mihir.

[2] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Proceedings of the Eurocrypt'00*, pp. 139–155, LNCS, Springer-Verlag, 2000.

[3] J. Katz, R. Ostrovsky, and M. Yung, "Efficient password-authenticated key exchange using human-memorable passwords," in *Proceedings of the EUROCRYPT'01*, vol. 2045, pp. 475–494, LNCS, 2001.

[4] B. LaMacchia, K. Lauter, and A. Mityagin, "Stronger security of authenticated key exchange," in *Proceedings of the ProvSec07*, vol. 4784, pp. 1–16, LNCS, Springer-Verlag, 2007.

[5] M. Abdalla and D. Pointcheval, "Interactive diffie-hellman assumptions with applications to password-based authentication," in *Proceedings of the FC 2005*, vol. 3570, pp. 341–356, Springer-Verlag, 2005.

[6] M. Abdalla, P.-A. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *Proceedings of the PKC*, vol. 3386, pp. 65–84, Springer-Verlag, 2005.

[7] E. Bresson, O. Chevassut, and D. Pointcheval, "Group diffie-hellman key exchange secure against dictionary attacks," in *Proceedings of the Asiacrypt'02*, vol. 2501, pp. 497–514, LNCS, Springer-Verlag, 2002.

[8] V. Boyko, P. MacKenzie, and S. Patel, "Provably secure password-authenticated key exchange using diffie-hellman," in *Proceedings of Eurocrypt00*, vol. 1807, pp. 156–171, LNCS, Springer-Verlag, 2000.

[9] P. MacKenzie, T. Shrimpton, and M. Jakobsson, "Threshold password-authenticated key exchange," in *Proceedings of the Crypto'02*, vol. 2442, pp. 385–400, LNCS, Springer-Verlag, 2002.

[10] H.-R. Chung, W.-C. Ku, and M.-J. Tsaur, "Weaknesses and improvement of Wang et al.'s remote user password authentication scheme for resource-limited environments," *Computer Standards & Interfaces*, vol. 31, no. 4, pp. 863–868, 2009.

[11] R. Song, "Advanced smart card based password authentication protocol," *Computer Standards & Interfaces*, vol. 32, no. 5-6, pp. 321–325, 2010.

[12] Y.-Y. Wang, J.-Y. Liu, F.-X. Xiao, and J. Dan, "A more efficient and secure dynamic ID-based remote user authentication scheme," *Computer Communications*, vol. 32, no. 4, pp. 583–585, 2009.

[13] M.-S. Hwang, S.-K. Chong, and T.-Y. Chen, "DoS-resistant ID-based password authentication scheme using smart cards," *The Journal of Systems and Software*, vol. 83, no. 1, pp. 163–172, 2010.

[14] J. Xu, W.-T. Zhu, and D.-G. Feng, "An improved smart card based password authentication scheme with provable security," *Computer Standards & Interfaces*, vol. 31, no. 4, pp. 723–728, 2009.

[15] H.-T. Yeh, H.-M. Sun, and T. Hwang, "Security analysis of the generalized key agreement and password authentication protocol," *IEEE Communications Letters*, vol. 5, no. 11, pp. 462-463, 2011.

[16] K.-H. Yeh, C. Su, N. W. Lo, Y. Li, and Y.-X. Hung, "Two robust remote user authentication protocols using smart cards," *The Journal of Systems and Software*, vol. 83, no. 12, pp. 2556–2565, 2010.

[17] G. Yang, D. S. Wong, H. Wang, and X. Deng, "Two-factor mutual authentication based on smart cards and passwords," *Journal of Computer and System Sciences*, vol. 74, no. 7, pp. 1160–1172, 2008.

[18] S. Halevi and H. Krawczyk, "Public-key cryptography and password protocols," *ACM Transactions on Information and System Security*, vol. 2, no. 3, pp. 230–268, 1999.

[19] D. Stebila, P. Udupi, and S. Chang, "Multi-factor password-authenticated key exchange," in *Proceedings of the ACISP10*, vol. 105, pp. 56–66, 2010.

[20] X. Huang, X. Chen, J. Li, Y. Xiang, and L. Xu, "Further observations on smart-card-based password-authenticated key agreement in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 7, pp. 1767–1775, 2014.

[21] W.-S. Juang, S.-T. Chen, and H.-T. Liaw, "Robust and efficient password-authenticated key agreement using smart cards," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 6, pp. 2551–2556, 2008.

[22] D. Z. Sun, J. P. Huai, J. Z. Sun, J. W. Zhang, and Z. Y. Feng, "Improvements of juang et al.'s password-authenticated key agreement scheme using smart cards," *IEEE Transactions on Industrial Electronics*, vol. 56, no. 6, pp. 2284–2291, 2009.

[23] D. Wang, W. Li, and P. Wang, "Measuring two-factor authentication schemes for real-time data access in industrial wireless sensor networks," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 9, pp. 4081–4092, 2018.

[24] F. Wu, L. Xu, S. Kumari, and X. Li, "A new and secure authentication scheme for wirless sensor networks with formal proof," *Peer-to-Peer Networking and Applications*, vol. 32, no. 1, pp. 16–30, 2017.

[25] J. Srinivas, S. Mukhopadhyay, and D. Mishra, "Secure and efficient user authentication scheme for multi-gateway wireless sensor networks," *Ad Hoc Networks*, vol. 54, pp. 147–169, 2017.

[26] D. Pointcheval and S. Zimmer, "Multi-factor authenticated key exchange," in *Proceedings of the ACNS'08*, vol. 5037, pp. 277–295, LNCS, 2008.

[27] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Proceedings of the CRYPTO 1984*, vol. 196, pp. 10–18, Springer-Verlag, 1985.

[28] F. Hao and D. Clarke, "Security analysis of a multi-factor authenticated key exchange protocol," Cryptology ePrint Archive, Report 2012/039, 2012.

[29] C.-I. Fan and Y.-H. Lin, "Provably secure remote truly three-factor authentication scheme with privacy protection on biometrics," *IEEE Transactions on Information Forensics and Security*, vol. 4, no. 4, pp. 933–945, 2009.

[30] Y. Liu, F. Wei, and C. Ma, "Multi-factor authenticated key exchange protocol in the three-party setting," in *Proceedings of the Inscrypt'10*, vol. 6584, pp. 255–267, 2010.

[31] D. He and D. Wang, "Robust biometrics-based authentication scheme for multiserver environment," *IEEE Systems Journal*, vol. 9, no. 3, pp. 816–823, 2015.

[32] F.-S. Wei, Q. Jiang, R.-J. Zhang, and C.-G. Ma, "A privacy-preserving multi-factor authenticated key exchange protocol with provable security for cloud computing," *Journal of Information Science and Engineering*, vol. 33, no. 4, pp. 907–921, 2017.

[33] X. Huang, Y. Xiang, A. Chonka, J. Zhou, and R. H. Deng, "A generic framework for three-factor authentication: Preserving security and privacy in distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 22, no. 8, pp. 1390–1397, 2011.

[34] J. Yu, G. Wang, Y. Mu, and W. Gao, "An efficient generic framework for three-factor authentication with provably secure instantiation," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 12, pp. 2302–2313, 2014.

[35] N. Fleischhacker, M. Manulis, and A. Azodi, "Modular design and analysis framework for multi-factor authentication and key exchange," Cryptology ePrint Archive, Report 2012/181, 2012.

[36] Y. Dodis, L. Reyzin, and A. Smith, "Fuzzy extractors: how to generate strong keys from biometrics and other noisy data," in *Proceedings of the EUROCRYPT'04*, vol. 3027, pp. 523–540, LNCS, Springer, 2004.

[37] A. Sadeghi, I. Visconti, and C. Wachsmann, "PUF-enhanced RFID security and privacy," in *Proceedings of the SECSI 2010*, Germany, 2010.

[38] K. B. Frikken, M. Blanton, and M. J. Atallah, "Robust authentication using physically unclonable functions," in *Proceedings of the ISC'09*, vol. 5735, pp. 262–277, LNCS, Springer, 2009.

[39] C. Brzuska, M. Fischlin, H. Schröder, and S. Katzenbeisser, "Physically unclonable functions in the universal composition framework," in *Proceedings of the CRYPTO'11*, vol. 6841, pp. 51–70, LNCS, 2011.

[40] M. V. Dijk and U. Ruhrmair, "Physical unclonable functions in cryptographic protocols: security proofs and impossibility results," Cryptology ePrint Archive, Report 2012/228, 2012.

[41] A. C. D. Resende, K. Mochetti, and D. F. Aranha, "PUF-based mutual multifactor entity and transaction authentication for secure banking," *LightSec*, vol. 9542, pp. 77–96, 2015.

[42] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian, "Zipf's law in passwords," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 11, pp. 2776–2791, 2017.

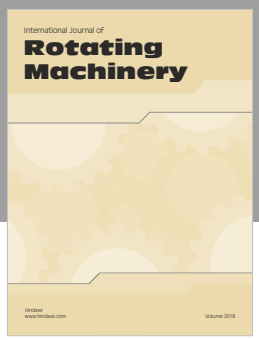
[43] D. Wang and P. Wang, "On the Implications of Zipfs Law in Passwords," in *Proceedings of the ESORICS16'*, vol. 9878, pp. 111–131, 2016.

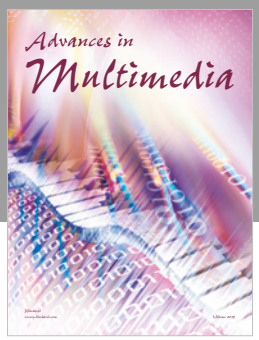
[44] V. Shoup, "Sequences of games: a tool for taming complexity in security proofs," Cryptology ePrint Archive, Report 2004/332, 2004.

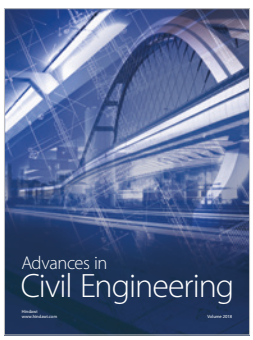
[45] D. Wang and P. Wang, "Two Birds with One Stone: Two-Factor Authentication with Security beyond Conventional Bound," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 708–722, 2018.

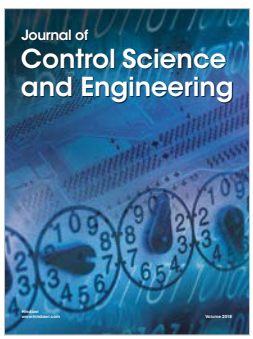
[46] E. Erdem and M. T. Sandikkaya, "OTPass-One Time Password as a Service," *IEEE Trans. on Information Forensics and Security*, vol. 14, no. 3, pp. 743–756, 2019.

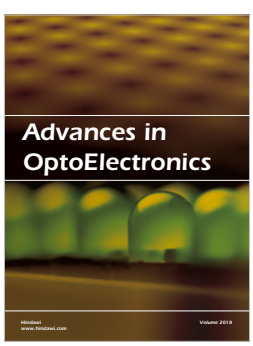
[47] H. Luo, G. Wen, and J. Su, "Lightweight three factor scheme for real-time data access in wireless sensor networks," *Wireless Networks*, pp. 1–16, 2018.

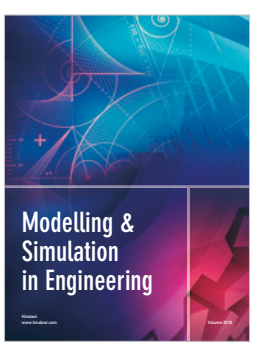


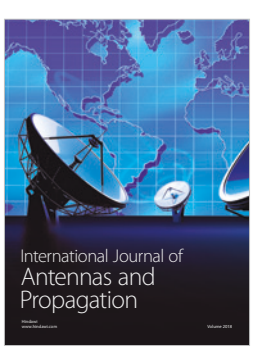


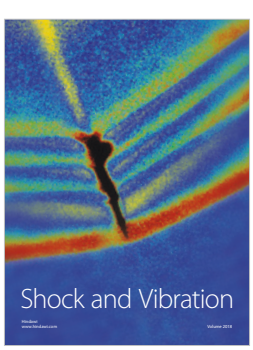


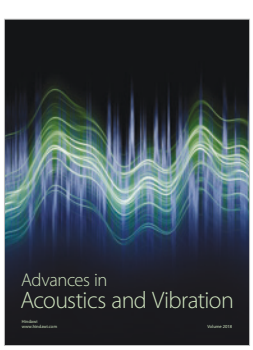












Hindawi

Submit your manuscripts at
www.hindawi.com