Research Article

Password-Only Authenticated Three-Party Key Exchange with Provable Security in the Standard Model

Junghyun Nam, 1 Kim-Kwang Raymond Choo, 2 Junghwan Kim, 1 Hyun-Kyu Kang, 1 Jinsoo Kim, 1 Juryon Paik, 3 and Dongho Won 3

1 Department of Computer Engineering, Konkuk University, 268 Chungwondaero, Chungju, Chungcheongbukdo 380-701, Republic of Korea
2 Information Assurance Research Group, Advanced Computing Research Centre, University of South Australia, Mawson Lakes, SA 5095, Australia
3 Department of Computer Engineering, Sungkyunkwan University, 2066 Seoburo, Suwon, Gyeonggido 440-746, Republic of Korea

Correspondence should be addressed to Dongho Won; dhwon@security.re.kr

Received 23 January 2014; Accepted 27 February 2014; Published 14 April 2014

Academic Editors: T. Cao, M. Ivanovic, and F. Yu

Copyright © 2014 Junghyun Nam et al. 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.

Protocols for password-only authenticated key exchange (PAKE) in the three-party setting allow two clients registered with the same authentication server to derive a common secret key from their individual password shared with the server. Existing three-party PAKE protocols were proven secure under the assumption of the existence of random oracles or in a model that does not consider insider attacks. Therefore, these protocols may turn out to be insecure when the random oracle is instantiated with a particular hash function or an insider attack is mounted against the partner client. The contribution of this paper is to present the first three-party PAKE protocol whose security is proven without any idealized assumptions in a model that captures insider attacks. The proof model we use is a variant of the indistinguishability-based model of Bellare, Pointcheval, and Rogaway (2000), which is one of the most widely accepted models for security analysis of password-based key exchange protocols. We demonstrated that our protocol achieves not only the typical indistinguishability-based security of session keys but also the password security against undetectable online dictionary attacks.

1. Introduction

Authenticated key exchange is one of the most fundamental problems in cryptography and network security. In 1992, Bellovin and Merritt [1] introduced encrypted key exchange (or EKE) protocols, which allow

arbitrary two parties, who share only a low-entropy password, to establish a common high-entropy secret key (called a session key) over an insecure public network.

Since the work of Bellovin and Merritt [1], password-only authenticated key exchange (PAKE) protocols have attracted much greater attention mainly due to the persistent popularity of passwords as a practical (and cheap) authentication method [2]. Since the publication of the first EKE protocol (with only heuristic security arguments), many provably secure PAKE protocols have been published. Recent examples include the protocol of Katz and Vaikuntanathan [3], which enjoys both round optimality and provable security in the standard model (i.e., without random oracles and ideal ciphers).

A major challenge in designing PAKE protocols is to protect passwords from a dictionary attack, in which an adversary enumerates all possible passwords while testing each one against known password verifiers in order to determine the correct one. The design of two-party PAKE protocols secure against dictionary attacks has been extensively studied over the past two decades and is now fairly well understood. However, three-party PAKE protocols have received far less attention and preventing dictionary attacks is more challenging in the three-party setting. Unlike the two-party setting that assumes the same password is shared
between the two parties, the three-party setting assumes that the two parties (commonly known as clients) wishing to establish a session key do not share the same password but hold their individual password shared only with a trusted server. This implies that in the three-party setting, a malicious client can attempt to mount an insider dictionary attack against its partner client. Indeed, many published three-party PAKE protocols were subsequently found to be vulnerable to an insider online/offline dictionary attack (e.g., [4–10]).

It is widely regarded that the design of key exchange protocols (including PAKE protocols) is notoriously hard, and conducting security analysis for such protocols is time-consuming and error-prone [11–13]. The many flaws discovered in published protocols have promoted the use of formal models and rigorous security proofs [14–16]. In the provable security paradigm for protocol analysis, a deductive reasoning process is adopted whereby emphasis is placed on a proven reduction from the problem of breaking the protocol to another problem believed to be computationally hard. A complete mathematical proof with respect to cryptographic definitions provides a strong assurance that a protocol is behaving as desired. It is now standard practice for protocol designers to provide security proofs in a well-defined formal model in order to assure protocol implementers about the security properties of protocols.

Over the past decade, we have seen a number of PAKE protocols proposed in the three-party setting [4–8, 17–29]. Many of these published protocols either did not have a proof of security [5, 6, 17, 22–25] or were subsequently found to be flawed [4–10, 12, 23, 24, 27, 30–36]. There are only a handful of provably secure three-party PAKE protocols [4, 7, 8, 21] whose claimed security properties have not been invalidated. However, there are limitations in the security proof of these protocols. For example, the protocols of [7, 8] are proven secure in the random oracle model. Although a proof of security in the random oracle model is definitely better than having no proof, it may not guarantee security in the real world (currently an open question). The protocols of [4, 21] are proven secure in a restricted model where the adversary is not allowed to corrupt protocol participants. Note that a protocol proven secure in such a restricted model cannot guarantee its security against attacks by malicious clients including insider online/offline dictionary attacks. (Readers who are unfamiliar with formal security models are referred to Section 2.1.) Although Yang and Cao [37] proposed a new three-party key exchange protocol that was proven secure in the standard model, the protocol is based on the ElGamal encryption scheme and thus requires a server’s public key as well as clients’ passwords to be preestablished before the protocol is ever executed. We refer the readers to [33, 38–44] for other recently published protocols designed to work in a “hybrid” setting where a cryptographic key is required in addition to passwords.

To the best of our knowledge, there is no published three-party PAKE protocol whose security is proven secure in the standard model that allows an adversary to corrupt protocol participants. In this work, we present the first three-party PAKE protocol that achieves provable security in the standard model against an active adversary with the corruption capability. We prove the security of session keys for our protocol in the widely accepted indistinguishability-based model of Bellare, Pointcheval, and Rogaway [14]—this model is, probably, one of the most popular proof models in the provable security paradigm for key exchange protocols. However, the indistinguishability-based security of session keys proven in the Bellare-Pointcheval-Rogaway model (and several other standard models) does not imply the security of passwords against undetectable online dictionary attacks, in which each guess on the password is checked undetectably via an online transaction with the server (see Section 2.3 for more details). We address this problem by providing a separate proof of security for the protocol against undetectable online dictionary (UDOD) attacks. This second proof is compact and elegant and does not rely upon idealized assumptions about the cryptographic primitives. Table 1 compares our protocol against other provably secure three-party PAKE protocols in terms of security proofs.

The remainder of this paper is structured as follows. Section 2 describes a formal proof model along with the associated definitions of security. Section 3 presents our proposed three-party PAKE protocol. In Section 4, we prove that the proposed protocol achieves not only the typical indistinguishability-based security of session keys but also the password security against undetectable online dictionary attacks. We conclude the paper in Section 5.

2. Formal Setting

In this section, we

1. first describe a security model adapted from the Bellare-Pointcheval-Rogaway 2000 model [14],

2. define a typical indistinguishability-based security of session keys, which we call the SK security,

3. provide a simple and intuitive definition of security against undetectable online dictionary attacks.

2.1. The Security Model

Protocol Participants. Let \( \mathcal{C} \) be the set of all clients registered with the trusted authentication server \( S \). Clients \( C, C' \in \mathcal{C} \) who are both registered with \( S \) may run a three-party PAKE protocol \( P \) at any point in time to establish a session key. Let \( \mathcal{U} = \mathcal{C} \cup \{S\} \). A party \( U \in \mathcal{U} \) may have several instances involved in distinct, possibly concurrent, executions of protocol \( P \). We use \( \Pi^j_U \) to denote the \( j \)th instance of party \( U \). A client instance \( \Pi^j_C \) is said to accept when it successfully computes its session key \( sk^j_C \) in a protocol execution.

Long-Term Keys. Each client \( C \in \mathcal{C} \) chooses a password \( pw_C \) from a fixed dictionary \( PW \) and shares it with the server \( S \) via a secure channel. Accordingly, \( S \) holds all the passwords \( \{pw_C | C \in \mathcal{C}\} \). Each password \( pw_C \) is used as the long-term secret key of \( C \) and \( S \).
Table 1: Security proof comparison.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Idealized assumption</th>
<th>Adversary capability</th>
<th>Resistance to UDOD attacks†</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our protocol</td>
<td>None</td>
<td>Not restricted</td>
<td>Proven</td>
</tr>
<tr>
<td>NGPAKE [4]</td>
<td>None</td>
<td>Not restricted</td>
<td>Not proven</td>
</tr>
<tr>
<td>Lin and Hwang [7]</td>
<td>Random oracles</td>
<td>Not restricted</td>
<td>Not proven</td>
</tr>
<tr>
<td>Wu et al. [8]</td>
<td>Random oracles</td>
<td>Not restricted</td>
<td>Not proven</td>
</tr>
</tbody>
</table>

† Resistance to undetectable online dictionary attacks.

**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 (see [45] on the role and the possible construct of session and partner identifiers as a form of partnering mechanism that enables the right session key to be identified in concurrent protocol executions.). 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 $sid_i$ to denote the sid of instance $\Pi^i_U$. 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 $pid_i$, we denote the pid given to instance $\Pi^i_U$. Notice that $pid_i$ consists of three participants: server $S$, client $C$, and another client $C'$. With whom $\Pi^i_C$ believes it runs the protocol. We say that any two instances $\Pi^i_C$ and $\Pi^{i'}_C$ are partners if (1) both $\Pi^i_C$ and $\Pi^{i'}_C$ have accepted, (2) $sid^i_C = sid^{i'}_C$, and (3) $pid^i_C = vid^{i'}_C$.

**Adversary.** In the model, the probabilistic polynomial-time (pPT) adversary, $a'$, controls all the communications that take place between parties via a predefined 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.

(i) **Execute($\Pi^i_C$, $\Pi^{i'}_C$, $\Pi^{i''}_C$).** This query models passive eavesdropping of a protocol execution. It prompts an honest execution of the protocol between the instances $\Pi^i_C$, $\Pi^{i'}_C$, and $\Pi^{i''}_C$. The transcript of the protocol execution is returned as the output of the query.

(ii) **Send($\Pi^i_U$, $m$).** This query models active attacks against the protocol. It sends message $m$ to instance $\Pi^i_U$ and returns the message that $\Pi^i_U$ sends out in response to $m$. A query of the form $Send(\Pi^i_U, \text{start}:(C, C', S))$ prompts $\Pi^i_U$ to initiate the protocol with $pid^i_C = (C, C', S)$.

(iii) **Reveal($\Pi^i_C$).** This query returns the session key $sk^i_C$. This query captures the notion of known key security (and it is often reasonable to assume that the adversary will be able to obtain session keys from any session different from the one under attack). Any client, $\Pi^i_C$, 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 $\Pi^i_C$ or its partner instance (see below for explanation of the $Test$ oracle).

(iv) **Corrupt($U$).** This query captures not only the notion of forward secrecy but also unknown key share attacks and insider attacks. 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 clients’ passwords stored by the server will be returned.

(v) **Test($\Pi^i_C$).** This query is the only oracle query that does not correspond to any of the adversary’s abilities. If $\Pi^i_C$ has accepted with some session key and is being asked a $Test(\Pi^i_C)$ 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 $b = 0$). The adversary can access the $Test$ oracle as many times as necessary. All the queries to the oracle are answered using the same value of the hidden bit $b$. Namely, the keys returned by the $Test$ oracle are either all real or all random. But, we require that for each different set of partners, the adversary should access the $Test$ oracle only once.

We represent the number of queries used by an adversary as an ordered sequence of five nonnegative integers, $Q = (q_{\text{co}}, q_{\text{se}}, q_{\text{co}}, q_{\text{co}}, q_{\text{co}})$, where the five elements refer to the numbers of queries that the adversary made, respectively, to its $Execute$, $Send$, $Reveal$, $Corrupt$, and $Test$ oracles. We call this usage of queries by an adversary the query complexity of the adversary.

2.2. **Session Key (SK) Security.** We now define the basic security, called the SK security, of a 3-party PAKE protocol. As usual, we define the SK security via the notion of freshness. Intuitively, a fresh instance is one that holds a session key which should not be known to the adversary $a'$, and an unfresh instance is one whose session key (or some information about the key) can be known by trivial means. The formal definition of freshness is explained in Definition 1.
Definition 1. An instance $\Pi_i^C$ is fresh if none of the following occurs: (1) $A$ queries $\text{Reveal}(\Pi_i^C)$ or $\text{Reveal}(\Pi_{i'}^C)$, where $\Pi_{i'}^C$ is the partner of $\Pi_i^C$ and (2) $A$ queries $\text{Corrupt}(U)$, for some $U \in \text{pid}_C$, before $\Pi_i^C$ or its partner $\Pi_{i'}^C$ accepts.

The SK security of a 3-party PAKE protocol $P$ is defined in the context of the following two-stage experiment.

Stage 1. $A$ makes any oracle queries at will except that:

(i) $A$ is not allowed to ask the $\text{Test}(\Pi_i^C)$ query if the instance $\Pi_i^C$ is unfresh.

(ii) $A$ is not allowed to ask the $\text{Reveal}(\Pi_i^C)$ query if it has already made a $\text{Test}$ query to $\Pi_i^C$ or $\Pi_{i'}^C$, where $\Pi_{i'}^C$ is the partner of $\Pi_i^C$.

Stage 2. Once $A$ decides that Phase 1 is over, it outputs a bit $b'$ as a guess on the hidden bit $b$ chosen by the $\text{Test}$ oracle. $A$ is said to succeed if $b = b'$.

Let $\text{Succ}$ be the event that $A$ succeeds in this experiment. Then we define the advantage of $A$ in breaking the SK security of protocol $P$ as

$$
\text{Adv}_P^\text{sk}(A) = 2 \cdot \Pr[\text{Succ}] - 1,
$$

and

$$
\text{Adv}_P^c(t, Q) = \max_A \left\{ \text{Adv}_P^k(A) \right\},
$$

where the maximum is over all PPT adversaries $A$ with time complexity at most $t$ and query complexity at most $Q$.

Definition 2. A 3-party PAKE protocol $P$ is SK-secure if, for any PPT adversary $A$ asking at most $q_{\text{sk}}$ Send queries, $\text{Adv}_P^\text{sk}(A)$ is only negligibly larger than $c \cdot q_{\text{sk}}/|\text{PW}|$, where $c$ is a very small constant (usually around 2 or 4) when compared with $|\text{PW}|$.

2.3. Modelling Undetectable Online Dictionary Attacks. The SK security does not imply security against undetectable online dictionary attacks. In other words, a 3-party PAKE protocol that is not secure against an undetectable online dictionary attack may be rendered SK-secure. To see this, suppose that a 3-party PAKE protocol $P$ is susceptible to undetectable online dictionary attacks whereby an attacker $A$ can find out the password of any registered client $B$. Then, we can construct an adversary $A$ who attacks protocol $P$ with advantage 1 as follows.

Corruption. If $A$ is a registered client, $A$ queries $\text{Corrupt}(A)$ to obtain the password $\text{pw}_A$. Otherwise, $A$ skips this step.

Undetectable Online Dictionary Attacks. Next, $A$ runs the protocol $P$ in the same way as $A$ conducts its undetectable online dictionary attacks against client $B$. Note that $A$ can perfectly simulate $A$’s attack by using the disclosed password $\text{pw}_A$ and by asking oracle queries appropriately. At the end of this step, $A$ will obtain the password $\text{pw}_B$ as a result of the attacks.

Impersonation. Then $A$ will initiate a new protocol session by querying $\text{Send}(\Pi_i^C)$, start: $(B, C, S)$, where $\Pi_i^C$ 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 $B$ (by using $\text{pw}_B$). At the end of the session, the instance $\Pi_i^C$ will accept with its session key $\text{sk}_C^i$. $A$ may ask $\text{Test}(\Pi_i^C)$ or its partner (which does not exist) and (2) no $\text{Corrupt}$ query has been made against any of $B, C$, and $S$. Thus, $A$ may ask the $\text{Test}(\Pi_i^C)$ query. Since $A$ can compute the session key $\text{sk}_C^i$ by itself, it follows that $\Pr_{P,A}[\text{Succ}] = 1$ and thus $\text{Adv}_P^\text{sk}(A) = 1$.

Since verifying the correctness of a password guess may require more than one $\text{Send}$ query to be asked, $A$ may have to ask $\text{Send}$ queries as many times as $d \cdot |\text{PW}|$, for some integer $d \geq 1$, to correctly determine the password $\text{pw}_A$. Then, even if $\text{Adv}_P^\text{sk}(A) = 1$, the following holds for some $c \geq 1$:

$$
\text{Adv}_P^c(t, Q) \leq \frac{cd |\text{PW}|}{|\text{PW}|},
$$

and the protocol $P$ is rendered SK-secure by Definition 2.

This result is not surprising since we call a protocol SK-secure if mounting an online dictionary attack by asking $\text{Send}$ queries is the best an adversary can do. However, we want to be able to distinguish undetectable online dictionary attacks from detectable online dictionary attacks, and ensure that the best an adversary can do is to mount a detectable online dictionary attack. The following new definitions together provide a simple and intuitive way of capturing security against undetectable online dictionary attacks.

Definition 3. The $\text{Send}(\Pi_i^C, m)$ query models an online dictionary attack if both the following are true at the time of the termination of instance $\Pi_i^C$: (1) $m$ was not output by a previous $\text{Send}$ query asked to an instance of $C$ by which, $\Pi_i^C$ believes, $m$ was sent and (2) the adversary $A$ queried neither $\text{Corrupt}(S)$ nor $\text{Corrupt}(C)$.

In Definition 3, the first condition implies that a straightforward delivery of a message between instances is not considered as an online dictionary attack while the second condition implies that, when $C'$ is the (assumed) peer of client $C$, the adversary $A$ can corrupt the peer client $C'$ to mount an (insider) online dictionary attack. Note that our definition of an online dictionary attack does not impose any restriction on asking $\text{Reveal}$ queries.

Consider the two-stage experiment described in the previous section. Let $\text{Undet}$ be the event that in the experiment, a server instance terminates normally when an online dictionary attack was mounted against the instance. We say that the adversary $A$ succeeds in mounting an undetectable online dictionary attack if the event $\text{Undet}$ occurs.
3. Our Proposed Protocol

As we have earlier claimed, our proposed protocol presented in this section is the first three-party PAKE protocol proven secure in the standard model against an active adversary who has the corruption ability.

3.1. Preliminaries. We begin by reviewing some cryptographic primitives which underlie the security of our protocol.

Decisional Diffie-Hellman (DDH) Assumption. Let $G$ be a cyclic (multiplicative) group of prime order $q$. Since the order of $G$ is prime, all the elements of $G$, except 1, are generators of $G$. Let $g$ be a random fixed generator of $G$ and let $x, y, z$ be randomly chosen elements in $\mathbb{Z}_q$ where $z \neq \pm 1$. Informally stated, the DDH problem for $G$ is to distinguish the distributions of $(g^x, g^y, g^z)$ and $(g^x, g^y, g^z)$, and the DDH assumption is said to hold in $G$ if it is computationally infeasible to solve the DDH problem for $G$. More formally, we define the advantage of $\mathcal{D}$ in solving the DDH problem for $G$ as $\text{Adv}_{G}^{\text{DDH}}(\mathcal{D}) = |\Pr[\mathcal{D}(g, g^x, g^y, g^z) = 1] - \Pr[\mathcal{D}(g, g^x, g^y, g^z) = 1]|$. We say that the DDH assumption holds in $G$ if $\text{Adv}_{G}^{\text{DDH}}$ is negligible for all ppt algorithms $\mathcal{D}$. We denote by $\text{Adv}_{G}^{\text{DDH}}(t)$ the maximum value of $\text{Adv}_{G}^{\text{DDH}}$ over all algorithms $\mathcal{D}$ running in time at most $t$.

Message Authentication Codes. Let $\Sigma = \{\text{Gen}, \text{Mac}, \text{Ver}\}$ be a message authentication code (MAC) scheme. The key generation algorithm $\text{Gen}$ takes as input a security parameter $1^t$ and outputs a key $k$ chosen uniformly at random from $\{0, 1\}^t$. The MAC generation algorithm $\text{Mac}$ takes as input a key $k$ and a message $m$ and outputs a MAC (also known as a tag) $\sigma$. The MAC verification algorithm $\text{Ver}$ takes as input a key $k$, a message $m$, and a MAC $\sigma$ and outputs 1 if $\sigma$ is valid for $m$ under $k$ or outputs 0 if $\sigma$ is invalid. Let $\text{Adv}_{\Sigma}^{\text{Mac}}(\mathcal{A})$ be the probability that an adversary $\mathcal{A}$ succeeds in breaking the existential unforgeability of $\Sigma$ under adaptive chosen message attacks. We say that the MAC scheme $\Sigma$ is secure if $\text{Adv}_{\Sigma}^{\text{Mac}}(\mathcal{A})$ is negligible for every ppt adversary $\mathcal{A}$. We use $\text{Adv}_{\Sigma}^{\text{Mac}}(t, q_{\text{mac}}, q_{\text{ver}})$ to denote the maximum value of $\text{Adv}_{\Sigma}^{\text{Mac}}(\mathcal{A})$ over all ppt adversaries $\mathcal{A}$ running in time at most $t$ and asking at most $q_{\text{mac}}$ and $q_{\text{ver}}$ queries to its MAC generation and verification oracles, respectively.

Two-Party PAKE Protocols. Let 2PAKE be a two-party PAKE protocol that outputs session keys distributed in $\{0, 1\}^t$. We assume that 2PAKE is SK-secure against an adversary who is given access to all the oracles: Send, Execute, Reveal, Corrupt, and Test. Let $\text{Adv}_{2\text{PAKE}}^{\text{sk}}(\mathcal{A})$ be the advantage of an adversary $\mathcal{A}$ in breaking the SK security of 2PAKE. We require that, for all ppt adversaries $\mathcal{A}$ making at most $q_{\text{sk}}$ Send queries, $\text{Adv}_{2\text{PAKE}}^{\text{sk}}(\mathcal{A})$ is only negligibly larger than $q_{\text{sk}}/|\mathcal{P}|$. We denote by $\text{Adv}_{2\text{PAKE}}^{\text{sk}}(t, Q)$ the maximum value of $\text{Adv}_{2\text{PAKE}}^{\text{sk}}(\mathcal{A})$ over all ppt adversaries $\mathcal{A}$ with time complexity at most $t$ and query complexity at most $Q$.

3.2. Protocol Description. Let $A$ and $B$ be two clients who wish to establish a session key, and let $S$ be a trusted server with which $A$ and $B$ have secretly shared their respective passwords $pw_A$ and $pw_B$. Our protocol proceeds as follows.

Step 1. $A$ and $S$ establish a shared secret key $k_{AS}$ by running the two-party protocol 2PAKE. Likewise, $B$ and $S$ establish a shared secret key $k_{BS}$.

Step 2. $A$ (resp., $B$ and $S$) selects a nonce $n_A$ (resp., $n_B$ and $n_S$) at random from $\mathbb{Z}_q$ and sends $A \parallel n_A$ (resp., $B \parallel n_B$ and $S \parallel n_S$) to the other two parties. All the parties ($A$, $B$, and $S$) define their session identifiers as $s_{id_A} = s_{id_B} = s_{id_S} = A \parallel n_A \parallel B \parallel n_B \parallel S \parallel n_S$.

Step 3. $A$ chooses a random $x \in \mathbb{Z}_q$, computes $X = g^x$ and $\sigma_{AS} = \text{Mac}_{k_{AS}}(A \parallel X \parallel s_{id_A})$, and sends $A \parallel X \parallel \sigma_{AS}$ to $S$. Meanwhile, $B$ chooses a random $y \in \mathbb{Z}_q$, computes $Y = g^y$ and $\sigma_{BS} = \text{Mac}_{k_{BS}}(B \parallel Y \parallel s_{id_B})$, and sends $B \parallel Y \parallel \sigma_{BS}$ to $S$.

Step 4. $S$ checks that $\text{Ver}_{k_{AS}}(A \parallel X \parallel s_{id_A}, \sigma_{AS}) = 1$ and $\text{Ver}_{k_{BS}}(B \parallel Y \parallel s_{id_B}, \sigma_{BS}) = 1$. If either of these is untrue, $S$ aborts the protocol. Otherwise, $S$ computes $\sigma_{SA} = \text{Mac}_{k_{SA}}(S \parallel Y \parallel s_{id_S})$ and $\sigma_{SB} = \text{Mac}_{k_{SB}}(S \parallel X \parallel s_{id_S})$, and sends $S \parallel Y \parallel \sigma_{SA}$ and $S \parallel X \parallel \sigma_{SB}$ to $A$ and $B$, respectively.

Step 5. $A$ computes the session key $sk = X^y$ if $\text{Ver}_{k_{SA}}(S \parallel Y \parallel s_{id_S}, \sigma_{SA}) = 1$, while $B$ computes the session key $sk = X^y$ if $\text{Ver}_{k_{SB}}(S \parallel X \parallel s_{id_B}, \sigma_{SB}) = 1$. $A$ and $B$ abort the protocol if their verification fails.

The operation of this protocol is illustrated in Figure 1. Steps 1 and 2 of the protocol are independent and can be performed in parallel. The session-key computation in the protocol is the same as in the Diffie-Hellman key exchange protocol (i.e., $sk = g^{xy}$). Hence, it is straightforward to verify the correctness of the protocol.

4. Security Proofs

In this section we prove that our three-party PAKE protocol is SK-secure and is resistant to undetectable online dictionary attacks. The proofs of both properties rely on neither random oracles nor ideal ciphers. Therefore, if 2PAKE is instantiated with a protocol proven secure in the standard model (e.g., [3, 49]), our three-party PAKE protocol would also be provably secure in the standard model. Hereafter, we denote our protocol by 3PAKEsm (“sm” for “standard model”).
A (pw_A) S (pw_A, pw_B) B (pw_B)

\[ \xrightarrow{2PAKE (\rightarrow k_{AS})} \xleftarrow{2PAKE (\rightarrow k_{BS})} \]

Exchange nonces: n_A, n_B, n_S

\[ x \in \mathbb{Z}_q, x = g^t \]

\[ y \in \mathbb{Z}_q, y = g^s \]

\[ \sigma_{AS} = \text{Mac}_{k_{AS}}(A[X]\|\text{sid}_A), \quad \sigma_{RS} = \text{Mac}_{k_{RS}}(B[Y]\|\text{sid}_B) \]

\[ A[X]\|\sigma_{AS} \quad B[Y]\|\sigma_{RS} \]

\[ \text{Set sid}_A = \text{sid}_B = \text{sid}_S = A \parallel \ldots \parallel A \]

\[ \text{2PAKE usestheoutputofthis Test query as theMACkeygeneratingthemessagesofSteps 2–4 ofprotocol 3PAKEsm.} \]

\[ \exists \mathcal{A} \text{ 's advantage between } 3PAKEsm \text{ isboundedby} \]

\[ \text{Adv}_{2PAKE}^{\mathcal{A}}(t) \leq 2 \cdot \text{Adv}_{2PAKE}^{\mathcal{A}}(t', Q) + \frac{(q_{se} + q_{cx})^2}{|G|} + 2 \cdot q_{se} \cdot \text{Adv}_{\Sigma}^{\mathcal{A}, \text{rand}}(t', 4, 4) + 4 \cdot \text{Adv}_{\Sigma}^{\mathcal{A}, \text{rand}}(t'), \]

where \( Q = (q_{ex}, q_{se}, q_{co}, q_{cx}) \). We prove \text{Theorem 5} by introducing a sequence of experiments \text{Expr}_0, \ldots, \text{Expr}_4 and bounding the difference in \( \mathcal{A} \) 's advantage between two consecutive experiments. \text{Expr}_0 is the original experiment (described in Section 2.2) in which \( \mathcal{A} \) attacks the actual protocol, and \text{Expr}_4 is the experiment in which the advantage of \( \mathcal{A} \) is 0. Let Succ_0 be the event that \( \mathcal{A} \) correctly guesses the hidden bit \( b \) (chosen by the Test oracle) in experiment \text{Expr}_4. By definition, we get \[ \text{Adv}_{3PAKE}^{\mathcal{A}}(\mathcal{A}) = 2 \cdot \text{Pr}[\text{Succ}_0] - 1. \]

Before providing details of the proof, we first define the notion of an uncorrupted instance.

\textbf{Definition 6.} We say an instance \( \Pi_{ij}^\mathcal{A} \) is clean if no one in \( \text{pid}_j \) has been asked a \text{Corrupt} query. Otherwise, we say it is unclean.

\textbf{Experiment \text{Expr}_4.} We modify the experiment so that each different MAC key is chosen uniformly at random from \( \{0, 1\}^e \) for all clean instances. The difference in \( \mathcal{A} \) 's success probability between \text{Expr}_0 and \text{Expr}_4 is bounded by

\[ \text{Claim 1.} \]

\[ |\text{Pr}[\text{Succ}_0] - \text{Pr}[\text{Succ}_4]| \leq \text{Adv}_{\text{2PAKE}}^{\mathcal{A}}(t', Q'). \quad (4) \]

\textbf{Proof.} Assume that the advantage of \( \mathcal{A} \) in attacking protocol \( 3PAKEsm \) is different between two experiments \text{Expr}_0 and \text{Expr}_4. Then we prove the claim by constructing, from \( \mathcal{A} \), an adversary \( \mathcal{A}_{\text{2PAKE}} \) attacking protocol \( 2PAKE \) with time complexity \( t' \) and query complexity \( Q' \).

\( \mathcal{A}_{\text{2PAKE}} \) begins by choosing a bit \( b \) uniformly at random. \( \mathcal{A}_{\text{2PAKE}} \) then invokes \( \mathcal{A} \) as a subroutine and answers the oracle queries of \( \mathcal{A} \) on its own as follows.

\textbf{Execute Queries.} \( \mathcal{A}_{\text{2PAKE}} \) answers \text{Execute queries of} \( \mathcal{A} \) by making \text{Execute} and \text{Test} queries to its own oracles. Specifically, \( \mathcal{A}_{\text{2PAKE}} \) handles each \text{Execute}(\Pi_{ij}^\mathcal{A}, \Pi_{ij}^\mathcal{B}, \Pi_{ij}^\mathcal{S}) \) query as follows.

(i) If anyone in \{A, B, S\} has been corrupted, then \( \mathcal{A}_{\text{2PAKE}} \) answers the \text{Execute} query as in experiment \text{Expr}_0.

(ii) Otherwise, \( \mathcal{A}_{\text{2PAKE}} \) first makes two queries \text{Execute}(\Pi_{ij}^\mathcal{A}, \Pi_{ij}^\mathcal{B}) \) and \text{Execute}(\Pi_{ij}^\mathcal{A}, \Pi_{ij}^\mathcal{S}) \). Let \( T_{\text{2PAKE}} \) and \( T_{\text{2PAKE}}' \) be two transcripts returned in response to the \text{Execute} queries. Next, \( \mathcal{A}_{\text{2PAKE}} \) makes the queries \text{Test}(\Pi_{ij}^\mathcal{A}) \) and \text{Test}(\Pi_{ij}^\mathcal{S}) \) and receives in return two keys \( k_{AS} \) and \( k_{RS} \) (either real or random). \( \mathcal{A}_{\text{2PAKE}} \) then generates the messages of Steps 2–4 of protocol \( 3PAKEsm \), using \( k_{AS} \) and \( k_{RS} \) as the MAC keys. Finally, \( \mathcal{A}_{\text{2PAKE}} \) returns these messages prepended by \( T_{\text{2PAKE}} \) and \( T_{\text{2PAKE}}' \).

\textbf{Send Queries.} At a high level, the simulation of the \text{Send} oracle is similar to that of the \text{Execute} oracle. Specifically, \( \mathcal{A}_{\text{2PAKE}} \) handles each \text{Send}(\Pi_{ij}^\mathcal{U}, m) \) query as follows.

(i) If the instance \( \Pi_{ij}^\mathcal{U} \) is clean or the message \( m \) belongs to Step 2 or later steps, then \( \mathcal{A}_{\text{2PAKE}} \) answers the query as in experiment \text{Expr}_0.

(ii) Otherwise, \( \mathcal{A}_{\text{2PAKE}} \) answers it by making the same query to its own \text{Send} oracle. If the query causes \( \Pi_{ij}^\mathcal{U} \) to accept, then \( \mathcal{A}_{\text{2PAKE}} \) also makes a \text{Test}(\Pi_{ij}^\mathcal{U}) \) query (if it had not previously asked a \text{Test} query to the partner of \( \Pi_{ij}^\mathcal{U} \)). As in the simulation of the \text{Execute} oracle, \( \mathcal{A}_{\text{2PAKE}} \) uses the output of this \text{Test} query as the MAC key in generating the messages of Steps 2–4 of protocol \( 3PAKEsm \).
Reveal Queries. These queries are answered in the obvious way. Namely, $s_{2PAKE}$ responds to the query $\text{Reveal}(\Pi'_0)$ by returning the session key $sk'_0$.

Corrupt Queries. When $a$ queries $\text{Corrupt}(U)$, $s_{2PAKE}$ makes the same query to its own $\text{Corrupt}$ oracle and simply forwards the output to $a$.

Test Queries. $s_{2PAKE}$ answers these queries according to the bit $b$ chosen at the beginning of the simulation. That is, $s_{2PAKE}$ returns real session keys, which it has computed on its own, if $b = 1$, and otherwise returns random keys chosen uniformly at random from $G$.

Now at some point in time, when $a$ terminates and outputs its guess $b'$, $s_{2PAKE}$ outputs 1 if $b = b'$ and outputs 0 otherwise.

From the simulation above, it is easy to see that $s_{2PAKE}$ has at most time complexity $t'$ and query complexity $Q'$. The advantage of $a_{2PAKE}$ in attacking protocol $2PAKE$ is immediate if we notice the following.

(i) The probability that $s_{2PAKE}$ outputs 1 when its Test oracle returns real session keys is equal to $\Pr[\text{Succ}_1]$, the probability that $a$ correctly guesses the bit $b$ in experiment $\text{Expr}_0$.

(ii) The probability that $s_{2PAKE}$ outputs 1 when its Test oracle returns random keys is equal to $\Pr[\text{Succ}_1]$, the probability that $a$ correctly guesses the bit $b$ in experiment $\text{Expr}_1$.

This means that $\text{Adv}^{sk}_{2PAKE}(a_{2PAKE}) = |\Pr[\text{Succ}_1] - \Pr[\text{Succ}_0]|$.

Claim 1 then follows.

Experiment $\text{Expr}_2$. Let $\text{Repeat}$ be the event that a nonce selected by an instance of a party is selected again by another instance of the same party. The experiment $\text{Expr}_2$ is aborted, and the adversary does not succeed, if the event $\text{Repeat}$ occurs. This is the only difference between $\text{Expr}_1$ and $\text{Expr}_2$.

By a straightforward calculation, we get the following.

Claim 2.

$$|\Pr[\text{Succ}_2] - \Pr[\text{Succ}_1]| \leq \frac{(q_{se} + q_{ex})^2}{2|G|}. \quad (5)$$

Experiment $\text{Expr}_3$. Let $\text{Forge}$ be the event that the adversary $a$ makes a $\text{Send}$ query of the form $\text{Send}(\Pi'_i, V \parallel \star \parallel \sigma)$ before querying $\text{Corrupt}(W)$, for some $W \in \text{pid}_i$, such that (1) $\sigma$ is a valid tag on $V \parallel \star \parallel \text{sid}_i$ and (2) no oracle had not previously generated a tag on $V \parallel \star \parallel \text{sid}_i$. If $\text{Forge}$ occurs, this experiment is aborted and the adversary does not succeed. Then we have the following.

Claim 3.

$$|\Pr[\text{Succ}_3] - \Pr[\text{Succ}_2]| \leq q_{se} \cdot \text{Adv}^{\text{euf-cma}}_{\Sigma} (t', 4, 4). \quad (6)$$

Proof. Assuming that the event $\text{Forge}$ occurs, we construct, from $a$, an algorithm $\mathcal{F}$ who outputs, with a negligible probability, a forgery against the MAC scheme $\Sigma$. The algorithm $\mathcal{F}$ is given oracle access to $\text{Mac}_a(\cdot)$ and $\text{Ver}_r(\cdot)$. The goal of $\mathcal{F}$ is to produce a message/tag pair $(m, \sigma)$ such that (1) $\sigma$ is a valid tag on the message $m$ (i.e., $\text{Ver}_r(m, \sigma) = 1$) and (2) $\mathcal{F}$ had not previously queried its oracle $\text{Mac}_a(\cdot)$ on the message $m$.

Let $n$ be the number of all active sessions that $a$ initiates by asking a $\text{Send}$ query. First, $\mathcal{F}$ chooses a random $\alpha \in \{1, \ldots, n\}$. $\mathcal{F}$ then simulates the oracle calls of $a$ as in experiment $\text{Expr}_3$; except that in the $\alpha$th session, it answers $\text{Send}$ queries by accessing its MAC generation and verification oracles. If $\text{Forge}$ occurs in the $\alpha$th session, $\mathcal{F}$ halts and outputs the message/tag pair generated by $a$ as its forgery. Otherwise, $\mathcal{F}$ halts and outputs a failure indication.

This simulation is perfect unless the adversary $a$ makes a $\text{Corrupt}$ query against a participant of the $\alpha$th session. But note that the event of $a$ making such a $\text{Corrupt}$ query should not happen if $\text{Forge}$ occurs in the $\alpha$th session.

From the simulation, it is immediate that $\text{Adv}^{\text{euf-cma}}_{\Sigma}(\mathcal{F}) = \text{Pr}[\text{Forge}] / n$. Since $n \leq q_{se}$, we get $\text{Pr}[\text{Forge}] \leq q_{se} \cdot \text{Adv}^{\text{euf-cma}}_{\Sigma}(\mathcal{F})$. Then, Claim 3 follows by noticing that $\mathcal{F}$ has at most time complexity $t'$ and makes at most 4 queries to $\text{Mac}_a(\cdot)$ and $\text{Ver}_r(\cdot)$.

Experiment $\text{Expr}_4$. This experiment is different from $\text{Expr}_3$ in that the session key $sk$ of each pair of instances partnered via an $\text{Execute}$ query is chosen uniformly at random from $G$ instead of being computed as $sk = g^{y'} = X^x = Y^x$. As the following claim states, the difference in $a$'s advantage between $\text{Expr}_3$ and $\text{Expr}_4$ is negligible if the DDH assumption holds in $G$.

Claim 4.

$$|\Pr[\text{Succ}_3] - \Pr[\text{Succ}_2]| \leq \text{Adv}^{\text{ddh}}_{G} (t'). \quad (7)$$

Proof. Assume that the advantage of $a$ is nonnegligibly different between $\text{Expr}_3$ and $\text{Expr}_4$. We prove the claim by constructing, from $a$, a distinguisher $\mathcal{D}$ that solves the DDH problem in $G$. Let $(g_1, g_2, g_3) \in G^3$ be an instance of the DDH problem given as input to $\mathcal{D}$. $\mathcal{D}$ begins by choosing a bit $b$ uniformly at random. $\mathcal{D}$ then invokes $a$ as a subroutine and proceeds to simulate the oracles. $\mathcal{D}$ answers all the oracle queries of $a$ as in experiment $\text{Expr}_3$, except that it handles each $\text{Execute}(\Pi'_i, \Pi'_j, \Pi'_k)$ query by

(1) selecting two random $a_i, b_i \in Z_q$

(2) computing $X' = g_1^{a_i}$ and $Y' = g_2^{b_i}$

(3) returning a transcript generated with $X'$ and $Y'$ in place of $X$ and $Y$

(4) then setting $sk'_a = sk'_3 = g_3^{a_i \cdot b_i}$

Let $b'$ be the output of $a$. $\mathcal{D}$ outputs 1 if $b = b'$ and outputs 0, otherwise.

Then, the following is clear:

(i) The probability that $\mathcal{D}$ outputs 1 on a true Diffie-Hellman triple is exactly the probability that $a$ correctly guesses the bit $b$ in experiment $\text{Expr}_3$. 

(ii) The probability that $D$ outputs 1 on a random triple is exactly the probability that $A$ correctly guesses the bit $b$ in experiment $Expr_4$. This completes the proof of Claim 4. \hfill $\square$

**Experiment $Expr_5$.** In this experiment, the session key $sk_C$ of each instance $\Pi_C$ activated by a Send query is chosen uniformly at random from $G$ if no one in $pid_C$ has been corrupted before $\Pi_C$ determines its session identifier $sid_C$. The difference in $A$'s advantage between $Expr_4$ and $Expr_5$ is bounded by the following.

**Claim 5.**

$$\left| \Pr[Succ_5] - \Pr[Succ_4] \right| \leq \text{Adv}_{G}^{\text{DH}}\left( t' \right). \quad (8)$$

**Proof.** The proof of this claim is essentially similar to that of Claim 4. From the adversary $A$ whose advantage is non-negligibly different between $Expr_5$ and $Expr_4$, we construct a distinguisher $D$ that solves the DDH problem in $G$. Let $(g_1, g_2, g_3) \in G^3$ be an instance of the DDH problem given as input to $D$. $D$ begins by selecting a bit $b$ uniformly at random and generating a list $\text{DDHList}$ which is used to link an instance of the DDH problem to a session identifier, $D$ then runs $A$ as a subroutine and simulates the oracles. It handles all the queries of $A$ as in experiment $Expr_4$ except for Send queries.

Consider a query of the form $\text{Send}(\Pi_C, U \parallel n_j)$ which delivers a random nonce $n_j$ to instance $\Pi_C$. Whenever such a query is made, $D$ answers it as follows.

(i) If $n_j$ is not the last nonce that $\Pi_C$ is expected to receive, $D$ simply waits for the next nonce.

(ii) Otherwise, $D$ defines $sid_C$ and checks that anyone in $pid_C$ was corrupted.

(a) If so, $D$ responds to the query as in experiment $Expr_4$.

(b) If not, $D$ checks if the list $\text{DDHList}$ contains an entry of the form $(sid_C, X', Y', Z')$, where $X', Y', Z' \in G$.

1. If it does, $D$ computes $\sigma_{CS} = \text{Mac}_{k_{CS}}(C \parallel Y' \parallel sid_C)$ and returns $C \parallel Y' \parallel \sigma_{CS}$ in response to the query.

2. Otherwise, $D$ selects two random $a, b \in Z_q$, computes $X' = g_1^a, Y' = g_2^b, Z' = g_3^{ab}$, and $\sigma_{CS} = \text{Mac}_{k_{CS}}(C \parallel X' \parallel sid_C)$, returns $C \parallel X' \parallel \sigma_{CS}$ to $A$, and finally adds the tuple $(sid_C, X', Y', Z')$ to $\text{DDHList}$. When $A$ makes a Send query that causes an instance $\Pi_C$ to accept, $D$ checks if $\text{DDHList}$ contains an entry of the form $(sid_C, X', Y', Z')$. If so, $D$ sets $sk_C = Z'$. Otherwise, $D$ computes $sk_C$ as in experiment $Expr_4$. For all other Send queries of $A$, $D$ answers them as in experiment $Expr_4$. Now when $A$ terminates and outputs its guess $b'$, $D$ outputs 1 if $b = b'$ and outputs 0 otherwise.

One can easily see the following.

(i) The probability that $D$ outputs 1 on a true Diffie-Hellman triple is exactly the probability that $A$ correctly guesses the bit $b$ in experiment $Expr_4$.

(ii) The probability that $D$ outputs 1 on a random triple is exactly the probability that $A$ correctly guesses the bit $b$ in experiment $Expr_4$.

This implies Claim 5. \hfill $\square$

In experiment $Expr_5$, the session keys of all fresh instances are chosen uniformly at random from $G$ and thus the adversary $A$ obtains no information on the bit $b$ chosen by the Test oracle. Therefore, it follows that $\Pr[Succ_5] = 1/2$. This result combined with the previous claims yields the statement of Theorem 5.

4.2. Proof of Resistance to Undetectable Online Dictionary Attacks. We now claim that $3PAKE_{sm}$ is secure against undetectable online dictionary attacks as long as the $2PAKE$ protocol is SK-secure.

**Theorem 7.** Let $Undet$ be as defined in Section 2.3 and assume that for any PPT adversary $A$ asking at most $q_{se}$ Send queries, $\text{Adv}_{2PAKE}^{sk}(A')$ is only negligibly larger than $q_{se}/|PW|$. Then, for any PPT adversary $A$ asking at most $q_{se}$ Send queries, $\Pr[3PAKE_{sm} \parallel Undet]$ is only negligibly larger than $2 \cdot q_{se}/|PW|$.

**Proof.** Let $A$ be a PPT adversary who asks $q_{se}$ Send queries in mounting an undetectable online dictionary attack against $3PAKE_{sm}$. Consider the experiment $Expr_1$ described in the proof of Theorem 5 (see Section 4.1). By $Undet_1$ (resp., $Undet_s$), we denote the event $Undet$ defined in experiment $Expr_1$ (resp., $Expr_s$). We prove Theorem 7 by first proving Claim 6 and then Claim 7. \hfill $\square$

**Claim 6.** $\Pr[3PAKE_{sm} \parallel Undet_s] - \Pr[3PAKE_{sm} \parallel Undet_1]$ is only negligibly larger than $q_{se}/|PW|$.\hfill $\square$

**Claim 7.** $\Pr[3PAKE_{sm} \parallel Undet_s]$ is only negligibly larger than $q_{se}/|PW|$.\hfill $\square$

**Proof of Claim 6.** We prove the claim by constructing an adversary $A'$ who attacks the SK security of $2PAKE$ with advantage equal to $\Pr[3PAKE_{sm} \parallel Undet_1] - \Pr[3PAKE_{sm} \parallel Undet_s]$. $A'$ chooses a random bit $b \in \{0, 1\}$ and invokes the adversary $A'$ as a subroutine. $A'$ then simulates the oracles for $A$ in the exactly same way as in the simulation for the proof of Claim 1. $A'$ outputs 1 if $Undet$ occurs and 0 otherwise. From the way the oracles are simulated, it is easy to see the following.

(i) The probability that $A'$ outputs 1 when its Test oracle returns real session keys is equal to the probability that the event $Undet$ occurs in experiment $Expr_1$. \hfill $\square$
(ii) The probability that $\mathcal{A}'$ outputs 1 when its Test oracle returns random keys is equal to the probability that the event $\text{Undet}_1$ occurs in experiment $\text{Expr}_1$.

Since $\mathcal{A}'$ makes at most $q_{se}$ Send queries, we obtain the statement of Claim 6.

\textbf{Proof of Claim 7.} Assume that $\Pr_{\text{PAKE},e,m}[\text{Undet}_1] > 0$. Given the adversary $\mathcal{A}'$, we construct an adversary $\mathcal{A}$ against 2PAKE who asks at most $q_{se}$ Send queries but has an advantage nonnegligibly larger than $q_{se}/|PW|$. $\mathcal{A}$ runs $\mathcal{A}'$ as a subroutine while simulating the oracles on its own. $\mathcal{A}'$ handles all the oracle queries of $\mathcal{A}$ as in the experiment $\text{Expr}_1$ except for Send queries. When $\mathcal{A}$ makes a Send($\Pi_U^i, m$) query, $\mathcal{A}'$ checks if $m$ is a message for initiating a new session (of 3PAKEsm) or the Send query belongs to an execution of 2PAKE.

(1) If both are untrue, $\mathcal{A}'$ responds to the query as in experiment $\text{Expr}_1$.

(2) Otherwise, $\mathcal{A}'$ answers it by making the same query to its own Send oracle. If the query prompts $\Pi_U^i$ to accept, then $\mathcal{A}'$ checks if $\Pi_U^i$ is clear.

(a) If so, $\mathcal{A}'$ sets the MAC key of $\Pi_U^i$ to be a random key drawn uniformly from $\{0, 1\}^\ell$.

(b) Otherwise, $\mathcal{A}'$ makes a $\text{Reveal}(\Pi_U^i)$ query and sets the MAC key of $\Pi_U^i$ to be the output of this Reveal query.

Let $\Pi_S^i$ be any server instance against which $\mathcal{A}$ has mounted an online dictionary attack. Let $k_{5s}^i$ be the session key (i.e., the MAC key) that the instance $\Pi_S^i$ has computed in its execution of 2PAKE. In order for the instance $\Pi_S^i$ to terminate normally, the adversary $\mathcal{A}$ has to make a query of the form Send($\Pi_S^i, C \parallel \star \parallel \sigma_{CS}$) such that $\Pr_{\mathcal{A}}[\text{Verify}_{k_{5s}}(C \parallel \star \parallel \sigma_{CS}) = 1]$. When $\mathcal{A}$ makes such a Send query (i.e., when the event $\text{Undet}_1$ occurs), $\mathcal{A}'$ makes a Test query against the instance $\Pi_S^i$. Note that the instance $\Pi_S^i$ is fresh as (1) it is partnered with no instance and (2) $S$ and $C$ must have not been corrupted. Let $k_{5s}^i$ be the key returned in response to the Test query. $\mathcal{A}'$ outputs 1 if $\Pr_{\mathcal{A}}[\text{Verify}_{k_{5s}}(C \parallel \star \parallel \sigma_{CS}) = 1]$ and outputs 0, otherwise. If $\text{Undet}_1$ does not occur, $\mathcal{A}'$ outputs a random bit. Then, it is not hard to see that

$$Adv_{2PAKE}^{sk} (\mathcal{A}') = 2 \cdot \Pr_{\text{PAKE},e,m}[\text{Undet}_1] - 1$$

$$= 2 \cdot \left( \Pr_{\text{PAKE},e,m}[\text{Undet}_1] + \frac{1}{2} \left( 1 - \Pr_{\text{PAKE},e,m}[\text{Undet}_1] \right) \right) - 1$$

$$= \Pr_{\text{PAKE},e,m}[\text{Undet}_1].$$

(9)

This completes the proof of Claim 7.

\textbf{5. Conclusion}

In this work, we have presented a three-party PAKE protocol whose security does not rely on the existence of random oracles. The model that we used to prove the security of our protocol allows the adversary to ask Corrupt queries and thus captures insider attacks as well as forward secrecy. It is a known fact that proving the security of protocols in such a model is of particular importance in the three-party setting as insider dictionary attacks are most serious threats to three-party PAKE protocols. To the best of our knowledge, our protocol is the first three-party PAKE protocol proven secure against insider, active adversaries in the standard model (i.e., without random oracles and ideal ciphers). Another advantage our protocol has over previously published protocols is that it also achieves provable security against undetectable online dictionary attacks. These attacks are found to be vulnerable to an undetectable online dictionary attack. We leave it as a future work to design a three-party PAKE protocol that achieves not only provable security in the standard model but is more efficient than our protocol.

\textbf{Conflict of Interests}

The authors of the paper do not have a direct financial relation with any institution or organization mentioned in the paper that might lead to a conflict of interest for any of the authors.

\textbf{Acknowledgment}

This work was supported by the Priority Research Centers Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2010-0020210).

\textbf{References}


Submit your manuscripts at
http://www.hindawi.com