Computer Security Symposium 2016
11 - 13 October 2016

匿名パスワード認証における匿名性について

辛 星漢[1,a)]　古原 和邦[1]

概要：本稿では匿名パスワード認証プロトコルにおける匿名性について議論する.

キーワード：パスワード, 認証, 匿名性

How to Fix Client Anonymity in Anonymous Password-based Authentication

SeongHan Shin[1,a)]　Kazukuni Kobara[1]

Abstract: In this paper, we discuss how to fix client anonymity in an anonymous password-based authentication protocol.

Keywords: Password, authentication, anonymity

1. Introduction

An anonymous password-based authentication protocol is designed to provide not only password-based authentication but also client anonymity. However, adding the most widely-used password authentication to anonymity is not trivial at all and would be an important research topic in the forthcoming future. So far, several anonymous password-based authentication protocols [12], [14], [15], [16], [17], [18], [19], [20] have been proposed. These protocols are quite attractive because they do not rely on PKI (Public-Key Infrastructures) and thus can be used in PKI-unavailable situations. Some potential applications of such protocols include whistle-blowing from insiders, questionnaire to qualified people, anonymous counseling and so on.

1.1 Anonymous Password-based Authentication Protocols Requiring Auxiliary Memory Device or Public Directory

The anonymous password-based authentication protocols [12], [19], [20] requiring auxiliary memory device or public directory have been proposed in order to eliminate the linear computation costs to the number of clients on the server side in Anonymous PAKE protocols [14], [15], [16], [17], [18]. In [19], Yang et al., proposed a new anonymous password-based authentication protocol (YZWB09) using the password-protected credentials. The YZWB09 protocol is constructed from Camenisch's signature [4] for clients' authentication credentials, and Paillier encryption [11] for server's homomorphic encryption. For better efficiency, Yang et al., [20] proposed another anonymous password-based authentication protocol (YZWB10) which is based on the BBS+ signature [2] and the ElGamal encryption [5]. In [12], Qian et al., proposed a simple anonymous password-based authentication (SAPAKE) protocol and its extended SAPAKE+ protocol both of which are constructed on a homomorphic

[1] 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology (AIST)
[a)] seonghan.shin@aist.go.jp

public-key encryption scheme. A distinguishing feature of [12], [19], [20] is that the password-protected credentials in [19], [20] and the protocol-specific values in [12] must not require any secure storage facility for usability so that these credentials/values can be stored on any auxiliary memory device or public directory. Recently, Shin et al., [13] showed that the SAPAKE and SAPAKE+ protocols [12] are susceptible to active attacks where an attacker can impersonate the server after modifying the protocol-specific values.

1.2 Our Contributions

In this paper, we revisit the SAPAKE protocol [12]. After describing the SAPAKE protocol [12], we show that it does not provide client anonymity against an outside attacker, who is much weaker than the server. Specifically, the attacker can specify which client has actually communicated with the server in the SAPAKE protocol [12] with probability 1 even though the attacker does not know the server's secrets. Then, we propose a secure anonymous password-based authentication (for short, SAP) protocol that is secure against modification attacks (as in Section 4.2 and [13]) on protocol-specific values. The SAP protocol allows a server to control the number of anonymous client authentication, and is more efficient than SAPAKE [12] in terms of client's computation costs. Also, we prove that the SAP protocol provides client anonymity against an outside adversary and a semi-honest server, who honestly follows the protocol.

2. Preliminaries

In this section, we explain some notations to be used throughout this paper and a public-key encryption scheme that is secure against chosen-plaintext attacks.

2.1 Notations

Let λ be a security parameter. Let $\{0,1\}^*$ be a set of finite binary strings and $\{0,1\}^\lambda$ be a set of binary strings of length λ. Let "$\|$" be a concatenation of binary strings in $\{0,1\}^*$. If U is a set, then $u \xleftarrow{R} U$ indicates the process of selecting u at random and uniformly over U. If U is a function (whatever it is), then $u := U$ indicates the process of assigning a result of U to u. Let D be a dictionary size of passwords. Let C_i, C and S be identities of each client C_i, all clients $\{C_i\}$ and server S, respectively.

2.2 A Public-Key Encryption Scheme

Here, we define the syntax of a public-key encryption scheme and its security notion (i.e., semantic security against chosen-plaintext attacks).

Definition1 (Public-Key Encryption) A public-key encryption scheme is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ such that:

(1) A key generation algorithm Gen takes as input the security parameter 1^λ and outputs a pair of public/private keys (pk, sk).

(2) An encryption algorithm E takes as input a public key pk and a message m from some underlying plaintext space. It outputs a ciphertext $c := \mathsf{E}_{pk}(m)$.

(3) A decryption algorithm D takes as input a private key sk and a ciphertext c, and outputs a message $m := \mathsf{D}_{sk}(c)$ or a special symbol $\perp$ denoting failure.

It is required that $\mathsf{D}_{sk}(\mathsf{E}_{pk}(m)) = m$ except with possibly negligible probability over (pk, sk), output by $\mathsf{Gen}(1^\lambda)$, and any randomness used by E.

Definition2 (CPA Security) A public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ is secure against chosen-plaintext attacks (CPA-secure) if, for an adversary $\mathcal{B}$, there exists a negligible function $\varepsilon(\cdot)$ in the security parameter λ such that

$$\Pr[PKE_{\Pi}^{\mathsf{cpa}}(\mathcal{B}) = 1] \leq 1/2 + \varepsilon(\cdot) \qquad (1)$$

in the experiment $PKE_{\Pi}^{\mathsf{cpa}}(\mathcal{B})$ defined as below:

(1) $\mathsf{Gen}(1^\lambda)$ is run to obtain keys (pk, sk).

(2) Adversary $\mathcal{B}$ is given pk as well as oracle access to $\mathsf{E}_{pk}(\cdot)$. The adversary outputs a pair of messages m_0, m_1 of the same length.

(3) A random bit $b \in \{0,1\}$ is chosen, and then a challenge ciphertext $c := \mathsf{E}_{pk}(m_b)$ is computed and given to $\mathcal{B}$.

(4) $\mathcal{B}$ continues to have access to $\mathsf{E}_{pk}(\cdot)$, and outputs a bit b'.

(5) The output of the experiment is defined to be 1 if $b' = b$, and 0 otherwise.

We denote by $\mathsf{Adv}_{\Pi}^{\mathsf{cpa}}(\mathcal{B}) = 2\Pr[PKE_{\Pi}^{\mathsf{cpa}}(\mathcal{B}) = 1] - 1$ the adversary's advantage in attacking the public-key encryption scheme Π.

Also, we define multiplicative homomorphic property of a public-key encryption scheme as follows. Let R be a set from which a random coin used by E is uniformly chosen, and let $\mathsf{E}_{pk}(m; r)$ be a ciphertext of a message m under a public key pk using a random coin r.

Definition3 (Homomorphic Property) A public-key encryption scheme $\Pi = (\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ is multiplicative homomorphic if, for two arbitrary messages m_1 and m_2 in the plaintext space, it holds that

$$\mathsf{E}_{pk}(m_1; r_1) \otimes \mathsf{E}_{pk}(m_2; r_2) = \mathsf{E}_{pk}(m_1 \cdot m_2; r') \qquad (2)$$

for some $r' \in R$, where both r_1 and r_2 are random coins, and $\otimes$ and $\cdot$ are group operations over ciphertexts and plaintexts, respectively.

3. The SAPAKE Protocol

In this section, we describe the SAPAKE and its extended SAPAKE+ protocols [12] both of which consist of **Setup**, **Registration** and **Authentication** phases.

3.1 Setup

It chooses a finite cyclic group $\mathbb{G}$ generated by g of prime order p, such that $|p| = \lambda$, where λ is the security parameter. Next, it chooses a homomorphic public-key encryption scheme $(\mathsf{Gen}, \mathsf{E}, \mathsf{D})$. Also, it chooses three hash functions $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3 : \{0,1\}^* \rightarrow \{0,1\}^\lambda$. Finally, it publishes (system-wide) public parameters $(\mathbb{G}, p, g, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathsf{Gen}, \mathsf{E}, \mathsf{D})$.

3.2 Registration

First, server S generates a key pair (pk, sk) by invoking $\mathsf{Gen}(1^\lambda)$, and picks an element $N \xleftarrow{R} \mathbb{G}$. Each client C_i randomly chooses his/her password $pw_i \in D$ and sends it to the server. Then, server S computes $T_i := \mathsf{E}_{pk}(N^{1/pw_i}; r_i)$, where $r_i \in R$. Finally, the server releases protocol-specific values $(\{T_i\}, pk)$. Note that password pw_i is kept by client C_i secretly, and $(N \in \mathbb{G}, sk)$ are held by server S secretly.

3.3 Authentication

The authentication phase of SAPAKE consists of four steps as below.

Step 1. When client $C_i (\in C)$ wants to login to server S, the client does the following: 1) The client chooses a random element $x \xleftarrow{R} \mathbb{Z}_p^\star$ and computes $X \equiv g^x$; 2) The client obtains a masked X by setting $\hat{X} := \mathsf{E}_{pk}(X; s) \otimes T_i^{pw_i}$, where $s \in R$; and 3) Finally, the client sends $(C, \hat{X})$ to the server.

Step 2. After receiving a message $(C, \hat{X})$ from client C_i, server S does the following: 1) The server extracts $X' := \mathsf{D}_{sk}(\hat{X})/N$; 2) The server chooses a random element $y \xleftarrow{R} \mathbb{Z}_p^\star$, computes $Y \equiv g^y$, and obtains a Diffie-Hellman key $K' \equiv (X')^y$; 3) The server computes an authenticator $Auth_S := \mathsf{H}_1(trans')$ where $trans' := C \| S \| \hat{X} \| Y \| X' \| K'$; and 4) The server sends back $(S, Y, Auth_S)$ to the client.

Step 3. After receiving a message $(S, Y, Auth_S)$ from server S, client C_i does the following: 1) The client computes a Diffie-Hellman key $K \equiv Y^x$; 2) The client checks whether $Auth_S = \mathsf{H}_1(trans)$ where $trans := C \| S \| \hat{X} \| Y \| X \| K$. If not, the client rejects; 3) Otherwise, the client computes an authenticator $Auth_C := \mathsf{H}_2(trans)$ and a session key $SK_C := \mathsf{H}_3(trans)$; and 4) The client sends $Auth_C$ to the server, and terminates with acceptance.

Step 4. After receiving a message $Auth_C$ from client C_i, server S checks whether $Auth_C = \mathsf{H}_2(trans')$. If not, the server rejects. Otherwise, the server computes a session key $SK_S := \mathsf{H}_3(trans')$ and terminates with acceptance.

In addition to the multiplicative homomorphic property of the public-key encryption scheme $(\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ as in Definition 3, Qian et al., [12] also defined the following: Given a message m and an integer n, it follows that

$$\left(\mathsf{E}_{pk}(m; r_3)\right)^n = \mathsf{E}_{pk}(m^n; r'') \qquad (3)$$

for some $r'' \in R$, where r_3 is a random coin.

As an instantiation of the public-key encryption scheme $(\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ in the SAPAKE protocol, Qian et al., suggested the classical ElGamal encryption [5] because it works well on a cyclic group of prime order and is quite efficient (see Section 2.2 of [12]). Let $(pk \equiv g^z, sk = z \in \mathbb{Z}_p^\star)$ be a key pair of the ElGamal encryption. With this key pair and the homomorphic property,

$$T_i = \mathsf{E}_{pk}\left(N^{1/pw_i}; r_i\right) = \left(g^{r_i}, g^{z \cdot r_i} \cdot N^{1/pw_i}\right) \qquad (4)$$

and

$$\begin{aligned} \hat{X} &= \mathsf{E}_{pk}(X; s) \otimes T_i^{pw_i} \\ &= \mathsf{E}_{pk}(X; s) \otimes \left(\mathsf{E}_{pk}\left(N^{1/pw_i}; r_i\right)\right)^{pw_i} \\ &= (g^s, g^{z \cdot s} \cdot X) \otimes (g^{r_i \cdot pw_i}, g^{z \cdot r_i \cdot pw_i} \cdot N) \\ &= (g^s \cdot g^{r_i \cdot pw_i}, g^{z \cdot s} \cdot X \cdot g^{z \cdot r_i \cdot pw_i} \cdot N) \ . \end{aligned} \qquad (5)$$

This leads to the same Diffie-Hellman key $K = K'$.

Also, Qian et al., proposed an extended SAPAKE+ protocol (see Section 6 of [12]) using an index to find the corresponding ciphertext among ciphertexts for all possible password candidates. Let f be a bijective map $f : D \rightarrow \{1, 2, \cdots, |D|\}$ which determines an index of each password. The difference from the SAPAKE protocol is that server S releases protocol-specific values $(\{T_i := \mathsf{E}_{pk}(N^{1/f^{-1}(i)}; r_i)\}_{1 \le i \le |D|}, pk)$, and accordingly client C_i computes $\hat{X} := \mathsf{E}_{pk}(X; s) \otimes T_{I_i}^{pw_i}$ where $s \in R$ and T_{I_i} is the I_i-th ciphertext of $\{T_i\}_{1 \le i \le |D|}$. In the SAPAKE+ protocol, client C_i keeps $(pw_i, I_i := f(pw_i))$

and server S holds (N, sk, f) secretly where the index I_i, computed by server S in the **Registration** phase, is remembered by client C_i. In the SAPAKE+ protocol, the protocol-specific values are quite large due to $\{T_i\}_{1 \le i \le |D|}$.

4. An Attack on SAPAKE

In [12], Qian et al., claimed that the SAPAKE protocol provides client anonymity against server in the sense that the server cannot specify which client has communicated with the server. In this section, we show that an outside attacker, who is much weaker than the server, can specify which client has actually communicated with the server in the SAPAKE protocol [12].

4.1 Public Parameters vs. Protocol-specific Values

Before showing an attack on the SAPAKE protocol, we explain why 'Public parameters' are different from 'Protocol-specific values'. The former are publicly verifiable parameters, while the latter are publicly unverifiable/uncheckable values and specific to the SAPAKE protocol [12]. For example, the domain parameters $(\mathbb{G}, p, g)$ can be validated with Appendix A of FIPS PUB 186-4 [9] or Chapter 7.2 of ANSI X9.42 [1], and such parameters can be found in Appendix A of [6]. Also, secure hash functions and homomorphic public-key encryption schemes are available in several international standards (e.g., [7], [8], [10]).

However, the ciphertexts $\{T_i\}$ computed with the server's public key pk, element N and client's password pw_i are not verifiable (i.e., whether these are correctly generated or not) from the client's viewpoint. In [12], Qian et al., clearly said that these ciphertexts $\{T_i\}$ can be stored on an auxiliary memory device or a public directory that does not need any security mechanism as in [19], [20] (refer to Section 4.5 and Table V of [12]). Also, the ownership of pk cannot be guaranteed in the SAPAKE protocol because there is no PKI (i.e., no certificate to bind the public key pk to the server).

4.2 On Client Anonymity

As discussed in Section 4.1, an outside attacker can modify the ciphertexts $\{T_i\}$ without the client to be noticed. Here, we show an attack on the SAPAKE protocol [12]. For clarity, suppose that there are only two clients C_1 and C_2 whose corresponding ciphertexts ($T_1 := \mathsf{E}_{pk}(N^{1/pw_1}; r_1)$ and $T_2 := \mathsf{E}_{pk}(N^{1/pw_2}; r_2)$) are entrusted to a public directory.

First, an attacker A chooses an element $M \xleftarrow{R} \mathbb{G}$, computes $\mathsf{E}_{pk}(M; r_2')$ where $r_2' \in R$, and then replaces T_2 with $T_2' := T_2 \otimes \mathsf{E}_{pk}(M; r_2')$.

Below is the authentication phase of the SAPAKE protocol between server S and client C_2, whose corresponding ciphertext is $T_2' := T_2 \otimes \mathsf{E}_{pk}(M; r_2')$. In the authentication phase, attacker A just eavesdrops the communications between client C_2 and server S. Of course, the attacker does not know which client is about to perform the SAPAKE protocol at the starting point of this protocol.

Step 1'. This is the same as **Step 1** of Section 3.3 except that client C_2 computes $\hat{X} := \mathsf{E}_{pk}(X; s) \otimes (T_2')^{pw_2}$ where $s \in R$.

Step 2'. This is the same as **Step 2** of Section 3.3.

Step 3'. After receiving a message $(S, Y, Auth_S)$ from server S, client C_2 does the following: 1) The client computes a Diffie-Hellman key $K \equiv Y^x$; and 2) The client checks whether $Auth_S = \mathsf{H}_1(trans)$ or not where $trans := C \| S \| \hat{X} \| Y \| X \| K$.

If the client terminates the protocol without sending $Auth_C$ (i.e., $Auth_S \neq \mathsf{H}_1(trans)$) in **Step 3'**, the attacker gets to know that the client who has just communicated with server S is client C_2. Otherwise, the attacker comes to a conclusion that the client who has just communicated with server S is client C_1.

Let $(pk \equiv g^z, sk = z \in \mathbb{Z}_p^\star)$ be the server's key pair of the ElGamal encryption. The invalidity of $Auth_S$ in **Step 3'** can be easily checked from the inequality $K(\equiv Y^x \equiv g^{xy}) \neq K'(\equiv (X')^y \equiv g^{xy} \cdot M^{y \cdot pw_2})$ since

$$
\begin{aligned}
\hat{X} &= \mathsf{E}_{pk}(X; s) \otimes (T_2')^{pw_2} \\
&= \mathsf{E}_{pk}(X; s) \otimes (T_2 \otimes \mathsf{E}_{pk}(M; r_2'))^{pw_2} \\
&= (g^s, g^{z \cdot s} \cdot X) \\
&\quad \otimes \left((g^{r_2}, g^{z \cdot r_2} \cdot N^{1/pw_2}) \otimes \left(g^{r_2'}, g^{z \cdot r_2'} \cdot M \right) \right)^{pw_2} \\
&= (g^s \cdot g^{r_2 \cdot pw_2} \cdot g^{r_2' \cdot pw_2}, \\
&\quad g^{z \cdot s} \cdot g^{z \cdot r_2 \cdot pw_2} \cdot g^{z \cdot r_2' \cdot pw_2} \cdot X \cdot N \cdot M^{pw_2}) \\
&\stackrel{\mathsf{def}}{=} \left(\hat{X}_1, \hat{X}_2 \right) \qquad (6)
\end{aligned}
$$

and

$$
X' = \frac{\mathsf{D}_{sk}(\hat{X})}{N} = \frac{\hat{X}_2}{\left(\hat{X}_1\right)^z \cdot N} = X \cdot M^{pw_2} \ . \qquad (7)
$$

4.3 Discussion

In the attack of Section 4.2, the outside attacker can specify client C_1 and C_2 with probability 1 by just eavesdropping the communications between the client and the server after replacing the ciphertext T_2 with T_2'. Also,

this attack does not require the server's secrets (N, sk) and any off-line dictionary attacks on passwords.

The main reason why the attack of Section 4.2 is possible is that the client can not check the integrity of T_2, at the same time, the server can not check whether the correct T_2 is used in the computation of $\hat{X}$ (due to the homomorphic property of the public-key encryption scheme). Therefore, the SAPAKE protocol [12] does *not* provide client anonymity against an outside attacker, who is much weaker than the server.[*1]

A simple countermeasure to the attack of Section 4.2 is to use integrity-preserving memory devices or public directories for storing the protocol-specific values. However, it is contrary to a distinguishing feature of the SAPAKE protocol [12] that the protocol-specific values (including $\{T_i\}$) must not require any secure facility for storage (on the client side) as in [19], [20].

5. A Secure Anonymous Password-based Authentication (SAP) Protocol

In this section, we propose a secure anonymous password-based authentication (for short, SAP) protocol that provides security against modification attacks (as in Section 4.2 and [13]) on protocol-specific values. Specifically, the SAP protocol guarantees not only AKE security against active attacks and modification attacks, but also client anonymity against an outside adversary and a semi-honest server, who honestly follows the protocol. In the SAP protocol, a server can control the number of anonymous client authentication.

5.1 Main Ideas

Here, we explain main ideas of the SAP protocol in order to prevent the modification attacks in Section 4.2 and [13]. As a countermeasure to the attack in Section 4.2, client C_i should check the integrity of protocol-specific values by verifying an authenticator that is sent from server S and computed with the protocol-specific values locally stored by server S. In order to avoid the attack in [13], client C_i should use his/her password pw_i for masking a Diffie-Hellman public value X as well as an element N in a ciphertext T_i. Note that the masking technique used in the SAP protocol is different from that in the SAPAKE protocol [12]. By doing these, we can prevent any effects caused by manipulation of protocol-specific values in the SAP protocol without requiring any secure facility for storing the protocol-specific values.

5.2 The SAP Protocol

The SAP protocol consists of **Setup**, **Registration** and **Authentication** phases.

5.2.1 Setup

It chooses a finite cyclic group $\mathbb{G}$ of prime order p and g is a generator of $\mathbb{G}$, where the operation is denoted multiplicatively.[*2] Let $\mathcal{G}, \mathcal{H}$ be full-domain hash functions mapping $\{0,1\}^* \to \mathbb{G}$. Next, it chooses a multiplicative homomorphic public-key encryption scheme $(\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ that is CPA-secure. Also, it chooses three hash functions $\mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3 : \{0,1\}^* \to \{0,1\}^\lambda$, where λ is the security parameter. Finally, it publishes (system-wide) public parameters $(\mathbb{G}, p, g, \mathsf{H}_1, \mathsf{H}_2, \mathsf{H}_3, \mathcal{G}, \mathcal{H}, \mathsf{Gen}, \mathsf{E}, \mathsf{D})$.

5.2.2 Registration

First, server S generates a key pair (pk, sk) by invoking $\mathsf{Gen}(1^\lambda)$, and picks an element $N \in \mathbb{G}$. Each client C_i chooses his/her password $pw_i \in D$ and sends $W_i := \mathcal{G}(C_i, pw_i)^{-1}$ to the server. Then, server S computes $T_i := \mathsf{E}_{pk}(W_i \cdot \mathcal{H}^k(N); r_i)$ where $k \in \mathbb{N}$ is the number of anonymous client authentication for C, $\mathcal{H}^k(\cdot)$ is a hash chain of length k, and $r_i \in R$.[*3] Finally, the server releases protocol-specific values $(\{T_i\}, pk)$. Note that client C_i remembers his/her password pw_i, and server S holds $(C, k, N, (pk, sk), \{T_i\})$. This registration phase should be done securely between client C_i and server S.

5.2.3 Authentication

The authentication phase of the SAP protocol consists of four steps as below.

Step 1. When client $C_i (\in C)$ wants to login to server S anonymously, the client does the following: 1) The client chooses a random element $x \xleftarrow{R} \mathbb{Z}_p^\star$, and computes a Diffie-Hellman public value $X \equiv g^x$ and its masked value $\bar{X} \equiv X \cdot \mathcal{G}(C_i, pw_i)$; 2) The client computes $\hat{X} := \mathsf{E}_{pk}(\bar{X}; s) \otimes T_i$ where $s \in R$; and 3) Finally, the client sends $(C, \hat{X})$ to the server.

Step 2. After receiving a message $(C, \hat{X})$ from client C_i, server S does the following: 1) If $k < 1$, the server aborts the protocol; 2) Otherwise, the server recovers $X' := \mathsf{D}_{sk}(\hat{X}) / \mathcal{H}^k(N)$; 3) The server chooses a random element $y \xleftarrow{R} \mathbb{Z}_p^\star$, and computes $Y \equiv g^y$ and a Diffie-Hellman key $K' \equiv (X')^y$; 4) The server

[*1] The attack of Section 4.2 is not applicable to the SAPAKE+ protocol because the index for ciphertexts $\{T_i\}$ is hidden with the secret bijective map f. However, the SAPAKE+ protocol is insecure against active attacks [13] (see Section 1.1).

[*2] In the aftermath, all the subsequent arithmetic operations are performed in modulo p unless otherwise stated.

[*3] For example, if $k = 3$ then $\mathcal{H}^3(N) := \mathcal{H}(\mathcal{H}(\mathcal{H}(N)))$.

computes an authenticator $V_S := \mathsf{H}_1(trans')$ where $trans' := C\|S\|\{T_i\}\|pk\|\hat{X}\|Y\|X'\|K'$; and 5) The server sends back (S, Y, V_S) to the client.

Step 3. After receiving a message (S, Y, V_S) from server S, client C_i does the following: 1) The client computes a Diffie-Hellman key $K \equiv Y^x$; 2) Let $trans := C\|S\|\{T_i\}\|pk\|\hat{X}\|Y\|X\|K$. If V_S is not valid (i.e., $V_S \neq \mathsf{H}_1(trans)$), the client aborts the protocol; 3) Otherwise, the client computes an authenticator $V_C := \mathsf{H}_2(trans)$ and a session key $SK_C := \mathsf{H}_3(trans)$; and 4) The client sends V_C to the server, and terminates the protocol with acceptance.

Step 4. After receiving a message V_C from client C_i, server S does the following: 1) If V_C is not valid (i.e., $V_C \neq \mathsf{H}_2(trans')$), the server aborts the protocol; 2) Otherwise, the server computes a session key $SK_S := \mathsf{H}_3(trans')$; 3) The server computes $T_i := T_i \otimes \mathsf{E}_{pk}(\mathcal{H}^{k-1}(N)/\mathcal{H}^k(N); t)$, where $t \in R$, and $k := k - 1$; and 4) The server updates $\{T_i\}$ and k, and terminates the protocol with acceptance.

Note that, if $k = 0$ in **Step 4**, any client $C_i (\in C)$ can no longer be authenticated anonymously with server S.

In the SAP protocol, the public-key encryption scheme $(\mathsf{Gen}, \mathsf{E}, \mathsf{D})$ can be instantiated with the ElGamal encryption [5]. Let $(pk \equiv g^z, sk = z \in \mathbb{Z}_p^\star)$ be a key pair of the ElGamal encryption. With this key pair and the multiplicative homomorphic property (as in Definition 3),

$$\begin{aligned} T_i &= \mathsf{E}_{pk}\left(W_i \cdot \mathcal{H}^k(N); r_i\right) \\ &= \left(g^{r_i}, g^{z \cdot r_i} \cdot W_i \cdot \mathcal{H}^k(N)\right) \end{aligned} \tag{8}$$

and

$$\begin{aligned} \hat{X} &= \mathsf{E}_{pk}(\bar{X}; s) \otimes T_i \\ &= \mathsf{E}_{pk}(X \cdot \mathcal{G}(C_i, pw_i); s) \otimes \mathsf{E}_{pk}\left(W_i \cdot \mathcal{H}^k(N); r_i\right) \\ &= (g^s, g^{z \cdot s} \cdot X \cdot \mathcal{G}(C_i, pw_i)) \\ &\quad \otimes \left(g^{r_i}, g^{z \cdot r_i} \cdot W_i \cdot \mathcal{H}^k(N)\right) \\ &= \left(g^s \cdot g^{r_i}, g^{z \cdot s} \cdot X \cdot g^{z \cdot r_i} \cdot \mathcal{H}^k(N)\right) \ . \end{aligned} \tag{9}$$

This leads to the same Diffie-Hellman key $K = K'$.

5.3 Discussions

In this subsection, we discuss security of the SAP protocol against the modification attacks in Section 4.2 and [13].

The attack in Section 4.2 is not applicable to the SAP protocol. If an attacker adds any modifications to the protocol-specific values $\{T_i\}$ as in Section 4.2, client C_i aborts the protocol due to the invalidity of V_S (i.e., $V_S \neq \mathsf{H}_1(trans)$) without sending out V_C to server S. The key point is that the invalidity of V_S is always checked by client C_i regardless of the values X and K.

Also, the SAP protocol is secure against the attack in [13]. Even if an attacker changes the protocol-specific values $(\{T_i\}, pk)$ to another ones as in [13], the attacker can not recover X from $\hat{X}$ with the probability better than that of on-line dictionary attacks since X is masked with $\mathcal{G}(C_i, pw_i)$.

5.4 Update of Protocol-specific Values

In this subsection, we describe how to update protocol-specific values in the SAP protocol depending on several situations.

5.4.1 When $k = 0$

If $k = 0$ and server S wants to provide additional anonymous authentication services to C, the server computes

$$\begin{aligned} T_i &:= T_i \otimes \mathsf{E}_{pk}(\mathcal{H}^{k_{\mathsf{new}}}(N_{\mathsf{new}})/\mathcal{H}^0(N); t) \\ k &:= k_{\mathsf{new}} \\ N &:= N_{\mathsf{new}} \end{aligned}$$

and updates $\{T_i\}$, k and N.

5.4.2 When Client is Revocated

If revocation of client C_j happens, the server computes

$$\begin{aligned} T_i &:= T_{i \setminus j} \otimes \mathsf{E}_{pk}(\mathcal{H}^{k-1}(N)/\mathcal{H}^k(N); t) \\ k &:= k - 1 \end{aligned}$$

and updates $\{T_i\}$ and k.

5.4.3 When Client Joins

If joining of client C_j happens, the server computes

$$\begin{aligned} T_i &:= T_i \otimes \mathsf{E}_{pk}(\mathcal{H}^{k-1}(N)/\mathcal{H}^k(N); t) \\ &\qquad \bigcup \mathsf{E}_{pk}(W_j \cdot \mathcal{H}^{k-1}(N); r_j) \\ k &:= k - 1 \end{aligned}$$

and updates $\{T_i\}$ and k.

6. Security

After explaining the computational Diffie-Hellman (CDH) problem, we show that the SAP protocol of Section 5.2 is provably secure in the random oracle model [3] under the CDH problem.

6.1 Computational Assumption

Here, we explain the computational Diffie-Hellman (CDH) problem on which the SAP protocol is based.

Definition4 (CDH Problem) Let $\mathbb{G}$ be a finite cyclic group of prime order p with g as a generator. A

(t_1, ε_1)-$\mathsf{CDH}_{\mathbb{G},p,g}$ attacker is a probabilistic polynomial time (PPT) machine $\mathcal{B}$, running in time t_1, such that its success probability $\mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G},p,g}(\mathcal{B})$, given random elements g^α and g^β to output $g^{\alpha\beta}$, is greater than ε_1. We denote by $\mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G},p,g}(t_1)$ the maximal success probability over every adversaries, running within time t_1. The CDH problem states that $\mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G},p,g}(t_1) \leq \varepsilon_1$ for any t_1/ε_1 not too large.

6.2 Security Proofs

In this subsection, we prove that the SAP protocol is AKE secure in the random oracle model [3] under the CDH problem, and provides client anonymity against an outside adversary and a semi-honest server.

Theorem6.1 Let P be the SAP protocol where passwords are independently chosen from a dictionary of size D. For any adversary $\mathcal{A}$ within a polynomial time t, with less than q_{send} active interactions with the parties (Send-queries), q_{execute} passive eavesdroppings (Execute-queries) and asking q_{hashH} hash queries to any H_l, for $l = 1, 2, 3$, respectively,

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ake}}_P(\mathcal{A}) \leq\ & \frac{2(q_{\mathsf{sendC}} + 3q_{\mathsf{sendS}})}{D} + 2n \cdot \mathsf{Adv}^{\mathsf{cpa}}_{\Pi}(\mathcal{B}) \\ & + 6q^2_{\mathsf{hashH}} \times \mathsf{Succ}^{\mathsf{cdh}}_{\mathbb{G},p,g}(t_1 + 3\tau_e) \\ & + \frac{(q_{\mathsf{execute}} + q_{\mathsf{send}})^2}{|\mathbb{G}|} + \frac{2q_{\mathsf{send}} + q^2_{\mathsf{hashH}}}{2^\lambda}\end{aligned} \tag{10}$$

where (1) q_{sendC} (resp., q_{sendS}) is the number of Send-queries to C_i (resp., S) instance, (2) n is the number of $\{T_i\}$, (3) λ is the security parameter for the hash functions, and (4) τ_e denotes the computational time for an exponentiation in $\mathbb{G}$.

This theorem shows that the SAP protocol is secure against off-line dictionary attacks since the advantage of the adversary essentially grows with the ratio of interactions to the number of passwords. Due to the lack of space, we omit the proof of Theorem 6.1.

Theorem6.2 The SAP protocol provides client anonymity against an outside adversary.

Proof. In this proof, we show that the outside adversary's advantage $\mathsf{Adv}^{\mathsf{ano}}_P(\mathcal{A}_{\mathsf{ano}})$ is upper-bounded by the AKE advantage $\mathsf{Adv}^{\mathsf{ake}}_P(\mathcal{A})$. First, we choose either C_i or $C_{j\neq i}$ with the probability 1/2, and then simulate all the instances of P for the chosen pair C_i (or $C_{j\neq i}$) and S. Let bre-ano be an event that $\mathcal{A}_{\mathsf{ano}}$ breaks client anonymity for the given pair. Also, let S be an event that $\mathcal{A}$ correctly guesses the bit b, involved in the Test-query, and let $\neg$S be its complement event.

$$\begin{aligned}\mathsf{Adv}^{\mathsf{ano}}_P(\mathcal{A}_{\mathsf{ano}}) &= 2\Pr[\mathsf{breaking\text{-}anonymity}] - 1 \\ &= 2\left(\frac{1}{2}\Pr[\mathsf{bre\text{-}ano}|\mathsf{S} \vee \neg\mathsf{S}]\right) - 1 \\ &= \Pr[\mathsf{bre\text{-}ano}|\mathsf{S}] + \Pr[\mathsf{bre\text{-}ano}|\neg\mathsf{S}] - 1 \\ &\leq \Pr[\mathsf{S}] + \Pr[\mathsf{bre\text{-}ano}|\neg\mathsf{S}] - 1 \\ &= \Pr[\mathsf{S}] - \frac{1}{2} = \frac{1}{2}\left(2\Pr[\mathsf{S}] - 1\right) \\ &= \frac{1}{2}\mathsf{Adv}^{\mathsf{ake}}_P(\mathcal{A})\ . \end{aligned} \tag{11}$$

Note that $\Pr[\mathsf{bre\text{-}ano}|\neg\mathsf{S}] = 1/2$ since $\mathcal{A}_{\mathsf{ano}}$ cannot distinguish C_i and $C_{j\neq i}$ in the event bre-ano$|\neg$S better than 1/2. □

Theorem6.3 The SAP protocol provides unconditional client anonymity against a semi-honest server.

Proof. Let us consider server S who honestly follows the SAP protocol, but it is curious about client's identity involved with the protocol. It is obvious that server S cannot get any information about the client's identity C_i since the $\hat{X}$ (actually, X) has a unique discrete logarithm of g and, with the randomly-chosen element x, it is the uniform distribution over $\mathbb{G}$. Also, the authenticator V_C does not reveal any information about the client's identity from the fact that the probability, for any clients, to compute the Diffie-Hellman key K is equal. Therefore, $\mathsf{Dist}[P(C_i, S)] = \mathsf{Dist}[P(C_j, S)]$ for any two clients C_i and $C_{j\neq i}$. □

7. Comparison

In this section, we compare security and efficiency of the SAPAKE [12] and SAP (Section 5.2) protocols, both of which are instantiated with the ElGamal encryption [5], in terms of computation and communication costs.

In Table 1, we summarize comparative results of the SAPAKE [12] and SAP (Section 5.2) protocols where '$\mathsf{Exp}_{\mathbb{G}}$' indicates a modular exponentiation in $\mathbb{G}$, '$|c|$' indicates a bit-length of c, and the parentheses mean the remaining number of modular exponentiations after excluding those that are pre-computable. The 'SAP w/o control of client authentication number' protocol can be obtained from the SAP (Section 5.2) protocol by setting $k = 1$, and removing the check of k in **Step 2** and the update of $\{T_i\}$ and k in **Step 4**.

With respect to computation costs, client C_i (resp., server S) in the SAP protocol needs to compute 4 (resp., 5) modular exponentiations in $\mathbb{G}$. In case of the 'SAP w/o control of client authentication number' protocol, client C_i (resp., server S) needs to compute 4 (resp., 3) modular exponentiations in $\mathbb{G}$. When pre-computation is allowed,

表 1 Comparison between the SAPAKE [12] and SAP (Section 5.2) protocols

Protocols	Security against modification attacks in Section 4.2 and [13]	Computation costs		Communication costs*1
		Client C_i	Server S	
SAPAKE [12]	insecure	$6\mathsf{Exp}_{\mathbb{G}}$ ($4\mathsf{Exp}_{\mathbb{G}}$)	$3\mathsf{Exp}_{\mathbb{G}}$ ($2\mathsf{Exp}_{\mathbb{G}}$)	$3\lvert\mathbb{G}\rvert + 2\lvert\mathsf{H}\rvert$ (3 moves)
SAP	secure	$4\mathsf{Exp}_{\mathbb{G}}$ ($2\mathsf{Exp}_{\mathbb{G}}$)	$5\mathsf{Exp}_{\mathbb{G}}$ ($2\mathsf{Exp}_{\mathbb{G}}$)	
SAP w/o control of client authentication number	secure	$4\mathsf{Exp}_{\mathbb{G}}$ ($2\mathsf{Exp}_{\mathbb{G}}$)	$3\mathsf{Exp}_{\mathbb{G}}$ ($2\mathsf{Exp}_{\mathbb{G}}$)	

*1: The bit-length of identities is excluded

the remaining costs of client C_i (resp., server S) in the SAP and 'SAP w/o control of client authentication number' protocols are 2 (resp., 2) modular exponentiations in $\mathbb{G}$. With respect to communication costs, the SAP and 'SAP w/o control of client authentication number' protocols need a bandwidth of $(3|\mathbb{G}| + 2|\mathsf{H}|)$-bits except the bit-length of identities C and S. From Table 1, we can see that the SAP and 'SAP w/o control of client authentication number' protocols are more efficient than SAPAKE [12] in terms of client's computation costs. However, the SAPAKE protocol [12] is not secure against the modification attacks in Section 4.2 and [13].

謝辞 This work was partly supported by JST, Infrastructure Development for Promoting International S&T Cooperation and JSPS KAKENHI Grant Number JP16H02834.

参考文献

[1] ANSI X9.42, "Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography," 2003.

[2] M. H. Au, W. Susilo, and Y. Mu, "Constant-Size Dynamic k-TAA," In *Proc. of SCN 2006*, LNCS 4116, pp. 111-125, Springer-Verlag, 2006.

[3] M. Bellare and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols," In *Proc. of ACM CCS'93*, pp. 62-73. ACM, 1993.

[4] J. Camenisch and A. Lysyanskaya, "A Signature Scheme with Efficient Protocols," In *Proc. of SCN 2002*, LNCS 2576, pp. 268-289, Springer-Verlage, 2002.

[5] T. ElGamal, "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, Vol. 31, Issue 4, pp. 469-472, 1985.

[6] D. Gillmor, "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for TLS," IETF Internet-Draft, March 2015. Available at `https://tools.ietf.org/html/draft-ietf-tls-negotiated-ff-dhe-08`.

[7] ISO/IEC 18033-6, "Information Technology − Encryption Algorithms − Part 6: Homomorphic Encryption," Under development. Available at `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=67740`.

[8] NIST FIPS PUB 180-4, "Secure Hash Standard (SHS)," March 2012. Available at `http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf`.

[9] NIST FIPS PUB 186-4, "Digital Signature Standard (DSS)," July 2013. Available at `http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf`.

[10] NIST FIPS PUB 202 (Draft), "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," May 2014. Available at `http://csrc.nist.gov/publications/drafts/fips-202/fips_202_draft.pdf`.

[11] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," In *Proc. of EUROCRYPT'99*, LNCS 1592, pp. 223-238, Springer-Verlage, 1999.

[12] H. Qian, J. Gong, and Y. Zhou, "Anonymous Password-based Key Exchange with Low Resources Consumption and Better User-friendliness," *Security and Communication Networks*, Vol. 5, pp. 1379-1393, 2012.

[13] S. H. Shin and K. Kobara, "On the Security of SAPAKE," *IEICE Technical Report*, ISEC2015-10, pp. 9-14, 2015.

[14] S. H. Shin, K. Kobara, and H. Imai, "A Secure Threshold Anonymous Password-Authenticated Key Exchange Protocol," In *Proc. of IWSEC 2007*, LNCS 4752, pp. 444-458, Springer-Verlag, 2007.

[15] S. H. Shin, K. Kobara, and H. Imai, "Very-Efficient Anonymous Password-Authenticated Key Exchange and Its Extensions," In *Proc. of AAECC 2009*, LNCS 5527, pp. 149-158, Springer-Verlag, 2009.

[16] S. H. Shin, K. Kobara, and H. Imai, "Threshold Anonymous Password-Authenticated Key Exchange Secure against Insider Attacks," *IEICE Transactions on Information and Systems*, Vol. E94-D, No. 11, pp. 2095-2110, 2011.

[17] D. Q. Viet, A. Yamamura, and H. Tanaka, "Anonymous Password-Based Authenticated Key Exchange," In *Proc. of INDOCRYPT 2005*, LNCS 3797, pp. 244-257, Springer-Verlag, 2005.

[18] J. Yang and Z. Zhang, "A New Anonymous Password-Based Authenticated Key Exchange Protocol," In *Proc. of INDOCRYPT 2008*, LNCS 5365, pp. 200-212, Springer-Verlag, 2008.

[19] Y. Yang, J. Zhou, J. W. Wong, and F. Bao, "A New Approach for Anonymous Password Authentication," In *Proc. of 2009 Annual Computer Security Applications Conference (ACSAC 2009)*, pp. 199-208, 2009.

[20] Y. Yang, J. Zhou, J. W. Wong, and F. Bao, "Towards Practical Anonymous Password Authentication," In *Proc. of 2010 Annual Computer Security Applications Conference (ACSAC 2010)*, pp. 59-68, 2010.