Hindawi Publishing Corporation
The Scientific World Journal
Volume 2014, Article ID 802359, 15 pages
http://dx.doi.org/10.1155/2014/802359

Research Article

Password-Only Authenticated Three-Party Key Exchange Proven Secure against Insider Dictionary Attacks

Junghyun Nam,[1] Kim-Kwang Raymond Choo,[2] Juryon Paik,[3] and Dongho Won[3]

[1] *Department of Computer Engineering, Konkuk University, 268 Chungwondaero, Chungcheongbukdo, Chungju 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, Gyeonggido, Suwon 440-746, Republic of Korea*

Correspondence should be addressed to Juryon Paik; wise96@skku.edu

Received 24 July 2014; Accepted 7 August 2014; Published 18 September 2014

Academic Editor: Jehwan Oh

While a number of protocols for password-only authenticated key exchange (PAKE) in the 3-party setting have been proposed, it still remains a challenging task to prove the security of a 3-party PAKE protocol against insider dictionary attacks. To the best of our knowledge, there is no 3-party PAKE protocol that carries a formal proof, or even definition, of security against insider dictionary attacks. In this paper, we present the first 3-party PAKE protocol proven secure against both online and offline dictionary attacks as well as insider and outsider dictionary attacks. Our construct can be viewed as a protocol compiler that transforms any 2-party PAKE protocol into a 3-party PAKE protocol with 2 additional rounds of communication. We also present a simple and intuitive approach of formally modelling dictionary attacks in the password-only 3-party setting, which significantly reduces the complexity of proving the security of 3-party PAKE protocols against dictionary attacks. In addition, we investigate the security of the well-known 3-party PAKE protocol, called GPAKE, due to Abdalla et al. (2005, 2006), and demonstrate that the security of GPAKE against online dictionary attacks depends heavily on the composition of its two building blocks, namely a 2-party PAKE protocol and a 3-party key distribution protocol.

1. Introduction

Key exchange protocols (also known as key establishment protocols) enable two or more parties communicating over a public network to establish a shared secret key. This secret key, called a *session key*, is then used for building a confidential or integrity-preserving communication channel between the involved parties. Typically, a key exchange protocol is integrated with an authentication mechanism so that each party can ensure that the session key is in fact shared with the intended parties and not with an impostor. Achieving authenticated key exchange (AKE) inevitably requires some secret information to be preestablished between the parties during an initialization phase. Password-only authenticated key exchange (PAKE) protocols, for example, are designed to work when the preestablished secret information for authentication is only a human-memorable password.

The design of secure PAKE protocols continues to be a subject of active research. A major challenge in designing a PAKE protocol is to prevent dictionary attacks, in which an attacker exhaustively enumerates all possible passwords to find out the correct password. The difficulty of designing PAKE protocols secure against dictionary attacks is compounded in the 3-party setting, in which two clients wishing to establish a session key share their individual passwords only with an authentication server but not with any other client. A 3-party PAKE protocol must prevent potential dictionary attacks by a malicious client, who is registered with the server and thus is able to set up normal protocol sessions with other clients. (Throughout the paper, we will use the term "insider attacks" to refer to attacks mounted by a registered (malicious) client and use the term "outsider attacks" to refer to attacks mounted by a nonregistered party.) Indeed, a cursory review of

the literature suggests that the majority of attacks mounted against 3-party PAKE protocols fall into the category of insider dictionary attacks; see, for example, Section 2 as well as [1–6].

The difficulties of obtaining a high level of assurance in the security of almost any new, or even existing, protocol are well illustrated with examples of errors found in many such protocols years after they were published. A widely accepted approach is to use a deductive reasoning process whereby the emphasis is placed on a proven reduction from the problem of breaking the protocol to another problem believed to be hard. Such an approach for key exchange protocols was made popular by Bellare and Rogaway [7] who provide the first formal definition for a model of adversary capabilities with an associated definition of the indistinguishability-based security. This model has been further revised several times, and one of the more recent revisions is the real-or-random (ROR) model proposed by Abdalla et al. [8, 9] for PAKE protocols. They also proposed a generic construction of a 3-party PAKE protocol that allows existing provably secure 2-party password-based key exchange and 3-party symmetric key distribution protocols to be *plugged and played* in a modular approach and yet remain secure [8, 9].

To date, a number of 3-party PAKE protocols have been presented with a formal proof of security [1, 8–17]. Most of these protocols were proven secure only in a restricted model, in which the adversary is not allowed to corrupt protocol participants [1, 8–10, 12–14]. In other words, these protocols were not proven secure against insider attacks including dictionary attacks conducted by insiders. Some protocols [11, 13] were subsequently found to be flawed [6, 18] and several protocols [15, 16] were proven secure only in a model that cannot capture (both insider and outsider) online dictionary attacks. Although protocols such as those of [19, 20] claimed to be provably secure against dictionary attacks of any kind, these protocols assume a "hybrid" 3-party setting where a server's public key is required in addition to passwords (see [21–27] for other protocols designed to work in a hybrid setting). To the best of our knowledge, no 3-party PAKE protocol has been proven secure against insider dictionary attacks.

We regard the contributions of this paper to be threefold.

Contribution 1. We present *the first 3-party PAKE protocol, a hashed variant of the protocol of [1], whose indistinguishability-based security as well as password security against all classes of dictionary attacks are formally proved in a well-defined communication model*. Similar to the protocols of [1, 8, 9], our proposed protocol is generic in the sense that it can be constructed from any 2-party AKE protocol. Our construct can be viewed as a protocol compiler that transforms any 2-party AKE protocol into a 3-party AKE protocol with 2 more rounds of communication. If the given 2-party protocol is password-only authenticated, then the 3-party protocol output by the compiler will also be password-only authenticated. We prove the security of our construct in the random oracle model under the gap Diffie-Hellman (GDH) assumption; see Section 4.

Contribution 2. We offer *a simple and intuitive approach of capturing dictionary attacks* in the widely accepted model of Bellare et al. [28]. First, we clarify the relationship between the indistinguishability-based security of session keys and the password security against dictionary attacks.

(i) The indistinguishability-based security property in the Bellare-Pointcheval-Rogaway model implies security against (both insider and outsider) offline dictionary attacks. We demonstrate this by showing that a protocol cannot achieve the indistinguishability-based security if it is not secure against an offline dictionary attack (see Section 3.3).

(ii) The indistinguishability-based security property does not imply security against undetectable online dictionary attacks (referred to as "UD online dictionary attacks" in the remainder of this paper). It is important to note that a protocol may be insecure against a UD online dictionary attack but it can still achieve the indistinguishability-based security (see Section 3.3).

This observation allows us to exclude offline dictionary attacks from our consideration once we have proved the indistinguishability-based security. We then introduce a new security definition to capture UD online dictionary attacks. We claim that our approach, when compared to those of [19, 20] (where a separate security definition is introduced to capture both UD online and offline dictionary attacks), provides a more intuitive and simpler way of proving security against dictionary attacks (see Section 3).

Contribution 3. We revisit the generic protocol GPAKE of Abdalla et al. [8, 9] to provide a detailed analysis of its security against UD online dictionary attacks. (We note that the work of Wang and Hu [1] has only provided a sketch of an insider UD online dictionary attack against GPAKE.) GPAKE is parameterized with a 2-party PAKE protocol 2PAKE and a 3-party key distribution protocol 3KD. We found that the security of GPAKE against both insider and outsider UD online dictionary attacks depends heavily on the security properties provided by 2PAKE and 3KD. For example, we can launch an insider UD online dictionary attack against the protocol GPAKE if neither 2PAKE nor 3KD provides client-to-server authentication. An outsider UD online dictionary attack can also be mounted under the same condition in addition to the condition that 2PAKE provides server-to-client authentication (see Section 2 for details).

2. Undetectable Online Dictionary Attacks against GPAKE

This section presents a comprehensive analysis of the security of the GPAKE protocol [8, 9] against UD online dictionary attacks. Our analysis shows that (1) GPAKE relies its security against UD online dictionary attacks on how its building blocks are instantiated and (2) the attacks could be mounted not only by a registered client (an insider) but also by a nonregistered party (an outsider). After conducting the

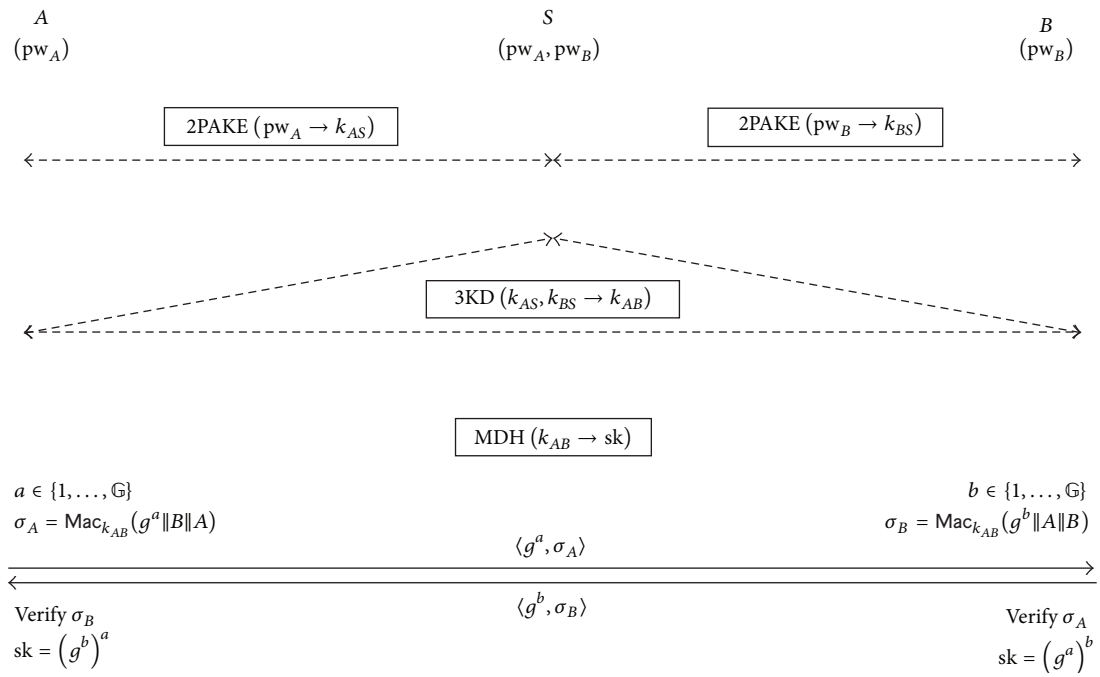

Figure 1: GPAKE: Abdalla et al.'s generic 3-party PAKE protocol [8, 9].

security analysis, we suggest possible combinations of the instantiations that allow the GPAKE protocol to avoid the attacks.

2.1. A Review of GPAKE. The GPAKE protocol is constructed using a combination of three building blocks: (1) a 2-party PAKE protocol 2PAKE, (2) a 3-party key distribution protocol 3KD, and (3) a message authentication code- (MAC-) based Diffie-Hellman key exchange protocol MDH. Let $\mathbb{G}$ be a finite cyclic group generated by an element g; and let $\Sigma = (\mathsf{Mac}, \mathsf{Ver})$ be a MAC scheme where Mac and Ver are the MAC generation and MAC verification algorithms, respectively. The security of MDH relies on the hardness of the decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$ and on the security of the underlying MAC scheme Σ against chosen message attacks. S represents a trusted server which registers clients and, during the registration, each client shares their individual password secretly with S. Suppose that A and B are two registered clients who wish to establish a session key and pw_A and pw_B are the passwords of A and B, respectively.

2.1.1. Description. A high-level depiction of GPAKE is given in Figure 1 and its brief description follows. First, A (and B) and S establish a shared high-entropy key, k_{AS} (and k_{BS}, resp.), by running the 2-party PAKE protocol, 2PAKE. Then, S generates a MAC key, k_{AB}, and distributes k_{AB} to both A and B by running the 3-party key distribution protocol 3KD which takes k_{AS} and k_{BS} as input. A session key, $sk = g^{ab}$, will be established between A and B after running the MAC-based Diffie-Hellman key exchange protocol MDH which takes k_{AB} as an input.

2.1.2. Instantiations. Abdalla et al. [8, 9] suggested several practical and provably secure protocols that can be used in the instantiation of the 2-party protocol 2PAKE. Among them are the KOY protocol [29, 30] and its generalization [31], the PAK suite [32], and other protocols based on encrypted key exchange (EKE) of Bellovin and Merritt [33] (e.g., Bresson et al.'s OMDHKE protocol [34]). For the instantiation of 3KD, any particular choice that is secure with respect to a single session will do since the symmetric keys given as input to 3KD differ from session to session. Bellare and Rogaway's 3PKD protocol [35], for example, was suggested as a possible choice. The MAC scheme used in MDH can be instantiated with the HMAC [36] and the CBC MAC.

2.2. An Insider Attack. Let us consider, for example, the case where 2PAKE and 3KD are instantiated with the PPK protocol [32] and with the 3PKD protocol [35], respectively. In this case, a malicious client B who is registered with the server, S, can mount a UD online dictionary attack against any other client, for example, A. It is important to note that we do not require either of the protocols, PPK (see Figure 2) and 3PKD (see Figure 3), to be insecure and, in fact, we assume that both protocols are secure.

2.2.1. The PPK Protocol. Let $\mathbb{G}$ be a finite cyclic group of order q, and let g be a generator of $\mathbb{G}$. The three hash functions used are $G_1 : \{0,1\}^* \to \mathbb{G}$, $G_2 : \{0,1\}^* \to \mathbb{G}$, and $H : \{0,1\}^* \to \{0,1\}^\ell$, where ℓ is the length of the session key. pw_A denotes A's password known only to A and $(\overline{v}_A, w_A) = ((G_1(pw_A))^{-1}, G_2(pw_A))$ denotes the password verifiers held by S. The function ACCEPTABLE$(*)$ returns true, if and only if $* \in \mathbb{G}$ (or, $* \in \widehat{\mathbb{G}}$ when $\mathbb{G}$ is defined as a (proper) subgroup

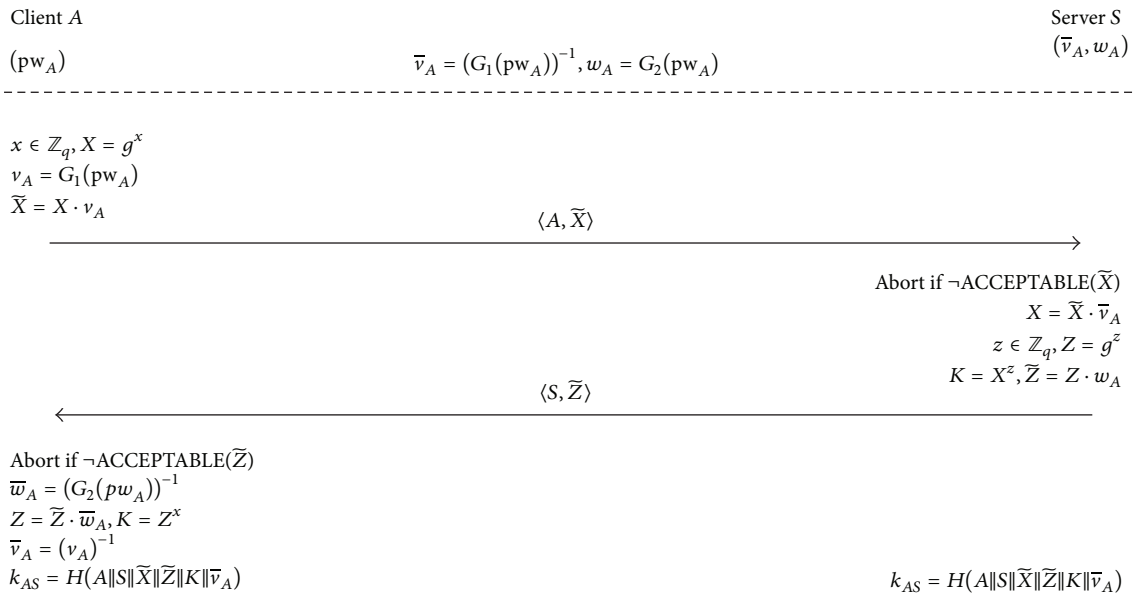

Figure 2: The PPK protocol [32].

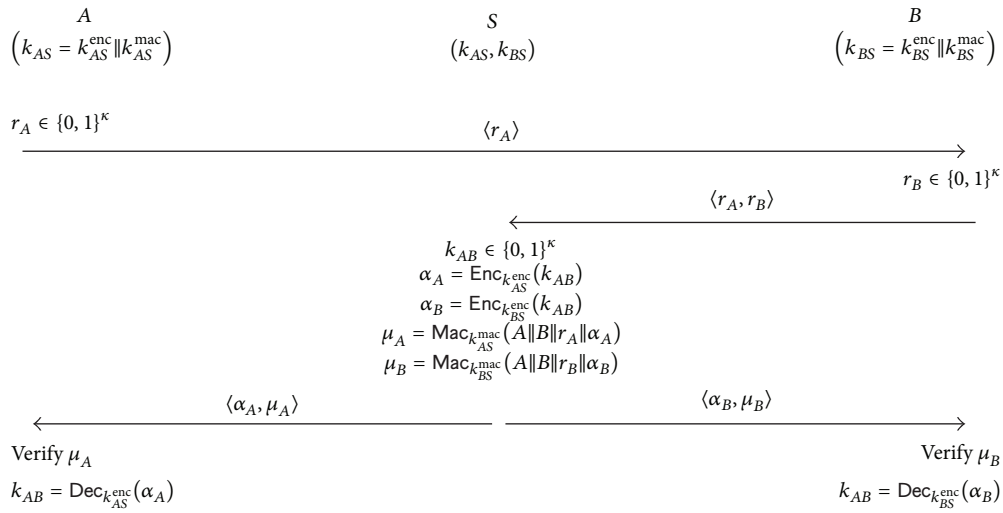

Figure 3: The 3PKD protocol [35].

of a group $\widehat{\mathbb{G}}$). The session key k_{AS} is computed as $k_{AS} = H(A\|S\|\widetilde{X}\|\widetilde{Z}\|K\|\overline{v}_A)$. We observe that two messages $\langle A, \widetilde{X} \rangle$ and $\langle S, \widetilde{Z} \rangle$ are independent and thus the protocol can be easily modified to run in a single round.

2.2.2. The 3PKD Protocol. The cryptographic tools used in 3PKD include (1) a symmetric encryption scheme consisting of a pair of encryption/decryption algorithms (Enc, Dec) and (2) a MAC scheme consisting of a pair of MAC generation/verification algorithms (Mac, Ver). The protocol runs between a trusted server, S, and two clients, A and B. A (and B) and S are assumed to have preestablished a 2κ-bit secret $k_{AS} = k_{AS}^{\mathrm{enc}} \| k_{AS}^{\mathrm{mac}}$ (resp., $k_{BS} = k_{BS}^{\mathrm{enc}} \| k_{BS}^{\mathrm{mac}}$). The protocol, depicted in Figure 3, begins by having A choosing a random κ-bit challenge r_A and sending it to client B. B also chooses a random κ-bit challenge r_B and sends $\langle r_A, r_B \rangle$ to S. Upon receiving $\langle r_A, r_B \rangle$, S generates a session key k_{AB} which he will distribute. S then encrypts k_{AB} under A's encryption key k_{AS}^{enc} (resp., B's encryption key k_{BS}^{enc}) to get ciphertext α_A (resp., α_B). Then, S computes μ_A (resp. ,μ_B), the MAC under key k_{AS}^{mac} (resp. k_{BS}^{mac}) of the string $A\|B\|r_A\|\alpha_A$ (resp. $A\|B\|r_B\|\alpha_B$). Then, S sends $\langle \alpha_A, \mu_A \rangle$ and $\langle \alpha_B, \mu_B \rangle$ to A and B, respectively. A and B accept the session key k_{AB} if and only if their received MAC is valid.

In order for PPK and 3PKD to be used together, the session keys generated by PPK need to be at least 2κ-bit long.

2.2.3. One Possible Attack Scenario. The malicious client, B, can mount the following UD online dictionary attack against another client, A.

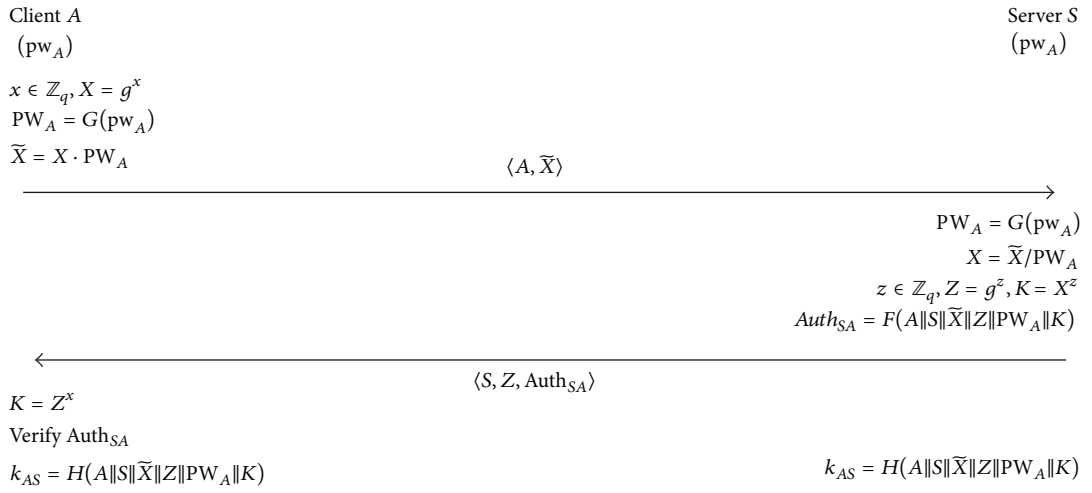

Figure 4: The OMDHKE protocol [34].

Step 1 (running PPK). B begins by initiating two concurrent runs, R1 & R2, of the PPK protocol.

(i) In the (dishonest) run R1, B impersonating A makes a guess (denoted by pw'_A) on A's actual password, pw_A, selects a random $x \in \mathbb{Z}_q$, computes $\widetilde{X}' = g^x \cdot G_1(pw'_A)$, and sends the server S a fabricated message $\langle A, \widetilde{X}' \rangle$. Since $\widetilde{X}' \in \mathbb{G}$, S will respond with the message $\langle S, \widetilde{Z} \rangle$. After obtaining $\langle S, \widetilde{Z} \rangle$, B computes a key $k'_{AS} = H(A\|S\|\widetilde{X}'\|\widetilde{Z}\|K'\|\overline{v}'_A)$, where $K' = (\widetilde{Z} \cdot (G_2(pw'_A))^{-1})^x$ and $\overline{v}'_A = (G_1(pw'_A))^{-1}$. We let $k'_{AS} = k'^{\mathrm{enc}}_{AS} \| k'^{\mathrm{mac}}_{AS}$.

(ii) In the (honest) run R2, B honestly performs all the operations as per protocol specification and establishes the key, $k_{BS} = k^{\mathrm{enc}}_{BS} \| k^{\mathrm{mac}}_{BS}$.

Step 2 (running 3PKD). Once R1 & R2 are completed, B proceeds to run the protocol 3PKD by sending the server S two random challenges r_A and r_B (selected as specified by the protocol). S will respond to the random challenges with two messages $\langle \alpha_A, \mu_A \rangle$ and $\langle \alpha_B, \mu_B \rangle$. After obtaining these two messages, B recovers the keys $k'_{AB} = \mathsf{Dec}_{k'^{\mathrm{enc}}_{AS}}(\alpha_A)$ and $k_{AB} = \mathsf{Dec}_{k^{\mathrm{enc}}_{BS}}(\alpha_B)$.

Step 3 (verifying the password guess). B verifies the correctness of pw'_A by checking that k'_{AB} is equal to k_{AB}. If they are equal, pw'_A is the correct password with an overwhelming probability. Otherwise, it means that $pw'_A \neq pw_A$.

2.3. An Outsider Attack. We now consider the case where 2PAKE and 3KD are instantiated with the OMDHKE protocol [34] and with the 3PKD protocol [35], respectively. Although OMDHKE (see Figure 4) is largely similar to PPK, there is a marked difference between the two. OMDHKE provides server-to-client authentication while PPK focuses on implicit key authentication.

2.3.1. The OMDHKE Protocol. Let $\mathbb{G}$ be a finite cyclic group generated by an element g of prime order q. The hash functions used are $G : \{0,1\}^* \to \mathbb{G}$, $H : \{0,1\}^* \to \{0,1\}^{\ell}$, and $F : \{0,1\}^* \to \{0,1\}^f$, where ℓ is the length of the session key and f is the length of the authenticator Auth_{SA}. Unlike the PPK protocol described in Figure 2, both A and S have a shared password, pw_A. The protocol starts when A chooses a random $x \in \mathbb{Z}_q$, computes $X = g^x$, $PW_A = G(pw_A)$, and $\widetilde{X} = X \cdot PW_A$, and sends $\langle A, \widetilde{X} \rangle$ to S. Upon receiving the message from A, S computes $PW_A = G(pw_A)$ and $X = \widetilde{X}/PW_A$, chooses a random $z \in \mathbb{Z}_q$, and computes $Z = g^z$, $K = X^z$, and $\mathrm{Auth}_{SA} = F(A\|S\|\widetilde{X}\|Z\|PW_A\|K)$. Then S sends $\langle S, Z, \mathrm{Auth}_{SA} \rangle$ to A and computes the session key, $k_{AS} = H(A\|S\|\widetilde{X}\|Z\|PW_A\|K)$. A computes $K = Z^x$, verifies Auth_{SA}, and computes the session key $k_{AS} = H(A\|S\|\widetilde{X}\|Z\|PW_A\|K)$ if the verification succeeds.

Note that, in OMDHKE, server-to-client authentication is achieved via the authenticator Auth_{SA} sent by S to A.

2.3.2. One Possible Attack Scenario. We now demonstrate how C, a malicious adversary who is not registered with the server, can mount a UD online dictionary attack against two registered clients, A and B.

Step 4 (running OMDHKE). C initiates two concurrent runs, R1 & R2, of the protocol OMDHKE.

(i) In the (dishonest) run R1, C impersonating A makes a guess (denoted by pw'_A) on A's password, pw_A, chooses a random $x \in \mathbb{Z}_q$, computes $PW'_A = G(pw'_A)$ and $\widetilde{X}' = g^x \cdot PW'_A$, and sends the fabricated message $\langle A, \widetilde{X}' \rangle$ to S. C will then receive the response, $\langle S, Z, \mathrm{Auth}_{SA} \rangle$, from S as per protocol specification.

(ii) In the (dishonest) run R2, C proceeds as per R1 but impersonating B (instead of A) and making a guess (denoted as pw'_B) on B's password, pw_B. Let Auth_{SB} denote the authenticator received (in response) from S.

Step 5 (running 3PKD). After R1 & R2 are completed, C runs the 3PKD protocol with S while impersonating both A and B.

(This step provides no useful information to C but is required for the attack to go undetected.)

Step 6 (verifying the password guess). C can then verify the correctness of pw'_A by computing $\text{Auth}'_{SA} = F(A\|S\|\widetilde{X}'\|Z\|PW'_A\|Z^x)$ and then checking if Auth'_{SA} is equal to Auth_{SA}. If they are equal, pw'_A is the correct password (with an overwhelming probability). Otherwise, C knows that $pw'_A \neq pw_A$. Similarly, the correctness of pw'_B can be verified using the received Auth_{SB}.

2.4. Discussion. The insider attack described in Section 2.2 works because neither PPK nor 3PKD provides client-to-server authentication. Indeed, the same attack also works if we replace the PPK protocol with the OMDHKE protocol [34], the EKE2 protocol [28], or the SPAKE protocol [37]. In other words, the GPAKE protocol becomes vulnerable to the insider attack when both 2PAKE and 3KD are instantiated with a protocol that does not provide client-to-server authentication. The outsider attack described in Section 2.3 works under the same circumstance but additionally exploits the fact that the OMDHKE protocol provides server-to-client authentication.

Informally, both attacks can be prevented if one of the two protocols, 2PAKE or 3KD, is instantiated with a protocol that provides client-to-server authentication. We observe that a typical 3-party key distribution protocol is not expected to provide client-to-server authentication, and hence, we suggest that the countermeasure targets the instantiation of 2PAKE. While some might also suggest that a round-optimal protocol (i.e., a protocol that runs in a single round) should be used in the instantiation of 2PAKE to achieve better efficiency, we caution against this as no round-optimal 2-party PAKE protocol is known to provide client-to-server authentication and achieve security against offline dictionary attacks.

3. Modelling Dictionary Attacks in the Password-Only 3-Party Setting

The ROR model used for security analysis of GPAKE [8, 9] does not allow the adversary to access the Corrupt oracle and thus cannot capture any kind of insider attacks, in particular, (UD) online and offline dictionary attacks by a malicious insider. The security definition associated with the ROR model intends to capture indistinguishability of session keys and does not consider mounting an online dictionary attack against a protocol to be a violation of the security of the protocol (see Section 3.3). Consequently, none of the online dictionary attacks presented in Section 2 can be captured in the model.

We begin this section by presenting a communication model adapted from the Bellare-Pointcheval-Rogaway 2000 model [28] to support key exchange in the password-only 3-party setting. Our communication model allows the adversary to ask Corrupt queries and thereby captures insider attacks (as well as forward secrecy and unknown key share attacks). We then define a typical indistinguishability-based security of session keys, which we call the *SK security*. As we demonstrate in Section 3.3, the SK security implies security against offline dictionary attacks but does not imply security against UD online dictionary attacks. We then introduce a separate security definition to capture UD online dictionary attacks. Unlike the approach of [19, 20] where a separate security definition is introduced to capture both online and offline dictionary attacks, we only need to prove the protocol secure against UD online dictionary attacks once we have proved that it is SK-secure.

3.1. The Communication Model

3.1.1. Participants and Long-Term Keys. We denote S by a trusted authentication server and $\mathscr{C}$ by the set of all clients registered with S. During registration, each client $C \in \mathscr{C}$ selects a password, pw_C, from a dictionary, $\mathscr{D}$, and shares pw_C with S via a secure channel. pw_C is used as the long-term secret key shared between C and S. Any two clients, $C, C' \in \mathscr{C}$, may run a 3-party PAKE protocol P with S at any point in time to establish a session key. Let $\mathscr{U} = \mathscr{C} \cup \{S\}$. A user, $U \in \mathscr{U}$, may participate in multiple protocol sessions running, either serially or concurrently, with the same or different participants. Thus, at any given time, there could be multiple instances of a single user. Π^i_U denotes instance i of user U. We say that a client instance, Π^i_C, *accepts* when it computes its session key, sk^i_C, in an execution of the protocol.

3.1.2. Partnering. We say, informally, that two instances are *partners* if they participate in a protocol execution and establish a (shared) session key. Formally, partnering between instances is defined in terms of the notions of session and partner identifiers (See [38] 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.) Session identifier (sid) is a unique identifier of a protocol session and is usually defined as a function of the messages transmitted in the session (although this may not be possible in a multiparty protocol where not all participants have the same view). sid^i_U denotes the sid of instance Π^i_U. A partner identifier (pid) is a sequence of identities of participants of a specific protocol session. Instances are given as input a pid before they can run the protocol. pid^i_U denotes the pid given to instance Π^i_U. Note that $\text{pid}^i_C = \langle C, C', S\rangle$, where C' is another client with whom Π^i_C believes it runs the protocol. We say that two instances, Π^i_C and $\Pi^j_{C'}$, are partners if the following holds: (1) both Π^i_C and $\Pi^j_{C'}$ have accepted, (2) $\text{sid}^i_C = \text{sid}^j_{C'}$, and (3) $\text{pid}^i_C = \text{pid}^j_{C'}$.

3.1.3. Adversary Capabilities. The probabilistic polynomial-time (PPT) adversary $\mathscr{A}$ is in complete control of all communications between users, and its capabilities are modeled via a predefined set of oracle queries described below.

(i) Execute $(\Pi^i_C, \Pi^j_{C'}, \Pi^k_S)$: this query models passive attacks against the protocol. It prompts an execution of the protocol between the instances Π^i_C, $\Pi^j_{C'}$ and Π^k_S and returns the transcript of the protocol execution to $\mathscr{A}$.

Experiment $\mathbf{Exp_0}$:
Phase 1. $\mathcal{A}$ makes any oracle queries at will as many times as it wishes, except that:
(1) $\mathcal{A}$ is not allowed to ask the $\mathsf{Test}(\Pi^i_C)$ query if the instance Π^i_C is unfresh.
(2) $\mathcal{A}$ is not allowed to ask the $\mathsf{Reveal}(\Pi^i_C)$ query if it has already made a Test query to Π^i_C or $\Pi^j_{C'}$, where $\Pi^j_{C'}$ is the partner of Π^i_C.
Phase 2. Once $\mathcal{A}$ decides that Phase 1 is over, it outputs a bit b' as a guess on the hidden bit b chosen by the Test oracle. $\mathcal{A}$ is said to succeed if $b = b'$.

Box 1

(ii) $\mathsf{Send}(\Pi^i_U, m)$: this query sends message m to instance Π^i_U, modelling active attacks against the protocol. Upon receiving m, the instance Π^i_U proceeds according to the protocol specification. The message output by Π^i_U, if any, is returned to $\mathcal{A}$. A query of the form $\mathsf{Send}(\Pi^i_C, \text{start}: \langle C, C', S\rangle)$ prompts Π^i_C to initiate the protocol with $\text{pid}^i_C = \langle C, C', S\rangle$.

(iii) $\mathsf{Reveal}(\Pi^i_C)$: this query captures the notion of known key security (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). The instance Π^i_C, upon receiving the query and if it has accepted, returns the session key, sk^i_C, back to $\mathcal{A}$.

(iv) $\mathsf{Corrupt}(U)$: this query returns the password pw_U of U. If $U = S$ (i.e., the server is corrupted), all clients' passwords stored by the server are returned. This query captures not only the notion of forward secrecy but also unknown key share attacks and insider attacks.

(v) $\mathsf{Test}(\Pi^i_C)$: this query is used to define the indistinguishability-based security of the protocol. If Π^i_C has accepted, then, depending on a randomly chosen bit b, $\mathcal{A}$ is given either the real session key sk^i_C (when $b = 1$) or a random key drawn from the session-key space (when $b = 0$). Following the ROR model [8, 9], we allow $\mathcal{A}$ to ask as many Test queries as it wishes. All Test queries are answered using the same value of the hidden bit b. Namely, the keys output by the Test oracle are either all real or all random. But, we require that, for each different set of partners, $\mathcal{A}$ should access the Test oracle only once.

We describe the number of queries asked by an adversary as the *query complexity* of the adversary. The query complexity is represented as an ordered sequence of five nonnegative integers, $Q = \langle q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}}, q_{\text{corr}}, q_{\text{test}}\rangle$, where q_{exec}, q_{send}, q_{reve}, q_{corr}, and q_{test} refer to the numbers of queries that the adversary asked, respectively, to the $\mathsf{Execute}$, Send, Reveal, $\mathsf{Corrupt}$, and Test oracles.

3.2. Session Key (SK) Security. We now proceed to define the basic security, called the SK security, of a 3-party PAKE protocol. The notion of *freshness* is a key element in defining the SK security. Intuitively, a fresh instance is one that holds a session key which should not be known to the adversary $\mathcal{A}$, and an unfresh instance is one whose session key (or some information about the key) can be known by trivial means. A formal definition of freshness follows.

Definition 11. An instance Π^i_C is fresh unless one of the following occurs: (1) $\mathcal{A}$ queries $\mathsf{Reveal}(\Pi^i_C)$ or $\mathsf{Reveal}(\Pi^j_{C'})$, where $\Pi^j_{C'}$ is the partner of Π^i_C; or (2) $\mathcal{A}$ queries $\mathsf{Corrupt}(U)$, for some $U \in \text{pid}^i_C$, before Π^i_C or its partner $\Pi^j_{C'}$ accepts.

The SK security of a 3-party PAKE protocol P is defined in the context of Box 1.

Let Succ_0 be the event that $\mathcal{A}$ succeeds in the experiment $\mathbf{Exp}_0$. Let $\mathsf{Adv}_P(\mathcal{A})$ denote the advantage of $\mathcal{A}$ in attacking protocol P and be defined as $\mathsf{Adv}_P(\mathcal{A}) = 2 \cdot \Pr_{P,\mathcal{A}}[\mathsf{Succ}_0] - 1$.

Definition 12. A 3-party PAKE protocol P is *SK-secure* if, for any PPT adversary $\mathcal{A}$ asking at most q_{send} Send queries, $\mathsf{Adv}_P(\mathcal{A})$ is only negligibly larger than $c \cdot q_{\text{send}}/|\mathcal{D}|$, where c is a very small constant (usually around 2 or 4) when compared with $|\mathcal{D}|$.

To quantify the security of protocol P in terms of the amount of resources expended by adversaries, we let $\mathsf{Adv}_P(t, Q)$ denote the maximum value of $\mathsf{Adv}_P(\mathcal{A})$ over all PPT adversaries $\mathcal{A}$ with time complexity at most t and query complexity at most Q.

3.3. Password Security

3.3.1. Capturing Offline Dictionary Attacks. The SK security described in Definition 12 implies security against offline dictionary attacks. In other words, a 3-party PAKE protocol P is not SK-secure if it is not secure against an offline dictionary attack. To demonstrate this, suppose that the protocol P is not secure against an offline dictionary attack whereby an attacker B can derive the password of any registered client A. Then we can construct an adversary $\mathcal{A}_{\text{off}}$ who breaks the SK security of protocol P as follows.

Corruption. If B is a malicious insider, $\mathcal{A}_{\text{off}}$ queries $\mathsf{Corrupt}(B)$ to obtain the password pw_B. Otherwise, $\mathcal{A}_{\text{off}}$ skips this step.

Dictionary Attack. Next, $\mathcal{A}_{\text{off}}$ runs the protocol P exactly in the way that B conducts its offline dictionary attack against client A. Note that $\mathcal{A}_{\text{off}}$ can perfectly simulate B's attack by using the disclosed password pw_B and by asking oracle

queries appropriately. At the end of this step, $\mathcal{A}_{\text{off}}$ will obtain the password pw_A of client A as a result of the attack.

Impersonation. Now, $\mathcal{A}_{\text{off}}$ initiates a new protocol session by querying $\mathsf{Send}(\Pi_C^i, \text{start}: \langle A, C, S\rangle)$, where Π_C^i is an unused instance of an uncorrupted client C. $\mathcal{A}_{\text{off}}$ 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 Π_C^i will accept with its session key sk_C^i.

Test. Clearly the instance Π_C^i is fresh, since (1) no Reveal query has been made on Π_C^i or its partner (which does not exist in this case) and (2) no $\mathsf{Corrupt}$ query has been made against any of A, C, and S. Thus, $\mathcal{A}_{\text{off}}$ may ask the $\mathsf{Test}(\Pi_C^i)$ query. Since $\mathcal{A}_{\text{off}}$ can compute the same session key as sk_C^i, the probability that $\mathcal{A}_{\text{off}}$ correctly guesses the bit b chosen by the Test oracle is 1 and so is the advantage of $\mathcal{A}_{\text{off}}$ in attacking the protocol. Then, by Definition 12, the protocol P is not SK-secure since the number of Send queries asked by $\mathcal{A}_{\text{off}}$ is much smaller (i.e., nonnegligibly smaller) than $|\mathcal{D}|/c$.

3.3.2. Capturing Undetectable Online Dictionary Attacks. Unfortunately, the SK security does not imply security against UD online dictionary attacks. In other words, a 3-party PAKE protocol that is not secure against a UD online dictionary attack may be rendered SK-secure. Let us assume a 3-party PAKE protocol P that is susceptible to a UD online dictionary attack (e.g., the GPAKE protocol in Section 2). Then, we can construct an adversary $\mathcal{A}_{\text{on}}$ who attacks protocol P with advantage 1. The construction of $\mathcal{A}_{\text{on}}$ is the same as that of $\mathcal{A}_{\text{off}}$, except that to correctly determine the password pw_A, $\mathcal{A}_{\text{on}}$ may have to ask Send queries as many times as $d \cdot |\mathcal{D}|$ for some integer $d \geq 1$. Note that verifying the correctness of a password guess may require more than one Send query to be asked. Even if $\mathsf{Adv}_P(\mathcal{A}_{\text{on}}) = 1$, the protocol P is still rendered SK-secure by Definition 12, as the following holds for some $c \geq 1$:

$$\mathsf{Adv}_P(\mathcal{A}_{\text{on}}) \leq \frac{cd\,|\mathcal{D}|}{|\mathcal{D}|}. \tag{1}$$

This result is not surprising since we call a protocol SK-secure if mounting an online dictionary attack by asking Send queries is the best an adversary can do. However, we want to be able to distinguish UD 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 UD online dictionary attacks.

Definition 13 (an online dictionary attack). The $\mathsf{Send}(\Pi_S^k, m)$ query models an *online dictionary attack* if both the following are true at the time of the termination of instance Π_S^k: (1) m was not output by a previous Send query asked to an instance of C by which Π_S^k believes m was sent and (2) the adversary $\mathcal{A}$ queried neither $\mathsf{Corrupt}(S)$ nor $\mathsf{Corrupt}(C)$.

In Definition 13, 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 $\mathcal{A}$ 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 Reveal queries.

Let Undet be the event that, in experiment $\mathbf{Exp}_0$, a server instance terminates normally when an online dictionary attack was mounted against the instance. We say that the adversary $\mathcal{A}$ succeeds in mounting an UD online dictionary attack if the event Undet occurs. Formally, we define protocol's security against UD online dictionary attacks as follows:

Definition 14. A 3-party PAKE protocol P is secure against a UD online dictionary attack if, for any PPT adversary $\mathcal{A}$ asking at most q_{send} Send queries, $\Pr_{P,\mathcal{A}}[\mathsf{Undet}]$ is only negligibly larger than $c \cdot q_{\text{send}}/|\mathcal{D}|$, where c is as defined in Definition 12.

4. A Compiler for 3-Party PAKE Protocols

We now present a protocol compiler that transforms any 2-party PAKE protocol into a 3-party PAKE protocol. If the given 2-party protocol is SK-secure, then the 3-party protocol output by the compiler is not only SK-secure but also secure against both insider and outsider UD online dictionary attacks. (We stress again that the SK security implies resistance against both insider and outsider offline dictionary attacks.) This is the case regardless of whether the underlying 2-party protocol provides client-to-server authentication or not. Our transformation does not require the use of a 3-party key distribution protocol and always takes only two additional rounds of communication. Hence, applying the compiler to a round-optimal 2-party PAKE protocol immediately yields a 3-party PAKE protocol running in three communication rounds.

Our generic construction, which we call H3PAKE ("H" for "hashed"), is a variant of NGPAKE that is the generic construction of Wang and Hu [1]. The key difference between H3PAKE and NGPAKE is in the computation of the session key. NGPAKE defines the session key simply as the Diffie-Hellman key g^{xy}, whilst H3PAKE defines the session key as $H(\mathsf{pid}\|\mathsf{sid}\|g^{xy})$ where H is a cryptographic hash function. The difference in how session key is computed, together with a minor modification in the specifications of the protocol messages, results in a significant improvement on the security of the constructions. More specifically, we are now able to prove that H3PAKE is secure against insider dictionary attacks, unlike NGPAKE where it is unclear whether it can be proven secure against insider dictionary attacks. Note that the security of NGPAKE was proved in the ROR model that does not allow the adversary to ask $\mathsf{Corrupt}$ queries; and as shown in Section 3.3, protocols proven secure in such a model cannot claim provable security against insider attacks of any kind.

4.1. Preliminaries. We begin with the cryptographic primitives on which the security of our construction relies.

Gap Diffie-Hellman (GDH) Assumption. Consider a finite cyclic group $\mathbb{G}$ of prime order q where the operation is denoted multiplicatively. Since the order of $\mathbb{G}$ is prime, all the elements of $\mathbb{G}$, except 1, are generators of $\mathbb{G}$. Let g be a random generator of $\mathbb{G}$. The GDH problem in $\mathbb{G}$ is to solve the computational Diffie-Hellman (CDH) problem in $\mathbb{G}$ when given an oracle $\mathcal{O}(\cdot,\cdot,\cdot)$ that solves the decisional Diffie-Hellman (DDH) problem in $\mathbb{G}$. The DDH oracle $\mathcal{O}(\cdot,\cdot,\cdot)$, on input a triple (g^a, g^b, C) for $a, b \in \mathbb{Z}_q$, outputs 1 if and only if $C = g^{ab}$. We define the advantage of a PPT algorithm $\mathcal{A}$ in solving the GDH problem in $\mathbb{G}$ as $\mathsf{Adv}_{\mathbb{G}}^{\mathrm{GDH}}(\mathcal{A}) = \Pr[\mathcal{A}^{\mathcal{O}(\cdot,\cdot,\cdot)}(\mathbb{G}, q, g, g^x, g^y) = g^{xy} : x, y \in_R \mathbb{Z}_q]$. We say that the GDH assumption holds in $\mathbb{G}$ if $\mathsf{Adv}_{\mathbb{G}}^{\mathrm{GDH}}(\mathcal{A})$ is negligible for all PPT algorithms $\mathcal{A}$. We denote by $\mathsf{Adv}_{\mathbb{G}}^{\mathrm{GDH}}(t)$ the maximum value of $\mathsf{Adv}_{\mathbb{G}}^{\mathrm{GDH}}(\mathcal{A})$ over all algorithms $\mathcal{A}$ running in time at most t.

Message Authentication Codes. A message authentication code (MAC) scheme Σ is a triple of efficient algorithms $(\mathsf{Gen}, \mathsf{Mac}, \mathsf{Ver})$ where (1) the key generation algorithm Gen takes as input a security parameter 1^ℓ and outputs a key k chosen uniformly at random from $\{0,1\}^\ell$; (2) the MAC generation algorithm Mac takes as input a key k and a message m and outputs a MAC (also known as a tag) σ; and (3) the MAC verification algorithm Ver takes as input a key k, a message m, and a MAC σ and outputs 1 if σ is valid for m under k or outputs 0 if σ is invalid. Let $\mathsf{Adv}_\Sigma(\mathcal{A})$ be the advantage of an adversary $\mathcal{A}$ in violating the strong existential unforgeability of Σ under adaptive chosen message attacks. More precisely, $\mathsf{Adv}_\Sigma(\mathcal{A})$ is the probability that an adversary $\mathcal{A}$, who mounts an adaptive chosen message attack against Σ with oracle access to $\mathsf{Mac}_k(\cdot)$ and $\mathsf{Ver}_k(\cdot)$, outputs a message/tag pair (m, σ) such that (1) $\mathsf{Ver}_k(m, \sigma) = 1$ and (2) σ was not previously output by the oracle $\mathsf{Mac}_k(\cdot)$ as a MAC on the message m. We say that the MAC scheme Σ is secure if $\mathsf{Adv}_\Sigma(\mathcal{A})$ is negligible for every PPT adversary $\mathcal{A}$. We use $\mathsf{Adv}_\Sigma(t, q_{\mathrm{mac}}, q_{\mathrm{ver}})$ to denote the maximum value of $\mathsf{Adv}_\Sigma(\mathcal{A})$ over all PPT adversaries $\mathcal{A}$ running in time at most t and asking at most q_{mac} and q_{ver} queries to $\mathsf{Mac}_k(\cdot)$ and $\mathsf{Ver}_k(\cdot)$, respectively.

2-Party PAKE Protocols. H3PAKE takes as input a 2-party PAKE protocol 2PAKE. We assume that the given 2-party protocol 2PAKE outputs session keys distributed in $\{0,1\}^\ell$ and is SK-secure against an adversary who is given access to all the oracles: Send, $\mathsf{Execute}$, Reveal, $\mathsf{Corrupt}$, and Test. Let $\mathsf{Adv}_{\mathsf{2PAKE}}(\mathcal{A})$ be the advantage of an adversary $\mathcal{A}$ in breaking the SK security of 2PAKE. We require that, for any PPT adversary $\mathcal{A}$ asking at most q_{send} Send queries, $\mathsf{Adv}_{\mathsf{2PAKE}}(\mathcal{A})$ is only negligibly larger than $q_{\mathrm{send}}/|\mathcal{D}|$. $\mathsf{Adv}_{\mathsf{2PAKE}}(t, Q)$ denotes the maximum value of $\mathsf{Adv}_{\mathsf{2PAKE}}(\mathcal{A})$ over all PPT adversaries $\mathcal{A}$ with time complexity at most t and query complexity at most Q.

Additionally, H3PAKE uses a cryptographic hash function H mapping $\{0,1\}^*$ to $\{0,1\}^\kappa$, where κ is a security parameter representing the length of session keys. H is modelled as a random oracle in our proof of security for H3PAKE.

4.2. Description of H3PAKE. We assume that the following information has been preestablished and is known to all users in the network: (1) a cyclic group $\mathbb{G}$ of prime order q and a generator g of $\mathbb{G}$, (2) a MAC scheme $\Sigma = (\mathsf{Gen}, \mathsf{Mac}, \mathsf{Ver})$, (3) a 2-party PAKE protocol 2PAKE, and (4) a cryptographic hash function H. These public parameters can be determined by the server and be broadcast to all its registered clients. Let A and B be two clients who wish to establish a session key, and let S be the trusted server with which A and B have registered their passwords pw_A and pw_B, respectively. We denote the partner identifier pid given as input to (an instance of) A (resp., B and S) by pid_A (resp., pid_B and pid_S). Recall that pid is a sequence of identities of protocol participants. The order of identities that appears in pid is of critical importance for the correctness of our construction and its security proof. For simplicity, we assume that $\mathsf{pid}_A = \mathsf{pid}_B = \mathsf{pid}_S = \langle A, B, S\rangle$. Figure 5 depicts how the generic 3-party PAKE protocol, H3PAKE, is constructed from any given 2-party protocol, 2PAKE. More specifically, H3PAKE is constructed as follows.

Phase 1. A and S establish a shared high-entropy key k_{AS} by running the 2-party protocol 2PAKE. Likewise, B and S establish a shared high-entropy key k_{BS}.

Phase 2. A and B establish their session key by running a MAC-based Diffie-Hellman key exchange protocol with assistance of S.

Step 1. A chooses a random $x \in \mathbb{Z}_q$, computes $X = g^x$ and $\sigma_{AS} = \mathsf{Mac}_{k_{AS}}(A\|X\|\mathsf{pid}_A)$, and sends $\langle A, X, \sigma_{AS}\rangle$ to S. Meanwhile, B chooses a random $y \in \mathbb{Z}_q$, computes $Y = g^y$ and $\sigma_{BS} = \mathsf{Mac}_{k_{BS}}(B\|Y\|\mathsf{pid}_B)$, and sends $\langle B, Y, \sigma_{BS}\rangle$ to S.

Step 2. S checks that $\mathsf{Ver}_{k_{AS}}(A\|X\|\mathsf{pid}_S, \sigma_{AS}) = 1$ and $\mathsf{Ver}_{k_{BS}}(B\|Y\|\mathsf{pid}_S, \sigma_{BS}) = 1$. If either verification fails, S aborts the protocol. Otherwise, S computes $\sigma_{SA} = \mathsf{Mac}_{k_{AS}}(S\|Y\|\mathsf{pid}_S)$ and $\sigma_{SB} = \mathsf{Mac}_{k_{BS}}(S\|X\|\mathsf{pid}_S)$ and sends $\langle S, Y, \sigma_{SA}\rangle$ and $\langle S, X, \sigma_{SB}\rangle$ to A and B, respectively.

Step 3. A verifies that $\mathsf{Ver}_{k_{AS}}(S\|Y\|\mathsf{pid}_A, \sigma_{SA}) = 1$. If the verification fails, A aborts the protocol. Otherwise, A sets the session identifier, $\mathsf{sid}_A = A\|X\|B\|Y$, and computes the Diffie-Hellman key, $K_A = Y^x$, and the session key, $\mathsf{sk}_A = H(\mathsf{pid}_A\|\mathsf{sid}_A\|K_A)$. Meanwhile, B checks if $\mathsf{Ver}_{k_{BS}}(S\|X\|\mathsf{pid}_B, \sigma_{SB}) = 1$ and aborts if the check fails. Otherwise, B sets $\mathsf{sid}_B = A\|X\|B\|Y$ and computes $K_B = X^y$ and $\mathsf{sk}_B = H(\mathsf{pid}_B\|\mathsf{sid}_B\|K_B)$.

At the end of the protocol execution, A and B will compute the same session key sk if they both hold the same sets of pid and sid and thus compute the same Diffie-Hellman key $K = g^{xy}$.

We do not require 2PAKE to be instantiated with a protocol that provides either unilateral or mutual authentication, as H3PAKE already provides mutual authentication between

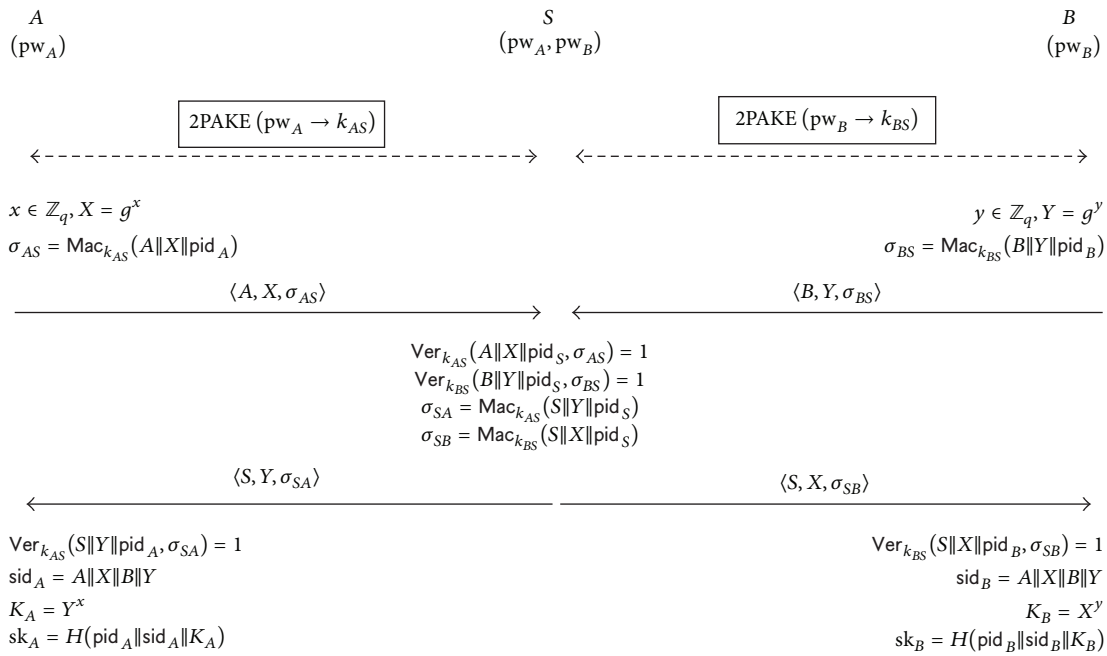

Figure 5: H3PAKE: our proposed generic 3-party PAKE protocol.

the server and the clients (via the MAC values exchanged in Phase 2). Hence, any 2-party protocol that provides implicit key authentication, including one-round protocols, will be suitable candidates to instantiate 2PAKE.

4.3. Proof of SK Security. We claim that the generic construction H3PAKE described in Figure 5 is SK-secure in the random oracle model under the GDH assumption in $\mathbb{G}$ and the security of the MAC scheme Σ.

Theorem 15. *Let H be a random oracle. Then, for any adversary with time complexity at most t and query complexity at most $Q = \langle q_{exec}, q_{send}, q_{reve}, q_{corr}, q_{test}\rangle$, its advantage in breaking the SK security of H3PAKE is bounded by*

$$\begin{aligned}\mathsf{Adv}_{H3PAKE}(t,Q) \le\ & 2\cdot \mathsf{Adv}_{2PAKE}\left(t', Q'\right)\\ & + 2\cdot q_{send}\cdot \mathsf{Adv}_{\Sigma}\left(t', 2, 2\right)\\ & + 2\cdot \mathsf{Adv}^{GDH}_{\mathbb{G}}\left(t'\right),\end{aligned} \tag{2}$$

where $Q' = \langle 2q_{exec}, q_{send}, q_{send}, q_{corr}, 2q_{exec} + q_{send}\rangle$ and t' is the maximum time required to perform the experiment $\mathbf{Exp}_0$ involving an adversary who attacks H3PAKE with time complexity t.

Proof. Assume a PPT adversary $\mathscr{A}$ who attacks H3PAKE with time complexity t and query complexity $Q = \langle q_{\text{exec}}, q_{\text{send}}, q_{\text{reve}}, q_{\text{corr}}, q_{\text{test}}\rangle$. We prove the theorem by making a series of modifications to the experiment $\mathbf{Exp}_0$, bounding the difference in $\mathscr{A}$'s advantage between two consecutive experiments, and ending up with an experiment in which $\mathscr{A}$'s advantage is negligible. By Succ_i, we denote the event that $\mathscr{A}$ correctly guesses the hidden bit b (chosen by the Test oracle) in experiment $\mathbf{Exp}_i$.

Before presenting the first modified experiment, we define the notion of a *clean* instance.

Definition 16. We say an instance Π^i_U is unclean if $\mathscr{A}$ has queried $\mathsf{Corrupt}(U')$ for some $U' \in \mathsf{pid}^i_U$. Otherwise, we say it is clean.

Experiment $\mathbf{Exp}_1$. This experiment is different from $\mathbf{Exp}_0$ only in that we replace each different MAC key with a random key drawn uniformly from $\{0,1\}^{\ell}$ for all clean instances. The difference in $\mathscr{A}$'s advantage between $\mathbf{Exp}_0$ and $\mathbf{Exp}_1$ is bounded by the following lemma.

Lemma 17. $|\Pr_{H3PAKE,\mathscr{A}}[\mathsf{Succ}_1] - \Pr_{H3PAKE,\mathscr{A}}[\mathsf{Succ}_0]| \le \mathsf{Adv}_{2PAKE}(t', Q')$, *where t' and Q' are as defined in Theorem 15.*

Proof. We prove the lemma by constructing an adversary $\mathscr{A}'$ attacking protocol 2PAKE from the adversary $\mathscr{A}$ whose advantage in attacking H3PAKE is different between $\mathbf{Exp}_0$ and $\mathbf{Exp}_1$. $\mathscr{A}'$ begins by choosing a bit b uniformly at random. Then, $\mathscr{A}'$ runs $\mathscr{A}$ as a subroutine while simulating the oracles as follows.

Execute *Queries*. When $\mathscr{A}$ makes an $\mathsf{Execute}(\Pi^i_A, \Pi^j_B, \Pi^k_S)$ query, $\mathscr{A}'$ first checks if any of A, B, and S was previously corrupted.

(i) If so, $\mathscr{A}'$ answers the Execute query as in experiment $\mathbf{Exp}_0$.

(ii) Otherwise, $\mathscr{A}'$ answers the query using its own oracles. $\mathscr{A}'$ first asks two queries $\mathsf{Execute}(\Pi^i_A, \Pi^k_S)$

and $\mathsf{Execute}(\Pi_B^j, \Pi_S^{k'})$. Let $\mathsf{T}_{\mathsf{2PAKE}}$ and $\mathsf{T}'_{\mathsf{2PAKE}}$ be two transcripts returned in response to the $\mathsf{Execute}$ queries. Next, $\mathscr{A}'$ makes the queries $\mathsf{Test}(\Pi_A^i)$ and $\mathsf{Test}(\Pi_B^j)$ and receives in return two keys $\overline{k}_{AS}$ and $\overline{k}_{BS}$ (either real or random). $\mathscr{A}'$ then generates the rest of the protocol transcript (i.e., the messages to be sent in Phase 2), using $\overline{k}_{AS}$ and $\overline{k}_{BS}$ as the MAC keys. Finally, $\mathscr{A}'$ returns these messages together with $\mathsf{T}_{\mathsf{2PAKE}}$ and $\mathsf{T}'_{\mathsf{2PAKE}}$ after ordering them properly.

Send *Queries*. Whenever $\mathscr{A}$ makes a $\mathsf{Send}(\Pi_U^i, m)$ query, $\mathscr{A}'$ checks if m is a message for initiating a new session (of H3PAKE) or the Send query belongs to an execution of 2PAKE.

(1) If both conditions are untrue, $\mathscr{A}'$ responds to the query as in experiment $\mathbf{Exp}_0$.

(2) Otherwise, $\mathscr{A}'$ answers it by making the same query to its own Send oracle. If the query prompts Π_U^i to accept, then $\mathscr{A}'$ checks if anyone in pid_U^i was previously corrupted.

   (a) If so, $\mathscr{A}'$ makes a $\mathsf{Reveal}(\Pi_U^i)$ query and uses the output of this Reveal query as the MAC key of Π_U^i.

   (b) Otherwise, $\mathscr{A}'$ makes a $\mathsf{Test}(\Pi_U^i)$ query (unless the partner of Π_U^i has already been tested) and uses the output of this Test query as the MAC key of Π_U^i.

Reveal *Queries*. These queries are handled as in experiment $\mathbf{Exp}_0$.

$\mathsf{Corrupt}$ *Queries*. $\mathscr{A}'$ answers these queries in the straightforward way using its own $\mathsf{Corrupt}$ oracle.

Test Queries. $\mathscr{A}'$ answers these queries according to the bit b that it has chosen at the beginning of the simulation. That is, $\mathscr{A}'$ returns real session keys, which it has computed on its own, if $b = 1$, and otherwise returns random keys chosen uniformly at random from $\{0,1\}^\kappa$.

At some point in time, $\mathscr{A}$ will terminate and output its guess b'. When this happens, $\mathscr{A}'$ outputs 1, if $b = b'$, and 0 otherwise.

From the simulation, it is obvious that

(i) the probability that $\mathscr{A}'$ outputs 1 when its Test oracle returns real session keys is equal to the probability that $\mathscr{A}$ correctly guesses the bit b in experiment $\mathbf{Exp}_0$;

(ii) the probability that $\mathscr{A}'$ outputs 1 when its Test oracle returns random keys is equal to the probability that $\mathscr{A}$ correctly guesses the bit b in experiment $\mathbf{Exp}_1$.

This means that $\mathsf{Adv}_{\mathsf{2PAKE}}(\mathscr{A}') = |\Pr_{\mathsf{H3PAKE},\mathscr{A}}[\mathsf{Succ}_1] - \Pr_{\mathsf{H3PAKE},\mathscr{A}}[\mathsf{Succ}_0]|$. Since $\mathscr{A}'$ has at most time complexity t' and query complexity $Q' = \langle 2q_{\text{exec}}, q_{\text{send}}, q_{\text{send}}, q_{\text{corr}}, 2q_{\text{exec}} + q_{\text{send}}\rangle$, we obtain Lemma 17. □

Experiment $\mathbf{Exp}_2$. Let Forge be the event that the adversary $\mathscr{A}$ makes a Send query of the form $\mathsf{Send}(\Pi_U^i, V\|*\|\sigma)$ before querying $\mathsf{Corrupt}(U)$ and $\mathsf{Corrupt}(V)$, where σ is a valid tag on $V\| * \|\mathsf{pid}_U^i$ and was not output by a previous oracle query as a tag on $V\| * \|\mathsf{pid}_U^i$. Then $\mathbf{Exp}_2$ is different from $\mathbf{Exp}_1$ only in that, if Forge occurs, the experiment is aborted and the adversary does not succeed. We claim the following lemma.

Lemma 18. $|\Pr_{H3PAKE,\mathscr{A}}[\mathsf{Succ}_2] - \Pr_{H3PAKE,\mathscr{A}}[\mathsf{Succ}_1]| \leq q_{send} \cdot \mathsf{Adv}_\Sigma(t', 2, 2)$, *where* t' *is as defined in Theorem 15.*

Proof. Given the adversary $\mathscr{A}$ attacking H3PAKE and assuming that the event Forge occurs, we construct an algorithm $\mathscr{F}$ that outputs, with a nonnegligible probability, a forgery against the MAC scheme Σ. The algorithm $\mathscr{F}$ is given oracle access to $\mathsf{Mac}_k(\cdot)$ and $\mathsf{Ver}_k(\cdot)$. The goal of $\mathscr{F}$ is to produce a message/tag pair (m, σ) such that (1) $\mathsf{Ver}_k(m, \sigma) = 1$ and (2) σ was not previously output by the $\mathsf{Mac}_k(\cdot)$ oracle on input m.

Let n be the number of all different MAC keys that are established via a Send query of $\mathscr{A}$. Clearly, $n \leq q_{\text{send}}$. $\mathscr{F}$ begins by choosing a random $\alpha \in \{1, \ldots, n\}$. Let k_α denote the αth key among all the n MAC keys, and let Send_α be a Send query that should be answered and/or verified using k_α. $\mathscr{F}$ invokes $\mathscr{A}$ as a subroutine and handles the oracle calls of $\mathscr{A}$ as in experiment $\mathbf{Exp}_1$ except that it answers all Send_α queries by accessing its MAC generation and verification oracles. As a result, the αth MAC key k_α is never used during the simulation. If Forge occurs against an instance that holds k_α, $\mathscr{F}$ halts and outputs the message/tag pair generated by $\mathscr{A}$ as its forgery. Otherwise, $\mathscr{F}$ halts and outputs a failure indication.

If the guess α is correct, then the simulation is perfect and $\mathscr{F}$ achieves its goal. Namely, $\mathsf{Adv}_\Sigma(\mathscr{F}) = \Pr[\mathsf{Forge}]/n$. Since $n \leq q_{\text{send}}$, we get $\Pr[\mathsf{Forge}] \leq q_{\text{send}} \cdot \mathsf{Adv}_\Sigma(\mathscr{F})$. Then, Lemma 18 follows by noticing that $\mathscr{F}$ has at most time complexity t' and makes at most two queries to $\mathsf{Mac}_k(\cdot)$ and $\mathsf{Ver}_k(\cdot)$. □

Experiment $\mathbf{Exp}_3$. This experiment is different from experiment $\mathbf{Exp}_2$ only in that the $\mathsf{Execute}$ and Send oracles are simulated as in "the $\mathbf{Exp}_3$ modification" described in Box 2.

Since the view of $\mathscr{A}$ is identical between $\mathbf{Exp}_2$ and $\mathbf{Exp}_3$, following Lemma 19 is clear.

Lemma 19. $\Pr_{H3PAKE,\mathscr{A}}[\mathsf{Succ}_3] = \Pr_{H3PAKE,\mathscr{A}}[\mathsf{Succ}_2]$.

In experiment $\mathbf{Exp}_3$, the advantage of $\mathscr{A}$ in attacking H3PAKE is bounded by the following lemma.

Lemma 20. $\Pr_{H3PAKE,\mathscr{A}}[\mathsf{Succ}_3] \leq (1/2) + \mathsf{Adv}_{\mathbb{G}}^{GDH}(t')$ *where* t' *is as defined in Theorem 15.*

Proof. The proof is via a reduction from the GDH problem which is believed to be hard. Assume that the advantage of $\mathscr{A}$ in attacking H3PAKE is nonnegligible. Then we can construct

The **Exp$_3$** Modification
When $\mathcal{A}$ asks an Execute or Send query, the simulator answers it exactly as in experiment **Exp$_2$**, except that it modifies the way of generating the public Diffie-Hellman values (denoted as X and Y in the protocol) as follows:

(i) The simulator chooses two random $v_1, v_2 \in \mathbb{Z}_q$ and computes $V_1 = g^{v_1}$ and $V_2 = g^{v_2}$.

(ii) For each instance Π_C^i, the simulator chooses a random $r \in \mathbb{Z}_q$, computes

$$R = \begin{cases} V_1^{\,r} & \text{if } C \text{ appears first in } \mathsf{pid}_C^i \\ V_2^{\,r} & \text{if } C \text{ appears second in } \mathsf{pid}_C^i, \end{cases}$$

and uses R as the public Diffie-Hellman value (i.e., as X or Y) of Π_C^i.

Box 2

Deciding $m \stackrel{?}{=} \mathrm{kds}_C^i$
Given a string m and a tuple $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, *, R', *)$, $\mathcal{A}_{\mathrm{GDH}}$ first checks if the bit-length of m is equal to the bit-length of key derivation strings. If it is, then $\mathcal{A}_{\mathrm{GDH}}$ checks that (1) $\mathsf{pid}_C^i \| \mathsf{sid}_C^i$ is a prefix of m and (2) $\mathcal{O}(R, R', m_{\mathbb{G}}) = 1$ where $m_{\mathbb{G}}$ is a $|\mathbb{G}|$-bit string that is a suffix of m. If both are true, then $m = \mathrm{kds}_C^i$.

Box 3

an algorithm $\mathcal{A}_{\mathrm{GDH}}$ that has a nonnegligible advantage in solving the GDH problem in $\mathbb{G}$. The goal of $\mathcal{A}_{\mathrm{GDH}}$ is to compute and output the value $W_3 = g^{w_1 w_2} \in \mathbb{G}$ when given a CDH-problem instance $(W_1 = g^{w_1}, W_2 = g^{w_2}) \in \mathbb{G}$ as well as an oracle $\mathcal{O}(\cdot, \cdot, \cdot)$ that solves the DDH problem in $\mathbb{G}$. $\mathcal{A}_{\mathrm{GDH}}$ runs $\mathcal{A}$ as a subroutine while simulating all the oracles on its own.

When $\mathcal{A}$ asks an Execute and Send query, $\mathcal{A}_{\mathrm{GDH}}$ answers it as specified in the **Exp$_3$** modification but using W_1 and W_2 instead of V_1 and V_2. In this way, $\mathcal{A}_{\mathrm{GDH}}$ can embed the CDH-problem instance (W_1, W_2) into all protocol sessions. Accordingly, $\mathcal{A}_{\mathrm{GDH}}$ can compute no session keys but can still correctly answer Reveal queries by storing all the keying materials associated with each instance. For each instance Π_C^i whose only remaining work is to compute its session key, $\mathcal{A}_{\mathrm{GDH}}$ checks if the instance Π_C^i is clean or unclean. If it is clean, $\mathcal{A}_{\mathrm{GDH}}$ stores a tuple $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, r, R', r')$ into a list, which we denote as CDHList, where $R = W_1^{\,r}$ and $R' = W_2^{\,r'}$. Here, the exponent r (resp., r') is the one chosen for the instance whose user identity comes first (resp., second) in pid_C^i. If it is unclean, $\mathcal{A}_{\mathrm{GDH}}$ stores a tuple $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, r, R', \perp)$ if C comes first in pid_C^i or a tuple $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, \perp, R', r')$ if C comes second in pid_C^i. Here, $\perp$ indicates that the exponent of the received public Diffie-Hellman value may have been chosen by $\mathcal{A}$.

While imbedding the CDH-problem instance as above, $\mathcal{A}_{\mathrm{GDH}}$ has to provide $\mathcal{A}$ with the same view as in experiment **Exp$_3$**. To this end, let *kds* be a *key derivation string* from which a session key is computed by applying the random oracle H. Let kds_C^i denote the *kds* of instance Π_C^i. Then, $kds_C^i = \mathsf{pid}_C^i \| \mathsf{sid}_C^i \| K_C^i$. As is clear from the above simulation, $\mathcal{A}_{\mathrm{GDH}}$ cannot compute any *kds* on its own. But, given a string m, $\mathcal{A}_{\mathrm{GDH}}$ can determine whether m is the *kds* of some instance Π_C^i or not by repeatedly performing the deciding operation for the tuples in CDHList as in Box 3.

The simulation of other oracles is provided as follows.

H Queries. $\mathcal{A}_{\mathrm{GDH}}$ uses a list, HList, to maintain input-output pairs of H. For each H query on a string m, $\mathcal{A}_{\mathrm{GDH}}$ first checks if an entry of the form (m, h) is in HList. If it is, $\mathcal{A}_{\mathrm{GDH}}$ returns h to $\mathcal{A}$. Otherwise, $\mathcal{A}_{\mathrm{GDH}}$ checks if m is the *kds* of some instance Π_C^i by repeatedly performing the deciding operation above until a match is found.

(i) If a match is found and the corresponding tuple is of the form $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, r, R', r')$, $\mathcal{A}_{\mathrm{GDH}}$ computes $W_3 = (m_{\mathbb{G}})^{1/rr'}$ and terminates outputting W_3. In this case, $\mathcal{A}_{\mathrm{GDH}}$ succeeds in solving the GDH problem.

(ii) If a match is found and the corresponding tuple is of the form $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, r, R', \perp)$ or $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, \perp, R', r')$, $\mathcal{A}_{\mathrm{GDH}}$ checks if a tuple of the form $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, R', \mathrm{sk})$ is in the RList which is maintained by $\mathcal{A}_{\mathrm{GDH}}$ to store revealed session keys. If it is, $\mathcal{A}_{\mathrm{GDH}}$ returns sk to $\mathcal{A}$ and adds (m, sk) to HList. Otherwise, $\mathcal{A}_{\mathrm{GDH}}$ returns a random κ-bit string str to $\mathcal{A}$ and adds (m, str) to HList.

(iii) Otherwise, $\mathcal{A}_{\mathrm{GDH}}$ returns a random κ-bit string str to $\mathcal{A}$ and adds (m, str) to HList.

Reveal *Queries*. When $\mathcal{A}$ asks a Reveal(Π_C^i) query, $\mathcal{A}_{\mathrm{GDH}}$ finds a tuple of the form $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, *, R', *)$ in CDHList and checks if a tuple of the form $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, R', \mathrm{sk})$ is in the RList. If it is, $\mathcal{A}_{\mathrm{GDH}}$ returns sk to $\mathcal{A}$. Otherwise, $\mathcal{A}_{\mathrm{GDH}}$ checks if HList contains an entry (m, h) such that $m = kds_C^i$. Given the tuple $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, *, R', *)$, this check can be done by performing the deciding operation for all entries in HList. If such entry (m, h) exists in HList, $\mathcal{A}_{\mathrm{GDH}}$ returns h in response to the query and adds the tuple $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, R', h)$ into

RList. Otherwise, $\mathcal{A}_{\mathrm{GDH}}$ returns a random κ-bit string str to $\mathcal{A}$ and adds the tuple $(\mathsf{pid}_C^i, \mathsf{sid}_C^i, R, R', str)$ into RList.

Corrupt *Queries*. $\mathcal{A}_{\mathrm{GDH}}$ answers these queries in the obvious way.

Test *Queries*. For each of these queries, $\mathcal{A}_{\mathrm{GDH}}$ responses with a random κ-bit string.

Let Ask be the event that $\mathcal{A}$ makes an H query on a string m that is the kds of some fresh instance. From the simulation of H, it can be easily seen that as soon as Ask occurs, $\mathcal{A}_{\mathrm{GDH}}$ outputs the desired result $W_3 = g^{w_1 w_2}$ and thus succeeds in solving the GDH problem in $\mathbb{G}$. But, since H is a random oracle, $\mathcal{A}$ gains no advantage in distinguishing the test keys from random if the event Ask does not occur. This implies the assertion of Lemma 20.

This result combined with Lemmas 17–19 concludes the proof for Theorem 15. □

4.4. Proof of Resistance to Undetectable Online Dictionary Attacks. We now claim that H3PAKE is secure against a UD online dictionary attack as long as the given 2-party protocol 2PAKE is SK-secure.

Theorem 21. *Assume that, for any PPT adversary $\mathcal{A}'$ asking at most q_{send} Send queries, $\mathsf{Adv}_{2PAKE}(\mathcal{A}')$ is only negligibly larger than $q_{send}/|\mathcal{D}|$. Then, for any PPT adversary $\mathcal{A}$ asking at most q_{send} Send queries, $\Pr_{H3PAKE,\mathcal{A}}[\mathsf{Undet}]$ is only negligibly larger than $q_{send}/|\mathcal{D}|$, where Undet is as defined in Section 3.3.*

Proof. Let $\mathcal{A}$ be an adversary who asks q_{send} Send queries in attacking the protocol H3PAKE. Assume that $\Pr_{\mathsf{H3PAKE},\mathcal{A}}[\mathsf{Undet}]$ is nonnegligibly larger than $q_{\mathrm{send}}/|\mathcal{D}|$. Given the adversary $\mathcal{A}$, we prove the theorem by constructing an adversary $\mathcal{A}'$ against 2PAKE who asks at most q_{send} Send queries but has an advantage nonnegligibly larger than $q_{\mathrm{send}}/|\mathcal{D}|$.

$\mathcal{A}'$ invokes $\mathcal{A}$ as a subroutine and answers the oracle queries of $\mathcal{A}$ as follows.

Execute *Queries*. When $\mathcal{A}$ makes an $\mathsf{Execute}(\Pi_A^i, \Pi_B^j, \Pi_S^k)$ query, $\mathcal{A}'$ answers the query using its own Execute and Reveal oracles. $\mathcal{A}'$ first queries $\mathsf{Execute}(\Pi_A^i, \Pi_S^k)$ and $\mathsf{Execute}(\Pi_B^j, \Pi_S^{k'})$. Let $\mathsf{T}_{\mathsf{2PAKE}}$ and $\mathsf{T}'_{\mathsf{2PAKE}}$ be two transcripts returned in response to the Execute queries. Next, $\mathcal{A}'$ obtains two keys k_{AS} and k_{BS} by querying $\mathsf{Reveal}(\Pi_A^i)$ and $\mathsf{Reveal}(\Pi_B^j)$. $\mathcal{A}'$ then generates the rest of the protocol transcript, using k_{AS} and k_{BS} as the MAC keys. Finally, $\mathcal{A}'$ returns these messages together with $\mathsf{T}_{\mathsf{2PAKE}}$ and $\mathsf{T}'_{\mathsf{2PAKE}}$ after ordering them properly.

Send *Queries*. When $\mathcal{A}$ makes a $\mathsf{Send}(\Pi_U^i, m)$ query, $\mathcal{A}'$ checks if m is a message for initiating a new session (of H3PAKE) or the Send query belongs to an execution of 2PAKE.

(1) If both conditions are untrue, $\mathcal{A}'$ responds to the query as that in the original experiment $\mathbf{Exp}_0$.

(2) Otherwise, $\mathcal{A}'$ answers it by making the same query to its own Send oracle. If the query prompts Π_U^i to accept, $\mathcal{A}'$ checks if Π_U^i is a server instance against which $\mathcal{A}$ has mounted an online dictionary attack. If not, $\mathcal{A}'$ makes a $\mathsf{Reveal}(\Pi_U^i)$ query (and later uses the output of this Reveal query as the MAC key of Π_U^i). (How to handle the other case will be explained below.)

Corrupt *Queries*. $\mathcal{A}'$ answers these queries using its own Corrupt oracle.

Reveal/Test Querise. $\mathcal{A}'$ answers these queries as in the original experiment $\mathbf{Exp}_0$.

Let Π_S^t be any server instance against which $\mathcal{A}$ has mounted an online dictionary attack. Let k_S^t be the session key that the instance Π_S^t has computed in its execution of 2PAKE. In order for the instance Π_S^t to terminate normally, the adversary $\mathcal{A}$ has to make a query of the form $\mathsf{Send}(\Pi_S^t, C\| * \|\sigma_{CS})$ such that $\mathsf{Ver}_{k_S^t}(C\| * \|\mathsf{pid}_S^t, \sigma_{CS}) = 1$. When $\mathcal{A}$ makes such a Send query (i.e., when the event Undet occurs), $\mathcal{A}'$ makes a Test query against the instance Π_S^t. Note that the instance Π_S^t is fresh as (1) it is partnered with no instance and (2) S and C must have not been corrupted. Let $\overline{k}_S^t$ be the key returned in response to the Test query. $\mathcal{A}'$ outputs 1, if $\mathsf{Ver}_{\overline{k}_S^t}(C\| * \|\mathsf{pid}_S^t, \sigma_{CS}) = 1$, and outputs 0, otherwise. If Undet does not occur, $\mathcal{A}'$ outputs a random bit.

From the simulation above, it is clear to see that

$$\begin{aligned}
&\mathsf{Adv}_{\mathsf{2PAKE}}(\mathcal{A}') \\
&\quad = 2 \cdot \Pr_{\mathsf{2PAKE},\mathcal{A}'}[\mathsf{Succ}] - 1 \\
&\quad = 2 \cdot \left(\Pr_{\mathsf{H3PAKE},\mathcal{A}}[\mathsf{Undet}] \right. \\
&\qquad \left. + \frac{1}{2}\left(1 - \Pr_{\mathsf{H3PAKE},\mathcal{A}}[\mathsf{Undet}]\right)\right) - 1 \\
&\quad = \Pr_{\mathsf{H3PAKE},\mathcal{A}}[\mathsf{Undet}].
\end{aligned} \tag{3}$$

Then, Theorem 21 immediately follows since the number of Send queries asked by $\mathcal{A}'$ against 2PAKE is at most q_{send}. □

5. Concluding Remarks

The undetectable online dictionary attacks we presented against the widely studied GPAKE protocol of Abdalla et al. [8, 9] are a reminder of the difficulty of designing a secure yet efficient 3-party PAKE protocol. The GPAKE protocol was proven secure in a model that does not capture undetectable online dictionary attacks, and thus, our attacks do not invalidate the proof of security for GPAKE.

We also presented a simple and intuitive approach of capturing all classes of dictionary attacks in the framework of the widely accepted Bellare-Pointcheval-Rogaway model.

What motivated our approach is the observation that no prior work has provided a rigorous formal treatment of insider (online/offline) dictionary attacks in the password-only 3-party setting and as a consequence 3-party PAKE protocols insecure against such attacks have proliferated. We believe that our approach provides protocol designers with an easier and more accessible way of proving security of their protocols against dictionary attacks.

Finally, we presented a generic 3-party PAKE protocol (H3PAKE) and proved its security in the random oracle model under the gap Diffie-Hellman assumption. To the best of our knowledge, H3PAKE is the first 3-party PAKE protocol proven secure against both insider and outsider dictionary attacks as well as offline and online dictionary attacks. Future work includes coming up with a 3-party PAKE protocol that achieves the same (or even better) level of security and efficiency as H3PAKE but does not rely its security proof on the random oracle model.

Conflict of Interests

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

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT, and Future Planning (2014R1A1A2002775).

References

[1] W. Wang and L. Hu, "Efficient and provably secure generic construction of three-party password-based authenticated key exchange protocols," in *Progress in Cryptology—INDOCRYPT 2006*, vol. 4329 of *Lecture Notes in Computer Science*, pp. 118–132, Springer, Berlin, Germany, 2006.

[2] H. Guo, Z. Li, Y. Mu, and X. Zhang, "Cryptanalysis of simple three-party key exchange protocol," *Computers and Security*, vol. 27, no. 1-2, pp. 16–21, 2008.

[3] J. Nam, J. Paik, H. Kang, U. M. Kim, and D. Won, "An off-line dictionary attack on a simple three-party key exchange protocol," *IEEE Communications Letters*, vol. 13, no. 3, pp. 205–207, 2009.

[4] E. Yoon and K. Yoo, "Cryptanalysis of a simple three-party password-based key exchange protocol," *International Journal of Communication Systems*, vol. 24, no. 4, pp. 532–542, 2011.

[5] J. Nam, K. K. R. Choo, M. Kim, J. Paik, and D. Won, "Dictionary attacks against password-based authenticated three-party key exchange protocols," *KSII Transactions on Internet and Information Systems*, vol. 7, no. 12, pp. 3244–3260, 2013.

[6] J. Nam, K. K. R. Choo, M. Park, J. Paik, and D. Won, "On the security of a simple three-party key exchange protocol without server's public keys," *The Scientific World Journal*, vol. 2014, Article ID 479534, 7 pages, 2014.

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

[8] M. Abdalla, P. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," in *Public Key Cryptography*, vol. 3386 of *Lecture Notes in Computer Science*, pp. 65–84, Springer, Berlin, Germany, 2005.

[9] M. Abdalla, P. Fouque, and D. Pointcheval, "Password-based authenticated key exchange in the three-party setting," *IEE Proceedings—Information Security*, vol. 153, no. 1, pp. 27–39, 2006.

[10] M. Abdalla and D. Pointcheval, "Interactive diffie-hellman assumptions with applications to password-based authentication," in *Proceedings of the 9th International Conference on Financial Cryptography and Data Security (FC '05)*, pp. 341–356, Springer, Berlin, Germany, March 2005.

[11] H. Wen, T. Lee, and T. Hwang, "Provably secure three-party password-based authenticated key exchange protocol using Weil pairing," *IEE Proceedings—Communications*, vol. 152, no. 2, pp. 138–143, 2005.

[12] E. Dongna, Q. Cheng, and C. Ma, "Password authenticated key exchange based on RSA in the three-party settings," in *Provable Security*, vol. 5848 of *Lecture Notes in Computer Science*, pp. 168–182, Springer, Berlin, Germany, 2009.

[13] T. Lee and T. Hwang, "Simple password-based three-party authenticated key exchange without server public keys," *Information Sciences*, vol. 180, no. 9, pp. 1702–1714, 2010.

[14] W. Wang, L. Hu, and Y. Li, "How to construct secure and efficient three-party password-based authenticated key exchange protocols," in *Information Security and Cryptology*, vol. 6584 of *Lecture Notes in Computer Science*, pp. 218–235, Springer, Berlin, Germany, 2011.

[15] C. Lin and T. Hwang, "On "a simple three-party password-based key exchange protocol"," *International Journal of Communication Systems*, vol. 24, no. 11, pp. 1520–1532, 2011.

[16] S. Wu, Q. Pu, S. Wang, and D. He, "Cryptanalysis of a communication-efficient three-party password authenticated key exchange protocol," *Information Sciences*, vol. 215, pp. 83–96, 2012.

[17] J. Nam, K. K. R. Choo, J. Kim, H. Kang, J. Paik, and D. Won, "Password-only authenticated three-party key exchange with provable security in the standard model," *The Scientific World Journal*, vol. 2014, Article ID 825072, 11 pages, 2014.

[18] J. Nam, Y. Lee, S. Kim, and D. Won, "Security weakness in a three-party pairing-based protocol for password authenticated key exchange," *Information Sciences*, vol. 177, no. 6, pp. 1364–1375, 2007.

[19] K. Yoneyama, "Efficient and strongly secure password-based server aided key exchange," in *Progress in Cryptology (INDOCRYPT '08)*, vol. 5365 of *Lecture Notes in Computer Science*, pp. 172–184, Springer, Berlin, Germany, 2008.

[20] J. Zhao and D. Gu, "Provably secure three-party password-based authenticated key exchange protocol," *Information Sciences*, vol. 184, no. 1, pp. 310–323, 2012.

[21] C. Lin, H. Sun, and T. Hwang, "Three-party encrypted key exchange: attacks and a solution," *ACM SIGOPS Operating Systems Review*, vol. 34, no. 4, pp. 12–20, 2000.

[22] H. Chien and T. Wu, "Provably secure password-based three-party key exchange with optimal message steps," *The Computer Journal*, vol. 52, no. 6, pp. 646–655, 2009.

[23] N. W. Lo and K. Yeh, "Cryptanalysis of two three-party encrypted key exchange protocols," *Computer Standards and Interfaces*, vol. 31, no. 6, pp. 1167–1174, 2009.

[24] D. Lou and H. Huang, "Efficient three-party password-based key exchange scheme," *International Journal of Communication Systems*, vol. 24, no. 4, pp. 504–512, 2011.

[25] J. Yang and T. Cao, "Provably secure three-party password authenticated key exchange protocol in the standard model," *Journal of Systems and Software*, vol. 85, no. 2, pp. 340–350, 2012.

[26] S. Wu, K. Chen, Q. Pu, and Y. Zhu, "Cryptanalysis and enhancements of efficient three-party password-based key exchange scheme," *International Journal of Communication Systems*, vol. 26, no. 5, pp. 674–686, 2013.

[27] H. Tsai and C. Chang, "Provably secure three party encrypted key exchange scheme with explicit authentication," *Information Sciences*, vol. 238, pp. 242–249, 2013.

[28] M. Bellare, D. Pointcheval, and P. Rogaway, "Authenticated key exchange secure against dictionary attacks," in *Advances in Cryptology—EUROCRYPT 2000*, vol. 1807 of *Lecture Notes in Computer Science*, pp. 139–155, Springer, Berlin, Germany, 2000.

[29] J. Katz, R. Ostrovsky, and M. Yung, "Efficient password-authenticated key exchange using human-memorable passwords," in *Advances in Cryptology—EUROCRYPT 2001*, vol. 2045 of *Lecture Notes in Computer Science*, pp. 475–494, Springer, Berlin, Germany, 2001.

[30] J. Katz, R. Ostrovsky, and M. Yung, "Efficient and secure authenticated key exchange using weak passwords," *Journal of the ACM*, vol. 57, no. 1, article 3, 39 pages, 2010.

[31] R. Gennaro and Y. Lindell, "A framework for password-based authenticated key exchange," in *Advances in Cryptology—EUROCRYPT 2003*, vol. 2656 of *Lecture Notes in Computer Science*, pp. 524–543, Springer, Berlin, Germany, 2003.

[32] P. MacKenzie, "The PAK suite: protocols for password-authenticated key exchange," Contributions to IEEE P1363.2, 2002.

[33] S. M. Bellovin and M. Merritt, "Encrypted key exchange: password-based protocols secure against dictionary attacks," in *Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy*, pp. 72–84, Oakland, Calif, USA, May 1992.

[34] E. Bresson, O. Chevassut, and D. Pointcheval, "New security results on encrypted key exchange," in *Public Key Cryptography*, vol. 2947 of *Lecture Notes in Computer Science*, pp. 145–158, Springer, Berlin, Germany, 2004.

[35] M. Bellare and P. Rogaway, "Provably secure session key distribution—the three party case," in *Proceedings of the ACM Symposium on Theory of Computing*, pp. 57–66, 1995.

[36] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in *Advances in Cryptology—CRYPTO '96*, vol. 1109 of *Lecture Notes in Computer Science*, pp. 1–15, Springer, Berlin, Germany, 1996.

[37] M. Abdalla and D. Pointcheval, "Simple password-based encrypted key exchange protocols," in *The Cryptographers' Track at the RSA Conference*, vol. 3376 of *Lecture Notes in Computer Science*, pp. 191–208, Springer, Berlin, Germany, 2005.

[38] K. K. R. Choo, "A proof of revised Yahalom protocol in the Bellare and Rogaway (1993) model," *Computer Journal*, vol. 50, no. 5, pp. 591–601, 2007.