Journal of Engineering and Applied Sciences 13 (15): 6211-6219, 2018
ISSN: 1816-949X

Insecure Instantiations of Random Oracles in Password-Based Key Exchange Protocols

Juryon Paik
Department of Digital Information and Statistics, Pyeongtaek University,
3825 Seodong-daero, Pyeongtaek, 17869 Gyeonggido, South Korea

Abstract: Protocols for Password-based Authenticated Key Exchange (PAKE) allow users to generate a shared secret key from their easy-to-remember passwords but at the same time have to protect the user's passwords from the notorious dictionary attacks. PAKE protocols often use a hash function that maps user passwords into elements of the underlying cyclic group G generated by an arbitrary fixed element g∈G. Such a hash function is usually modelled as a random oracle G in proofs of security of protocols. One obvious way of instantiating the random oracle G is to use a random oracle H: $\{0, 1\}^* \rightarrow Z_q$ and then define $G(.) = g^{H(.)}$. However, we argue that this obvious instantiation of G is likely to result in a critical vulnerability for most of PAKE protocols. In the present research, we provide a strong evidence in support of this argument by showing that two popular protocols-Bresson two-party PAKE protocol and Abdalla and Pointcheval's three-party PAKE protocol-become susceptible to an offline dictionary attack as soon as G is instantiated as $G(.) = g^{H(.)}$. Our result suggests that designers of PAKE protocols should clearly specify how G can be securely instantiated for their protocols in order to prevent protocol implementers from employing an insecure instantiation of G.

Key words: Authenticated key exchange, password, random oracle, dictionary attack, prevent protocol implementers, PAKE protocols, pointcheval's

INTRODUCTION

Passwords have shown incredible persistence as an almost universal means of authentication over the Internet. Despite countless attempts to dislodge them, passwords are more widely used and firmly entrenched than ever, protecting thousands of millions of internet accounts everyday. The popularity of passwords has promoted extensive research on the design of cryptographic protocols for Password-based Authenticated Key Exchange (PAKE) (Bellovin and Merritt, 1992; Bellare and Rogaway, 1993; Bresson *et al*., 2004; Abdalla and Pointcheval, 2005, 2015; Nam *et al*., 2009, 2013; Katz *et al*., 2009; Goyal *et al*., 2010; Canetti *et al*., 2012; Xiong *et al*., 2013; Katz and Vaikuntanathan, 2013; Yi *et al*., 2013). In the setting where passwords are the only long-term secrets pre-established between users (Bellovin and Merritt, 1992) proposed the first PAKE protocols known as encrypted key exchange which illustrated that:

> Arbitrary two users who share only a weak (e.g., short) password can agree on a cryptographically strong secret key (called a session key) over an insecure public network which might be controlled by an adversary

Since, the publication of encrypted key exchange (with only heuristic security arguments), many practical and provably secure PAKE protocols have been proposed over the last two decades (Bellare and Rogaway, 1993; Bresson *et al*., 2004; Canetti *et al*., 2012; Katz and Vaikuntanathan, 2013).

Key exchange protocols (including PAKE protocols) are often designed using cryptographic hash functions both for efficiency and for security reasons. Such protocols are usually proven secure in the random oracle model (Bellare and Rogaway, 1993) where in hash functions are modeled as random functions. A proof of security in the random oracle model definitely provides a compelling argument in support of the security of a protocol but does not guarantee that the protocol is indeed secure in the real world. A protocol proven secure in the random oracle model can only be claimed to the extent that it is resistant to generic attacks that do not exploit a specific instantiation of the random oracle.

PAKE protocols sometimes make use of a hash function G to map user passwords pw (possibly concatenated with additional information such as user identifiers) into elements of the underlying cyclic group G which is typically defined as a subgroup of order q in Z^*_p where p and q are two large primes such that p = rq+1 for

some value r co-prime to Bellare and Rogaway (1993), Bresson *et al.* (2004) and Abdalla and Pointcheval (2005). In those PAKE protocols, the hash value G(pw) serves as a mask generation function and is typically multiplied by a protocol message such as an ephemeral Diffie-Hellman value. Such hash function, G: {0, 1}*→G is modelled as a random oracle in security proofs and in practice can be instantiated in various ways. Let g be a generator of the cyclic group G. Then, a straightforward instantiation of G is to choose a hash function $\mathcal{H}$: {0, 1}*→Z_q and then define G(.) = $g^{\mathcal{H}(.)}$. In this instantiation, G certainly becomes a random oracle if $\mathcal{H}$ is a random oracle. Let r be as defined above. Another, less straightforward, instantiation of G is to choose a hash function $\mathcal{H}$: {0, 1}*→Z^*_p and then define G(.) = $\mathcal{H}$(.). We refer the readers to MacKenzie for other examples of instantiation of G.

Instantiating the G-oracle as G(.) = $g^{\mathcal{H}(\bullet)}$(which we will refer to simply as "the bad G-oracle instantiation") is likely to result in insecure PAKE protocols. In this study, we show this by examining the security of some published PAKE protocols under the assumption of the bad G-oracle instantiation. We use two popular protocols as case studies. The two-party PAKE protocol (named OMDHKE) of Bresson *et al.* (2004) and the three-party PAKE protocol (named 3PAKE) due to Abdalla and Pointcheval (2005). Our case studies show that when the G -oracle is instantiated as G (.) = $g^{\mathcal{H}(.)}$ both OMDHKE and 3PAKE protocols become vulnerable to an offline dictionary attack and due to this vulnerability both the protocols are rendered insecure in the widely accepted security model of Bellare *et al.* (2000). Based on the studies, we suggest that designers of PAKE protocols should clearly specify how the G-oracle can be securely instantiated for their protocols in order to prevent protocol implementers from employing the bad G-oracle instantiation and other potentially insecure ones.

The OMDHKE protocol originates from AuthA which is an efficient PAKE protocol considered for standardization by the IEEE P1363 standard working group (Bellare and Rogaway, 1993). The AuthA protocol is open-ended in its use of the encryption primitive. It allows the encryption primitive to be instantiated either via a password-keyed symmetric cipher or a mask generation function (e.g., the G-oracle) whose output is multiplied by the plaintext. OMDHKE was proposed to prove that AuthA is secure in the random oracle model when the encryption primitive is realized by a mask generation function. OMDHKE is two-round two-message protocol with one message being sent in each round. The first-round message is simply the product of a Diffie-Hellman value and the hash value G(pw). In our dictionary attack against OMDHKE, the attacker forges the first-round message and exploits the second-round message as the password verifier.

The 3PAKE protocol is based on the AuthA protocol (Bellare and Rogaway, 1993), the PAK suite (MacKenzie) and the OMDHKE protocol (Bresson *et al.*, 2004) which in turn are based on the EKE protocol of Bellovin and Merritt (1992). The design goal of 3PAKE is to achieve both efficiency and security in the three-party setting, where two clients wishing to establish a session key do not share the same password but hold their individual password shared only with a trusted server. One of the biggest challenges in designing a three-party PAKE protocol is to prevent insider dictionary attacks in which the attacker could be any malicious client who wants to find out the password of its partner client. Although, 3PAKE was originally claimed to be provably secure in the random oracle model, several security-related flaws in its design have been disclosed by later studies assummarized below.

Choo *et al.* (2005) showed that 3PAKE is susceptible to an unknown key share attack in a model that allows the adversary to corrupt the protocol participants. This weakness was attributed to a specification error in the protocol description, the identities of the clients were mistakenly omitted from the input of two hash functions (denoted as G_1 and G_2 in our description of 3PAKE. The researcher corrected this error in a new version of their study which is available at http://www.di.ens.fr/~-mabdalla. Our description of 3PAKE follows the corrected specification of the protocol.

Szydlo (2006) demonstrated that, the chosen-basis decisional Diffie-Hellman assumption (on which the security of 3PAKE relies) is not a sound basis for a security proof. Although, Szydlo (2006)'s observation does not translate to a direct attack on 3PAKE, it implies that the existence of a proof of security for 3PAKE is an open question.

Wang and Hu (2006) pointed out that, 3PAKE suffers from an undetectable online dictionary attack in which each guess on the password is checked undetectably via. an online transaction with the server. This weakness of 3PAKE reinforces the fact that no three-party PAKE protocol can resist an undetectable online dictionary attack unless the protocol provides client-to-server authentication.

Recently, Nam *et al.* (2013) have revealed that 3PAKE is also susceptible to an offline dictionary attack in the presence of a malicious client who can set up a normal protocol session with other clients.

But no previous research has discussed security issues related to the instantiation of the G-oracle in

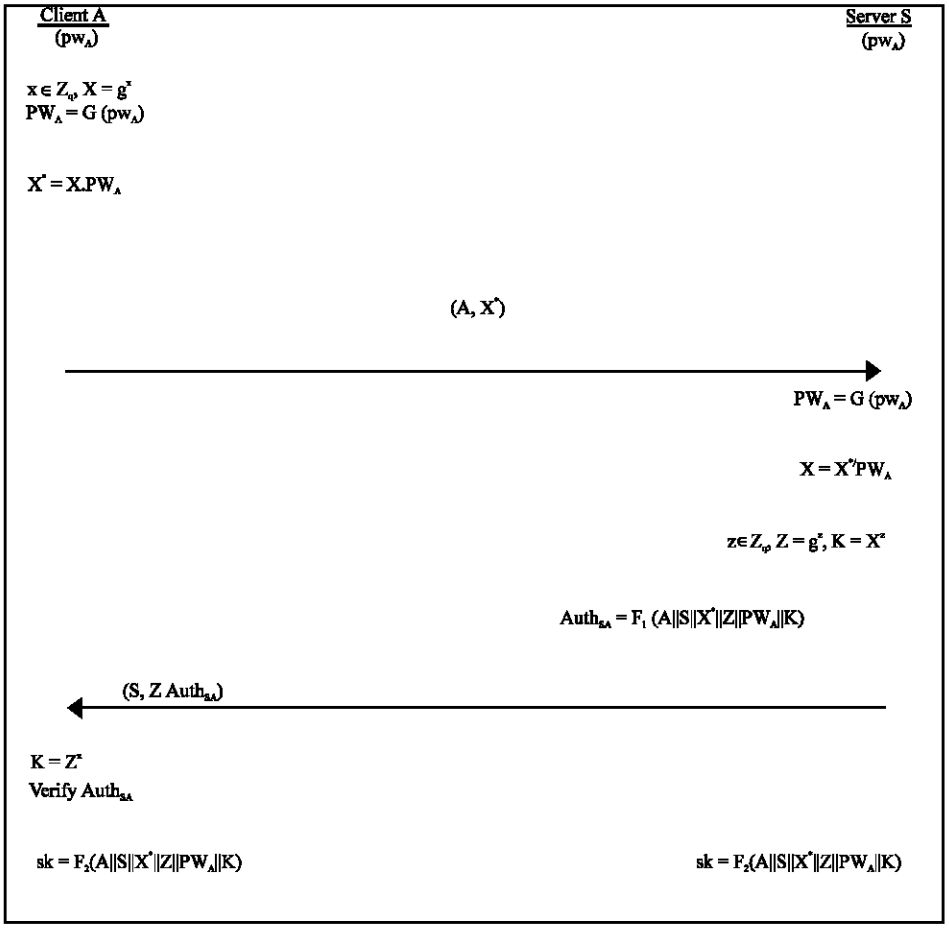

Fig. 1: OMDHKE; Bresson *et al*. (2004) PAKE protocol

3PAKE. Under the assumption of the bad G-oracle instantiation our dictionary attack on 3PAKE can be mounted by any malicious client against any other clients and does not even require the participation of the victim.

Case study; OMDHKE: By Bresson *et al*. (2004) OMDHKE protocol the client encrypts its ephemeral Diffie-Hellman value using the G-oracle as a mask generation function. In this study, we show that the OMDHKE protocol is not secure against an offline dictionary attack if the G-oracle is instantiated as $G(.) = g^{\mathcal{H}(\bullet)}$.

Protocol description: The OMDHKE protocol proceeds as in Fig. 1. The arithmetic is in a finite cyclic group G of prime order q. Let, g be a generator of G. OMDHKE uses three hash functions:

- G: $\{0, 1\}^* \rightarrow G$
- F_1: $\{0, 1\}^* \rightarrow \{0, 1\}^k$ where K is the bit-length of the authenticator $Auth_{SA}$ (see the protocol description below)
- F_2: $\{0, 1\}^* \rightarrow \{0, 1\}^l$ where l is the bit-length of session keys

The client A and the server S are assumed to have pre-established a shared password pw_A. The protocol starts when A chooses a random $x \in Zq$, computes $X = g^x$, $PW_A = G(pw_A)$ and $X^* = X.PW_A$ and sends $\langle A, X^* \rangle$ to S. Upon receiving the message $\langle A, X^* \rangle$ from A, S computes $PW_A = G(pw_A)$ and $X = X^*/PW_A$, chooses a random $z \in Zq$ and computes $Z = g^z$, $K = X^z$ and $Auth_{SA} = F_1(A||S||X^*||Z||PW_A||K)$. Then S sends $\langle S, Z, Auth_{SA} \rangle$ to A and computes the session key, $sk = F_2(A||S||X^*||Z||PW_A||K)$. A computes $K = Z^x$, verifies $Auth_{SA}$ in the straightforward way and computes the session key $sk = F_2(A||S||X^*||Z||PW_A||K)$ if the verification succeeds. OMDHKE provides server-to-client authentication via. the authenticator $Auth_{SA}$ sent in the second round.

Dictionary attack: The OMDHKE protocol described above was proven secure under the assumption that the hash functions G, F_1 and F_2 are random oracles. Assume

a hash function $\mathcal{H}$: $\{0, 1\}^* \rightarrow Zq$ and consider the case that G is instantiated as $G(.) = g^{\mathcal{H}(\cdot)}$. Then, it is clear that G is a random oracle if $\mathcal{H}$ is a random oracle. This instantiation makes OMDHKE vulnerable to an offline dictionary attack as described.

Step 1: As a preliminary research, the attacker B chooses a random $y \in Zq$ and computes $Y = g^y$.

Step 2: B intercepts the first protocol message $\langle A, X^* \rangle$ from A to S and sends, instead, the forged message $\langle A, Y \rangle$ to S as if it is from A. Since, X^* was replaced by Y, the server S will compute X and K as:

$$\begin{aligned} X &= X^*/PW_A \\ &= Y/g^{H(pwA)} \\ &= g^{y-H(pwA)} \\ K &= Xz \\ &= g^{z(y-H(pwA))} \end{aligned}$$

Step 3: B then intercepts the message $\langle S, Z, Auth_{SA} \rangle$ from S to A.

Step 4: Now, B makes a guess pw_A' on the password pw_A and computes:

$$\begin{aligned} PW_A' &= gH(pwA'), \\ K' &= Zy\text{-}H(pwA') \\ &= gz(y\text{-}H(pwA')), \\ Auth'SA &= F1(A||S||Y||Z||PWA'||K') \end{aligned}$$

Step 5: B checks the correctness of pw_A' by comparing $Auth'_{SA}$ against $Auth_{SA}$. Note that, $Auth'_{SA}$ is equal to $Auth_{SA}$ (with an overwhelming probability) if and only if pw_A' and pw_A are equal.

Step 6: B repeats steps 4 and 5 until the correct password is found.

Having not received the server's message, the client A will abort the protocol after a certain amount of time. This offline dictionary attack can have devastating implications for all clients registered with the server, since, the victim could be any of the clients and the attacker need not be a registered client. Given the security threat, we recommend that implementers of OMDHKE should not instantiate the hash function G as $G(.) = g^{\mathcal{H}(\cdot)}$.

Case study; 3PAKE: We use the 3PAKE protocol, Abdalla and Pointcheval (2005)'s three-party PAKE protocol as another case study of insecure instantiations of hash functions. Our result, here is that the security of 3PAKE against an offline dictionary attack also depends on the instantiation of a hash function.

Protocol description: The 3PAKE protocol runs among the three participants, a trusted server S and two clients A and B. The server S assists the clients A and B in establishing a session key by providing them with a central authentication service. Each client holds their individual password shared only with the authentication server S via. a secure channel. Let pw_A and pw_B be the passwords of A and B, respectively. The public system parameters defined for 3PAKE are:

- A large cyclic group G with prime order q and an arbitrary fixed generator g of the group G
- Two hash functions G_1 and G_2 which outputs the elements of the cyclic group G. G_1 and G_2 are both modeled as random oracles in the security proof of 3PAKE
- A hash function F which outputs l bit strings. Here, l is a security parameter representing the bit-length of session keys. F is also modeled as a random oracle

3PAKE works as follows: Client A chooses a random $x \in Zq$ and computes $X = g^x$, $PW_{A,1} = G_1(A, B, pw_A)$ and $X^* = X.PW_{A,1}$. Then A sends $\langle X^* \rangle$ to the server S.

Meanwhile, client B chooses a random $y \in Zq$, computes $Y = g^y$, $PW_{B,1} = G_1(A, B, pw_B)$ and $Y^* = Y.PW_{B,1}$ and sends $\langle Y^* \rangle$ to S.

After receiving $\langle X^* \rangle$ and $\langle Y^* \rangle$, S first recovers X and Y by computing $X = X^*/G_1(A, B, pw_A)$ and $Y = Y^*/G_1(A, B, pw_B)$. Next, S selects a random element $z \in Zq$ and a random string $R \in \{0, 1\}^k$ where K is a security parameter which determines the bit-length of R. S then computes:

$$\begin{aligned} \bar{X} &= X^z \\ \bar{Y} &= Y^z \\ PW_{A,2} &= g_2(A.B.R.\, pw_A, X^*) \\ PW_{B,2} &= g_2(A.B.R.\rho w_B, Y^*) \\ \bar{X}^* &= \bar{X}.PW_{B.2} \\ \bar{Y}^* &= \bar{Y}.PW_{A,2} \end{aligned}$$

and sends $\langle R, Y^*, \bar{X}^*, \bar{Y}^* \rangle$ and $\langle R, X^*, \bar{X}^*, \bar{Y}^* \rangle$ to A and B, respectively. Upon receiving $\langle R, Y^*, \bar{X}^*, \bar{Y}^* \rangle$ from S, A computes:

$$PW_{A,2} = G_2(A, B, R, pwA, X^*)$$

$$\overline{Y} = \left(\frac{\overline{Y}}{PW_{A,2}}\right),$$

$$K = \overline{Y}^x$$

A then defines the transcript $T = R||X^*||Y^*||\overline{X}^*||Y^*$ and computes the session key $sk = F(A||B||S||T||K)$. Upon $\langle R, X^*, \overline{X}^*, \overline{Y}^*\rangle$ receiving from S, B computes:

$$PW_{B,2} = G(A, B, R, p\omega B, Y^*)$$

$$\overline{X} = \left(\frac{\overline{X}}{PW_{B,2}}\right),$$

$$K = \overline{X}^y$$

B then defines the transcript $T = R||X^*||Y^*||\overline{X}^*||Y^*$ and computes the session key $sk = F(A||B||S||T||K)$. The correctness 3PAKE can be easily verified from the equations:

$$K = \left(\frac{\overline{X}^*}{PW_{B,2}}\right)^y$$

$$= \left(\frac{\overline{X}\cdot PW_{B,2}}{PW_{B,2}}\right)^y$$

$$= g^{xyz}$$

and:

$$K = \left(\frac{\overline{X}^*}{PW_{B,2}}\right)^y$$

$$= \left(\frac{\overline{X}\cdot PW_{B,2}}{PW_{B,2}}\right)^y$$

$$= g^{xyz}$$

As depicted in Fig. 2, the 3PAKE protocol takes two rounds of communications.

Dictionary attack: Assume that G_1 is instantiated as $G_1(.) = g^{\mathcal{H}(\cdot)}$. Then, the 3PAKE protocol described above is vulnerable to the following dictionary attack where by a malicious client B is able to verify all guesses on the password of client A in an offline manner.

Phase 1: The attacker B runs the protocol with the server S while playing dual roles of B itself and the victim A.

Step 1: B selects two random numbers x, y∈Zq and computes X^* and Y^* as:

$$X^* = g^x,$$

$$Y^* = g^y.PW_{B,1}$$

$$= g^y.g^{H1(A, B, pwB)}$$

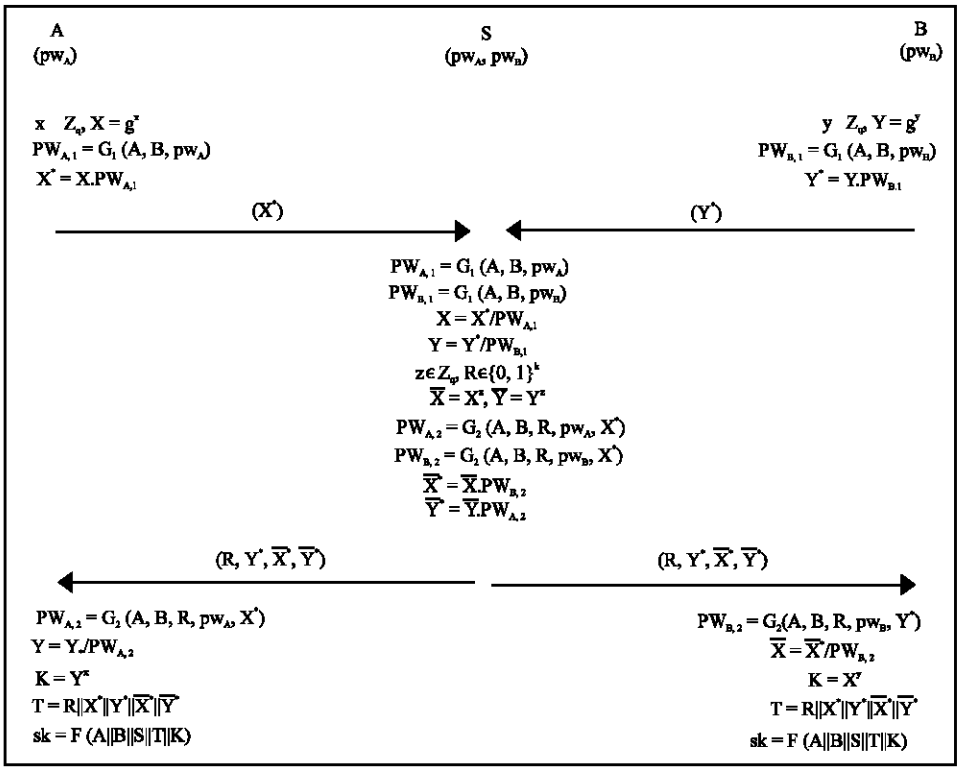

Fig. 2: 3PAKE; Abdalla and Pointcheval (2005)'s three-party PAKE protocol

Then, B sends X^* to as if it is from A while sending Y^* (to S) as its own message.

Step 2: S will send $\langle R, Y^*, \overline{X^*}, \overline{Y^*}\rangle$ and $\langle R, X^*, \overline{X^*}, \overline{Y^*}\rangle$, respectively to A and B in response to X^* and Y^*. B intercepts the message $\langle R, Y^*, \overline{X^*}, \overline{Y^*}\rangle$. Notice here that, X^* is set equal to $g^{(x-H1(A,B,pwA))z}$. $PW_{B,2}$ because S computes it as:

$$\begin{aligned}\overline{X^*} &= \overline{X}.PW_{B,2}\\ &= X^z.PW_{B,2}\\ &= \left(\frac{X^*}{PW_{A.1}}\right)^z.PW_{B,2}\\ &= \left(\frac{g^x}{g^{\mathcal{H}1(A,\ B,\ \rho\omega A)}}\right)^z.PW_{B,2}\\ &= g^{(x-\mathcal{H}_1(A,\ B,\ \rho\omega A))^z}.PW_{B,\ 2}\end{aligned}$$

But $\overline{Y}$ is set equal to g^{yz}. $PW_{A,\ 2}$ as in the usual protocol execution.

Phase 2: Using $\overline{X^*}$ and Y^* obtained in phase 1, B now guesses possible passwords and checks them for correctness.

Step 1: B computes $PW_{B,2} = G_2(A, B, R, pw_B, Y^*)$:

$$\begin{aligned}K &= \left(\frac{\overline{X^*}}{PW_{B,2}}\right)^y\\ &= \left(\frac{g^{x-H_1(A,B,pwA))^z}.PW_{B,2}}{PW_{B,2}}\right)^y\\ &= g^{(x-H_1(A,\ B,\ pwA))\ yz}\end{aligned}$$

Step 2: B makes a guess pw'_A on the password pw_A and computes $PW'_{A,2} = G_2(A, B, R, pw'_A, X^*)$:

$$\begin{aligned}K' &= \left(\frac{\overline{Y^*}}{PW'_{A,2}}\right)^{(x-\mathcal{H}_1(A,B,pw'_A))}\\ &= \left(\frac{g^{yz}.PW_{A,2}}{PW_{A,2}}\right)^{(x-\mathcal{H}_1(A,B,pw'_A))}\end{aligned}$$

Step 3: B verifies the correctness of pw'_A by checking that K is equal to K'. Note that if pw'_A and pw_A are equal, then the equation K = K' ought to be satisfied.

Step 4: B repeats steps 2 and 3 of phase 2 until the correct password is found.

The offline dictionary attack described above can be mounted by any client against any other clients and does not even require the participation of the victim. The existence of the attack demonstrates that similar to the OMDHKE protocol, the 3PAKE protocol cannot guarantee the security of client's passwords when the hash function G_1 is instantiated as $G_1(.) = g^{H(\cdot)}$.

RESULTS AND DISCUSSION

Formal analysis: In this study, we show that our dictionary attacks mounted against OMDHKE and 3PAKE are well captured by Bellare *et al.* (2000) indistinguishability-based security model for analysis of PAKE protocols. We first provide an overview of Bellare *et al.* (2000) security model and then interpret the above-described dictionary attacks in the context of the model.

Security model

Participants: Let, C be the set of all clients registered with a trusted server S. Any registered client $C \in C$ may run a PAKE protocol P with the server S at any point in time to establish a session key. Let, $U = C \cup \{S\}$. A user $U \in U$ may have several instances involved in distinct, possibly concurrent, executions of protocol P. We use $\prod^i_U$ to denote the ith instance of user U. A user instance $\prod^i_U$ is said to accept when it successfully computes its session key sk^i_U in a protocol execution.

Long-term keys: Each client $C \in C$ chooses a password pw_C from a fixed dictionary DIC and shares it with the server S via. a secure channel. Accordingly, S holds all the passwords $\{pw_C | C \in C\}$. Each password pw_C is used as the long-term secret key of C and S.

Partnership: The notion of partnership is a key element in defining the security of the protocol. Two instances are partners if both participate in a protocol execution and establish a (shared) session key. We define the partnership relations between instances using the notions of session identifiers and partner identifiers. A session identifier (sid) is a unique identifier of a protocol session and is defined as a function of the messages transmitted in the protocol session. We use sid^i_U to denote the sid of instance $\prod^i_U$. A partner identifier (pid) is the set of participants of a specific protocol session. Instances should receive as input a pid before they can run the protocol. By pid^i_U, we denote the pid given to instance $\prod^i_U$. We say that any two instances $\prod^i_U$ and $\prod^j_{U'}$ are partners if both $\prod^i_U$ and $\prod^j_{U'}$ have accepted, $sid^i_U = sid^j_{U'}$ and $pid^i_U = pid^j_{U'}$.

Adversary: In the model, the Probabilistic Polynomial-Time (PPT) adversary. A, controls all the communications that take place between users via. a pre-defined set of oracle queries. For example, the adversary can ask participants to reveal session keys and passwords using reveal and corrupt queries as described below.

Execute (pid): This query models passive eavesdropping of a protocol execution. It prompts an honest execution of the protocol between unused instances of the users specified by pid. The transcript of the protocol execution is returned as the output of the query.

Send ($\prod^i_U$, m): This query models active attacks against the protocol. It sends message m to instance $\prod^i_U$ and returns the message that $\prod^i_U$ sends out in response to m. A query of the form send ($\prod^i_U$, start: pid) prompts $\prod^i_U$ to initiate the protocol with pid^i_U = pid.

Reveal ($\prod^i_U$): This query returns the session key sk^i_U. This query captures the notion of known key security. Any instance, $\prod^i_U$, upon receiving such a query and if it has accepted and holds some session key will send this session key back to A. However, the adversary is not allowed to ask this query if it has already made a test query to the instance $\prod^i_U$ or its partner instance (see below for explanation of the test oracle).

Corrupt (U): This query captures not only the notion of forward secrecy but also attacks by malicious clients. The query provides the adversary with U's password pw_U. Notice that, a corrupt query does not result in the release of the session keys, since, the adversary already has the ability to obtain session keys through reveal queries. If U = S (i.e., the server is corrupted), all client's passwords stored by the server will be returned.

Test ($\prod^i_U$): This query is the only oracle query that does not correspond to any of the adversary's abilities. If $\prod^i_U$ has accepted with some session key and is being asked a test ($\prod^i_U$) query, then depending on a randomly chosen bit b, the adversary is given either the actual session key (when b = 1) or a session key drawn randomly from the session key distribution (when b = 0). A test query can be only asked against a fresh instance.

Freshness: Intuitively, a fresh instance is one that holds a session key which should not be known to the adversary and an unfresh instance is the one whose session key (or some information about the key) can be known by trivial means. Formally, we say that an instance $\prod^i_U$ is fresh if none of the following is true.

The adversary queried reveal ($\prod^i_U$) or reveal ($\prod^i_{U'}$) where $\prod^i_{U'}$ is the partner of $\prod^i_U$. The adversary queried corrupt (V) for some $V \in pid^i_U$, before $\prod^i_U$ or its partner $\prod^i_U$ accepts.

Definition of AKE security: The security of protocol P against A is defined in terms of the probability that A can correctly guess the hidden bit b chosen by the test oracle in the following two-stage experiment.

Stage 1: A makes any allowed oracle queries at will as many times as it wishes.

Stage 2: Once A decides that stage 1 is over, it outputs a bit b' as a guess on the hidden bit b chosen by the test oracle. A is said to succeed if b = b'.

Let, Succ be the event that A succeeds in this experiment. Then we define the advantage of A in attacking protocol P as $Adv_P(A) = 2.Pr[Succ]-1$. We say that, a three-party PAKE protocol P is AKE-secure if for all PPT adversaries A making at most q_{send} send queries, $Adv_P(A)$ is only negligibly larger than $c.q_{send}/|DIC|$ where c is a constant.

Breaking AKE security: The AKE security of 3PAKE can be easily broken under the assumption of the bad instantiation $G_1(.) = g^{H(\bullet)}$. We show this by constructing an adversary A who can distinguish with probability 1, random keys from real session keys established by 3PAKE by Bellare *et al.* (2000) Model. The adversary A can be constructed as in Table 1. Since, the dictionary attack against 3PAKE is mounted by a malicious client B, the adversary a begins by asking the corrupt (B) query and thereby, obtaining the password pw_B of client B. With pw_B in hand, a can perfectly simulate the dictionary attack (mounted by B against A) by asking send queries appropriately. After obtaining the password pw_A of client A, A can easily break the AKE security of 3PAKE by performing the Impersonation and test steps of Table 1.

An adversary A who breaks the AKE security of OMDHKE can be similarly constructed as the adversary A against the AKE security of 3PAKE. Due to the similarity, we here only describe the differences between the constructions of A' and A. The most notable difference is that the adversary A' does not have to

Table 1: Construction of an adversary A who breaks the AKE security of 3PAKE

Step	Description
Corruption	A obtains B's password pw_B by querying corrupt (B)
Dictionary attack	Let X* and Y* be defined as in the dictionary attack by B against A. A asks two send queries, send ($\prod^i_S$, ⟨X*⟩) and send ($\prod^j_S$, ⟨Y*⟩) to obtain the password verifiers X and Y from an unused server instance $\prod^j_S$. A then verifies password guesses in the exact same way as B did in its dictionary attack and as a result, finds out the password pw_A of client A
Impersonation	A initiates a new protocol session by querying send ($\prod^i_C$, start: {A, C, S}) where $\prod^i_C$ is an unused instance of an uncorrupted client C. A runs this session as per the protocol specification but simulating by itself all the actions of A (by using pw_A). At the end of the session, the instance $\prod^i_C$ will accept with its session key sk^i_C
Test	Clearly, the instance $\prod^i_C$ is fresh: (1) no reveal query has ever been made for any instance and (2) no corrupt query has been asked for any of pid^i_C = {A, C, S}. Thus, A may ask the query test ($\prod^i_C$). Since, A can compute the same session key as sk^i_C, the probability that A guesses correctly the bit b chosen by the test oracle is 1 and so is the advantage of A in attacking the 3PAKE protocol

perform the corruption step as the attacker in our dictionary attack against OMDHKE need not necessarily be a registered client. The rest of the steps are also different, slightly from the ones for A.

In the dictionary attack step, A' asks the send ($\prod^j_S$ ⟨A, Y⟩) query where Y is as computed by the attacker B to obtain the password verifier ⟨S, Z, $Auth_{SA}$⟩ from a server instance $\prod^j_S$.

In the impersonation step, A' queries send ($\prod^k_S$, start: {A, S}) to prompt a server instance $\prod^k_S$ to initiate a new protocol session with A. A' runs this new session as per the protocol specification except for simulating by itself all the actions of A by using pw_A.

Later, in the test step, A' will ask its test query against the server instance $\prod^k_S$ whose session key is also known to A'.

The remaining parts of the steps proceed as in Table 1. Consequently, A' breaks the AKE security of OMDHKE with advantage 1.

CONCLUSION

Using two popular protocols as case studies, this research has shown that instantiating the hash oracle G: $\{0, 1\}^*$ ⟨G as $G(.) = g^{H(\cdot)}$ where H : $\{0, 1\}^*$⟩ Z_q is likely to render PAKE protocols insecure against an offline dictionary attack. One might think that, the dictionary attacks presented in the study can be readily prevented by instantiating the hash oracle G as $G(.) = g'^{H(\cdot)}$ where g' is a generator of G chosen independently from g. But, this variant of the insecure instantiation also makes the two PAKE protocols vulnerable to offline dictionary attacks similar to the ones we presented. We leave the details of the attack scenarios to readers.

SUGGESTIONS

However, the G-oracle instantiations suggested for the PAK suite; MacKenzie seem to be secure in the sense that they do not cause at least the security vulnerabilities discussed in this research. We suggest that designers of PAKE protocols should clarify which instantiations of the G-oracle can or cannot infringe the security of their protocols. In conclusion, our study, here, highlights the importance of designing a PAKE protocol whose security does not rely on the G-oracle instantiations.

ACKNOWLEDGEMENTS

This research was supported by basic science research program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2017R1A2B1007015). Specially thank to my husband Junghyun Nam for all his supports, discussions and constructive comments during writing of the study.

REFERENCES

Abdalla, M. and D. Pointcheval, 2005. Simple password-based encrypted key exchange protocols. Proceedings of the 2005 RSA Conference on Cryptographers' Track, February 14-18, 2005, Springer, San Francisco, California, USA., ISBN: 978-3-540-24399-1, pp: 191-208.

Abdalla, M. and D. Pointcheval, 2015. Interactive Diffie-Hellman assumptions with applications to password-based authentication. Proceedings of the 9th International Conference on Financial Cryptography and Data Security, February 28-March 3, 2005, Springer, Roseau, Dominica, ISBN:978-3-540-26656-3, pp: 341-356.

Bellare, M. and P. Rogaway, 1993. Random oracles are practical: A paradigm for designing efficient protocols. Proceedings of the 1st ACM Conference on Computer and Communications Security, November 3-5, 1993, ACM, Fairfax, Virginia, USA., ISBN:0-89791-629-8, pp: 62-73.

Bellare, M., D. Pointcheval and P. Rogaway, 2000. Authenticated key exchange secure against dictionary attacks. Proceedings of the 2000 International Conference on the Theory and Applications of Cryptographic Techniques, May 14-18, 2000, Springer, Belgium, ISBN:978-3-540-67517-4, pp: 139.

Bellovin, S.M. and M. Merritt, 1992. Encrypted key exchange: Password-based protocols secure against dictionary attacks. Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy, May 4-6, 1992, IEEE, Oakland, California, pp: 72-84.

Bresson, E., O. Chevassut and D. Pointcheval, 2004. New security results on encrypted key exchange. Proceedings of the 7th International Workshop on Theory and Practice in Public Key Cryptography, March 1-4, 2004, Springer, Singapore, ISBN:978-3-540-21018-4, pp: 145-158.

Canetti, R., D. Dachman-Soled, V. Vaikuntanathan and H. Wee, 2012. Efficient password authenticated key exchange via oblivious transfer. Proceedings of the 15th International Conference on Practice and Theory in Public Key Cryptography, May 21-23, 2012, Springer, Darmstadt, Germany, ISBN:978-3-642-30056-1, pp: 449-466.

Choo, K.K.R., C. Boyd and Y. Hitchcock, 2005. Errors in computational complexity proofs for protocols. Proceedings of the 11th International Conference on Theory and Application of Cryptology and Information Security, December 4-8, 2005, Springer, Chennai, India, ISBN:978-3-540-30684-9, pp: 624-643.

Goyal, V., A. Jain and R. Ostrovsky, 2010. Password-authenticated session-key generation on the internet in the plain model. Proceedings of the 30th Annual Conference on Cryptology, August 15-19, 2010, Springer, Santa Barbara, California, USA., ISBN:978-3-642-14622-0, pp: 277-294.

Katz, J. and V. Vaikuntanathan, 2013. Round-optimal password-based authenticated key exchange. J. Cryptology, 26: 714-743.

Katz, J., R. Ostrovsky and M. Yung, 2009. Efficient and secure authenticated key exchange using weak passwords. J. ACM., 57: 3:1-3:39.

Nam, J., J. Paik, H.K. Kang, U.M. Kim and D. Won, 2009. An off-line dictionary attack on a simple three-party key exchange protocol. IEEE. Commun. Lett., 13: 205-207.

Nam, J., K.K.R. Choo, M. Kim, J. Paik and D. Won, 2013. Dictionary attacks against password-based authenticated three-party key exchange protocols. KSII. Trans. Internet Inf. Syst., 7: 3244-3260.

Szydlo, M., 2006. A note on chosen-basis decisional Diffie-Hellman assumptions. Proceedings of the 10th International Conference Financial Cryptography and Data Security (FC'06), February 27-March 2, 2006, Springer, Anguilla, ISBN:978-3-540-46255-2, pp: 166-170.

Wang, W. and L. Hu, 2006. Efficient and provably secure generic construction of three-party password-based authenticated key exchange protocols. Proceedings of the 7th International Conference on Cryptology in India, December 11-13, 2006, Springer, Kolkata, India, ISBN:978-3-540-49767-7, pp: 118-132.

Xiong, H., Y. Chen, Z. Guan and Z. Chen, 2013. Finding and fixing vulnerabilities in several three-party password authenticated key exchange protocols without server public keys. Inf. Sci., 235: 329-340.

Yi, X., S. Ling and H. Wang, 2013. Efficient two-server password-only authenticated key exchange. IEEE Trans. Parallel Distrib. Syst., 24: 1773-1782.