EKE Meets Tight Security in the Universally Composable Framework

Xiangyu Liu[1,2], Shengli Liu[1,2,3(✉)], Shuai Han[1,2], and Dawu Gu[1]

[1] School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China
{xiangyu_liu,slliu,dalen17,dwgu}@sjtu.edu.cn
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
[3] Westone Cryptologic Research Center, Beijing 100070, China

Abstract. (Asymmetric) Password-based Authenticated Key Exchange ((a)PAKE) protocols allow two parties establish a session key with a pre-shared low-entropy password. In this paper, we show how Encrypted Key Exchange (EKE) compiler [Bellovin and Merritt, S&P 1992] meets tight security in the Universally Composable (UC) framework. We propose a strong 2DH variant of EKE, denoted by 2DH-EKE, and prove its tight security in the UC framework based on the CDH assumption. The efficiency of 2DH-EKE is comparable to the original EKE, with only $O(\lambda)$ bits growth in communication (λ the security parameter), and two (resp., one) extra exponentiation in computation for client (resp., server).

We also develop an asymmetric PAKE scheme 2DH-aEKE from 2DH-EKE. The security reduction loss of 2DH-aEKE is N, the total number of client-server pairs. With a meta-reduction, we formally prove that such a factor N is inevitable in aPAKE. Namely, our 2DH-aEKE meets the optimal security loss. As a byproduct, we further apply our technique to PAKE protocols like SPAKE2 and PPK in the relaxed UC framework, resulting in their 2DH variants with tight security from the CDH assumption.

Keywords: (Asymmetric) PAKE · UC Framework · Tight Security

1 Introduction

Password-based Authenticated Key Exchange (PAKE) [8] allows two parties (client and server) who share a low-entropy password pw to agree on a session key via public networks. Such session keys can later be used to establish secure channels. Different from authenticated key exchange (AKE) which needs a PKI to authenticate the validity of public keys, PAKE takes short human-memorizable passwords rather than long cryptographic keys. Therefore, PAKE is more convenient for deployments and applications.

For PAKE, the server has to store all clients' passwords and once compromised, all clients are in high risk. Asymmetric PAKE (aPAKE) [9, 19] is a variant of PAKE that considers security against server compromise. In the scenario of aPAKE, the server stores a password file (usually a hash value $H(\mathsf{pw})$) for the client, rather than a plain password. A client can establish a session key with a server if it holds a pre-image of the password file.

Started from the pioneering works by Belloven and Merritt [8, 9], (a)PAKE has been studied extensively, and a variety of protocols have been proposed over the past decades. For example, SPEKE [28], PPK/PAK [35], SPAKE2 [4], Dragonfly [24], J-PAKE [23], KOY [31], KV [32] for PAKE, and VB-PAKE [33], OPAQUE [30], KC-SPAKE2+ [41], KHAPE [21], YLZT [44], aEKE and OKAPE [39] for aPAKE. Among these protocols, SPAKE2, J-PAKE, OPAQUE are under the process of standardization [40, 5, 27, 38]. (a)PAKE protocols have also been increasingly applied to numerous settings, including TLS [30, 37], ad hoc networks [43], and the Internet of Things [42].

Since passwords have limited entropy, an adversary $\mathcal{A}$ can always try a password guess and actively engage in a session, and hence break the security with a noticeable probability. Such online attacks are inherit to (a)PAKE, but we can still fence these attacks via engineering methods, e.g., by limiting the number of online password guesses. Another type of attacks is offline dictionary attacks, i.e., the adversary eavesdrops on executions of the protocol and tries to break the security via a brute-force attack with all possible passwords in a given dictionary. Intuitively, a PAKE protocol is secure, if offline dictionary attacks help nothing to the adversary, and the only feasible way to break the security, is to engage in an online attack. In aPAKE, we further consider security when the server is compromised. That is, the password files help nothing for the adversary in impersonating a client, as long as $\mathcal{A}$ does not obtain the correct password from the compromised password file via brute-force search.

Security models for (a)PAKE. There are two types of security notions for (a)PAKE, namely, the game-based security in the Indistinguishability (IND) model (see [7] for PAKE and [33, 10, 11] for aPAKE) and the simulation-based security in the Universally Composable (UC) framework (see [15] for PAKE and [19] for aPAKE). The IND model is formalized as an experiment between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. We say an (a)PAKE protocol is secure in this model, if $\mathcal{A}$ cannot distinguish a real session key from a random session key, after it implements a variety of attacks.

The UC framework/model is another popular approach to formalize the security of (a)PAKE. In the UC framework, an ideal function $\mathcal{F}$ is defined to capture the essential functionality of an (a)PAKE protocol in the ideal world. We say that an (a)PAKE protocol is secure in the UC framework, if it securely emulates $\mathcal{F}$, i.e., no PPT environment can distinguish the real world execution from the ideal world execution (involving $\mathcal{F}$ and an ideal world simulator).

The UC framework is preferable to the IND model in a number of important aspects.

– The UC framework allows an arbitrary correlation and distribution for passwords. But in the IND model, passwords are required to be uniformly distributed over the password set (or at least have a min-entropy) for the sake of security proofs, e.g., [7, 31].

– UC security is preserved even if the protocol is running in arbitrary networks, where multiple different protocols may run concurrently. This is guaranteed by the universal composition theorem [14] in the UC framework.
– PAKE with UC security implies simulation-based security of secure-channel protocols built on PAKE [15]. In contrast, it is not sure for the IND security [41].

Tight security. The security of (a)PAKE (in both the IND and UC models) is achieved by a security reduction under proper assumptions. The security reduction transforms the ability of a successful adversary $\mathcal{A}$ to an algorithm $\mathcal{B}$ solving some well-known hard problem in about the same running time. If $\mathcal{A}$'s attack succeeds with probability ϵ, then $\mathcal{B}$ solves the problem with probability ϵ/L. Here L is defined as the security loss factor. We say that the reduction is tight if L is a constant. Otherwise the reduction is loose. A loose factor L is generally a polynomial of Q, where Q is the total number of queries involved by $\mathcal{A}$, and it can be of arbitrary polynomial. PAKE and aPAKE are generally implemented in the multi-user and multi-challenge setting. With a loose security reduction, the deployment of (a)PAKE has to choose a larger security parameter to compensate the loss factor L, resulting in larger elements and slower computations in the execution of (a)PAKE. Therefore, pursuing tight security of (a)PAKE is not only of theoretical value but also of practical significance.

There are very few works considering tight security of (a)PAKE. Becerra et al. [6] proved that the security of the PAK protocol [35] can be tightly reduced to the Gap DH assumption in the IND model. Under the same assumption, Abdalla et al. [1] proved that SPAKE2 [4] is tightly secure in the relaxed UC framework. However, both of the works used the non-standard Gap DH assumption, which states that it is hard to compute g^{xy}, given g^x, g^y, and an oracle deciding whether the input (g^a, g^b, g^c) is a DDH tuple. Besides, their securities are proved in the IND or relaxed UC model [1], rather than the (regular) UC framework. Up to now, there is no research on (a)PAKE with tight security in the UC framework.

Therefore, a challenging question is:

Can we construct a tightly secure (a)PAKE protocol in the UC framework, preferably from the standard assumption?

Our contributions. In this paper, we aim to answer the above question. For PAKE, we propose a tightly secure PAKE protocol based on the CDH assumption in the UC framework, and hence answer the question for PAKE in affirmative. For aPAKE, we prove a negative result via a meta-reduction, showing that a loss factor $L = N$ (the number of client-server pairs) is inevitable in aPAKE. Nevertheless, we still come up with an aPAKE protocol that meets this optimal security loss. In more detail, we revisit the EKE compiler/protocol in [8], and make the following contributions.

1. We propose a strong 2DH variant of EKE, denoted by 2DH-EKE, and prove that it is a tightly secure PAKE from the CDH assumption in the UC framework. The efficiency of 2DH-EKE is comparable to the original EKE, with

only $O(\lambda)$ bits growth in communication (λ the security parameter) and two (resp., one) extra exponentiation in computation for client (resp., server).
2. We show a negative result for aPAKE, indicating that it is impossible for aPAKE to be tightly secure. With a meta-reduction, we prove that the security loss of aPAKE is lower bounded by N, the number of client-server pairs.
3. We develop our 2DH-EKE to an aPAKE protocol, denoted by 2DH-aEKE, that meets the optimal security loss N based on the CDH assumption. Compared with 2DH-EKE, the 2DH-aEKE protocol adds one extra round for message authentications.
4. As a byproduct, we further apply our technique to PAKE protocols like SPAKE2 [4] and PPK [35] in the relaxed UC framework [1], resulting in their 2DH variants with tight security from the CDH assumption.

Related works. Bellovin and Merritt started the research of PAKE in [8], and proposed the well-known EKE compiler/protocol. The security of EKE was formally proved later by Bellare et al. [7] in the IND model, and by Dupont et al. [17] in the UC framework. Most of the efficient PAKE constructions ([35, 13, 4, 12, 36], to name a few) rely on Random Oracles (RO), and they can be viewed as different variants of the classical EKE compiler [8]. There are some works [31, 18, 20, 32] that consider PAKE in the standard model (i.e., without any ideal functions), but the constructions usually rely on heavy building blocks like CCA2-secure PKE [20] or NIZK [32], and hence are less efficient.

Given the advantages of the UC framework over the IND model, a large amount of (a)PAKE protocols [41, 30, 21, 39] are proposed and proved in the UC framework recently. There are some other works [3, 2] focusing on the existing IND-secure (a)PAKE schemes and aiming to prove their security in the stronger UC framework. In [1], Abdalla et al. relaxed the UC framework by introducing a modified lazy-extraction PAKE functionality, which allows the adversary in the ideal world to postpone its password guess until *after* the session is completed. Under this relaxed model, they proved that SPEKE [29], SPAKE2 [4], and TBPEKE [36] are UC-secure.

The only two works considering tight security of PAKE are [6] by Becerra et al., and [3] by Abdalla et al. (both of them are in the RO model). However, their securities are proved in the IND model or the relaxed UC framework [3], based on the non-standard Gap DH assumption. As far as we know, there exists no tightly secure (a)PAKE schemes in the regular UC framework up to now.

1.1 Technical Overview

In this subsection we briefly overview the technique used in this paper.

The main challenge to achieve tight security for (a)PAKE, is to embed the hard problem into *multiple* sessions, while keeping the ability to output their session keys in case the adversary $\mathcal{A}$ has the power to compute them (e.g., $\mathcal{A}$ correctly guesses the password). Furthermore, the reduction algorithm should extract (possibly from a set) the correct solution for the hard problem, if $\mathcal{A}$ wins the security experiment non-trivially.

Now let us consider the EKE compiler/protocol [8]. The client samples x and sends $\mathsf{E}(\mathsf{pw}, g^x)$, where $\mathsf{E}(\cdot)$ is a symmetric encryption under key pw. Similarly, the server samples y and sends $\mathsf{E}(\mathsf{pw}, g^y)$. The session key is computed as $\mathsf{key} = H(\mathsf{aux}, Z = g^{xy}, \mathsf{pw})$ with aux some public information. Now we explain why it is difficult for EKE to achieve tight security based on the CDH assumption.

In the reduction, given a CDH problem instance $(g^{\bar{x}}, g^{\bar{y}})$, the reduction algorithm $\mathcal{B}$ may use the random self-reducibility of the DH problem to generate multiple (g^{x_i}, g^{y_j}), and embed them into multiple protocol sessions. Since $H(\cdot)$ works as a random oracle, $\mathcal{A}$ has no advantage in distinguishing a real session key from a random key, unless it queries $H(\cdot)$ on the right CDH value $g^{x_i y_j}$. Now suppose that $\mathcal{A}$ does query $H(\cdot)$ on the right CDH value, here come two problems for $\mathcal{B}$.

(1) $\mathcal{A}$ may ask hash queries on $(\mathsf{aux}, Z_i, \mathsf{pw})$ with different Z_i, but $\mathcal{B}$ cannot identify/compute the right CDH value $g^{\bar{x}\bar{y}}$ from all Z_i. Therefore, $\mathcal{B}$ has to guess one for the CDH problem, leading to a loose security factor Q_h (maximum number of hash queries).

(2) $\mathcal{A}$ may correctly guess the password and send g^y out after seeing some g^{x_i}, i.e., $\mathcal{A}$ has the power to compute $g^{x_i y}$ and hence the session key. However, without the knowledge of x_i, $\mathcal{B}$ is unable to compute $g^{x_i y}$.

To solve these two problems, a natural idea is resorting to a decision oracle, and that is exactly what [1, 6] did. However, [1, 6] rely on the non-standard Gap DH assumption. In this paper, we solve these two problems with the twin DH decision oracle and the standard CDH assumption.

Twin DH decision oracle. In [16], Cash et al. proposed the strong twin-DH (st2DH) assumption and proved its equivalence to the (standard) CDH assumption. Here the strong 2DH problem is to compute $(g^{\bar{x}_1\bar{y}}, g^{\bar{x}_2\bar{y}})$, given $g^{\bar{x}_1}, g^{\bar{x}_2}, g^{\bar{y}}$, as well as a decision oracle $2\mathrm{DH}(\cdot,\cdot,\cdot)$ that inputs (Y, Z_1, Z_2) and outputs whether $(\bar{X}_1, Y, Z_1)$ and $(\bar{X}_2, Y, Z_2)$ are both DDH tuples. Inspired by [16], we propose our 2DH variant protocol for EKE, named 2DH-EKE. Now the client sends $\mathsf{E}(\mathsf{pw}, g^{x_1}||g^{x_2})$ and the server sends $\mathsf{E}(\mathsf{pw}, g^y)$, and the session key is computed as $\mathsf{key} = H(\mathsf{aux}, Z_1 = g^{x_1 y}, Z_2 = g^{x_2 y}, \mathsf{pw})$ with aux some public information. Next, we show how the twin DH decision oracle can be used to solve the above two problems.

(1) With the decision oracle $2\mathrm{DH}(\cdot,\cdot,\cdot)$, the reduction algorithm $\mathcal{B}$ can easily locate the correct Z_1, Z_2 among all possible candidates, by checking whether $2\mathrm{DH}(Y, Z_1, Z_2) = 1$. In this way, $\mathcal{B}$ succeeds in solving the strong 2DH problem, and avoiding the loose factor Q_h.

(2) In the reduction $\mathcal{B}$ may need to simulate the session key $\mathsf{key} = H(\mathsf{aux}, g^{x_1 y}, g^{x_1 y}, \mathsf{pw})$ for some adversarially generated g^y, and the exponents $x_1||x_2$ are unknown to $\mathcal{B}$ due to the embedded hard problem. In this case, $\mathcal{B}$ randomly samples a key and implicitly sets it as the "right" key. Since $H(\cdot)$ works as a random oracle, $\mathcal{A}$ will not obverse this difference unless it asks a hash query on the right 2DH values Z_1, Z_2 later. If this happens, $\mathcal{B}$ can detect it with the decision oracle,

and reprogram the random oracle such that $H(\mathsf{aux}, Z_1, Z_2, \mathsf{pw}) = \mathsf{key}$, and the view of $\mathcal{A}$ is consistent.

Towards UC security. To achieve UC security, we need to construct a PPT simulator to simulate the interactions with the environment in the real world, with the help of the ideal functionality $\mathcal{F}$. In our 2DH-EKE protocol, the symmetric encryption (E, D) is modeled as an Ideal Cipher (IC), and hence the transcripts ($e_1 = \mathsf{E}(\mathsf{pw}, X_1||X_2)$ and $e_2 = \mathsf{E}(\mathsf{pw}, Y)$) are perfect hiding. Consequently, the simulator can perfectly simulate the transcripts with random messages.

To deal with the adversarially generated message (say e_1'), we can always look up the IC list to extract the password $\mathcal{A}$ guesses "in mind". Then the simulator can resort to the TestPW interface provided by $\mathcal{F}$, to check whether $\mathcal{A}$ succeeds in guessing the password. If yes, the simulator can compute the "real" session key, with the help of the twin DH decision oracle, as discussed above. Otherwise, the session key is simulated as a random key, and this is indistinguishable to the adversary due to the CDH assumption.

Asymmetric PAKE. Generally in the scenario of aPAKE, the server stores a password file (usually a hash of the password) rather than the password in plain. The resistance to server compromise requires that getting the password file helps nothing for the adversary in impersonating a client, unless it implements a brute-force attack and successfully recovers the pre-image pw. In this paper, we develop our 2DH-EKE to an aPAKE protocol 2DH-aEKE, with only one extra round to transmit a confirming message.

2DH-aEKE inherits the idea of the generic CDH-based compiler in [26], and it works as follows. Let $\mathsf{H}_0(\cdot)$ be a hash function, $\mathsf{H}_0(\mathsf{pw}) = (\mathsf{h}, v_1, v_2)$ and $V_1 := g^{v_1}, V_2 := g^{v_2}$. Now the password file stored in the server is (h, V_1, V_2). In the execution of 2DH-aEKE, the client and the server first run the symmetric 2DH-EKE protocol using h as the key of symmetric encryption. Recall that the client and the server's unencrypted messages are $(X_1||X_2) = (g^{x_1}||g^{x_2})$ and $Y = g^y$, respectively. Let $H(\mathsf{aux}, g^{x_1 y}, g^{x_2 y}, g^{v_1 y}, g^{v_2 y}, \mathsf{h}) = (\mathsf{key}, \sigma)$, where aux is the public information, key is the sesssion key, and σ is the key confirmation message. Then the client sends σ to the server as an extra round message. From the strong 2DH assumption we know that it is hard to compute $g^{v_1 y}||g^{v_2 y}$, even with the password file (h, V_1, V_2) and Y. That is how the security of 2DH-aEKE is guaranteed even after the server compromise. Note that the security reduction has a loss factor of N, the number of total client-server pairs, due to the commitment of client's password in the password file.

With a meta-reduction, we prove that the security loss of aPAKE is lower bounded by N. Hence, our 2DH-aEKE meets the optimal reduction loss. Now we give an intuition why the loss factor N is inevitable in aPAKE. In the reduction, the hard problem $(\bar{X}_1, \bar{X}_2, \bar{Y})$ is embedded into the password file $V_1||V_2$ and the server's message Y, respectively. Meanwhile, if $\mathcal{A}$ asks the value of $\mathsf{H}_0(\mathsf{pw})$ with the correct password, then the discrete log of $V_1||V_2$ should be returned. However, the reduction algorithm does not know whether and when $\mathcal{A}$ will issue such a query. Hence, it has to choose a particular client-server pair among all N pairs, embed the hard problem into this password file, and hope $\mathcal{A}$ breaks the security of one session involving this password file but does not query $\mathsf{H}_0(\mathsf{pw})$ at the time being.

Recall that almost all previous aPAKE schemes [26, 41, 21] have a loose reduction loss at least $Q_h N\theta$, where Q_h, N, θ denote the maximum numbers of hash queries, client-server pairs, and protocol executions per client-server pair, respectively. We stress that the decision oracle 2DH helps us improving the loss factor from $Q_h N\theta$ to the optimal bound N (note that $Q_h N\theta \gg N$ in general).

Extend to the relaxed UC framework. Our method can also apply to some IC-free protocols like SPAKE2 [4] and PPK [35], to get their 2DH variants. And the tight security can be proved based on the CDH assumption in the relaxed UC framework [1]. We take the SPAKE2 protocol as an example. In SPAKE2, the transcript messages are $X \cdot M^{\mathsf{pw}}$ and $Y \cdot N^{\mathsf{pw}}$ with M, N public parameters. In our 2DH-SPAKE2, X is replaced by $(X_1||X_2) = (g^{x_1}||g^{x_2})$, Y is replaced by $(Y_1||Y_2) = (g^{y_1}||g^{y_2})$, and the session key is computed as $\mathsf{key} = H(\mathsf{aux}, g^{x_1 y_1}, g^{x_1 y_2}, g^{x_2 y_1} g^{x_2 y_2}, \mathsf{pw})$. Similar to the proof of 2DH-EKE, the decision oracle 2DH is essential to make a tight reduction in the relaxed UC framework.

Forward security. Both 2DH-EKE and 2DH-aEKE achieve <u>P</u>erfect <u>F</u>orward <u>S</u>ecurity [22] (PFS, a.k.a. perfect forward secrecy). PFS means that once a party is corrupted at some moment, then all session keys completed before the corruption remain hidden from the adversary. Let us take 2DH-EKE as an example. Note that a completed session has already uniquely determined e_1 and e_2, even if one of them is adversarially generated. If $\mathcal{A}$ later gets pw via a corruption, the information it obtains from the corruption is limited by $X_1||X_2 = \mathsf{D}(\mathsf{pw}, e_1)$ and $Y = \mathsf{D}(\mathsf{pw}, e_2)$. However, given $X_1||X_2$ and Y, computing the session key is as hard as solving the 2DH problem, and PFS is guaranteed as a result. The analysis of PFS for SPAKE2 (2DH-SPAKE2) can be found in [1].

1.2 Roadmap

This paper is organised as follows. In Section 2 we present preliminaries, including notations and some hardness assumptions. In Section 3 we describe the UC framework for PAKE, propose the 2DH-EKE protocol, and prove its security. In Section 4 we describe the UC framework for aPAKE, and propose the asymmetric variant 2DH-aEKE protocol. The optimal reduction loss in aPAKE is shown in Section 5. Consequently, we extend our technique to SPAKE2 to obtain 2DH-SPAKE2 in Section 6. We refer the full version [34] for details of the proofs, and the functionalities of ideal ciphers, random oracles, and lazy-extraction PAKE.

2 Preliminaries

We use $\lambda \in \mathbb{N}$ to denote the security parameter throughout the paper. Denote by $x := y$ the operation of assigning y to x. Denote by $x \xleftarrow{\$} \mathcal{X}$ the operation of sampling x uniformly at random from a set $\mathcal{X}$. For an algorithm $\mathcal{A}$, denote by

$y \leftarrow \mathcal{A}(x; r)$, or simply $y \leftarrow \mathcal{A}(x)$, the operation of running $\mathcal{A}$ with input x and randomness r and assigning the output to y. "PPT" is short for probabilistic polynomial-time.

The the functionalities of ideal ciphers and random oracles are given in the full version [34].

2.1 Hardness Assumptions

Let GGen be a group generation algorithm such that $(\mathbb{G}, q, g) \leftarrow \mathsf{GGen}(1^\lambda)$, where $\mathbb{G}$ is a cyclic group of prime order q with generator g.

Definition 1. *For any adversary $\mathcal{A}$, its advantage in solving the Computational Diffie-Hellman (CDH) problem is defined as*

$$\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G},\mathcal{A}}(\lambda) := \Pr[x, y \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}(g, g^x, g^y) = g^{xy}].$$

In [16], Cash et al. proposed the Strong Twin Diffie-Hellman (strong 2DH or st2DH) problem, and proved that it is as hard as the CDH problem.

Definition 2. *[16] For any adversary $\mathcal{A}$, its advantage in solving the st2DH problem is defined as*

$$\mathsf{Adv}^{\mathsf{st2DH}}_{\mathbb{G},\mathcal{A}}(\lambda) := \Pr[\bar{x}_1, \bar{x}_2, \bar{y} \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}^{\mathrm{2DH}(\cdot,\cdot,\cdot)}(g, g^{\bar{x}_1}, g^{\bar{x}_2}, g^{\bar{y}}) = (g^{\bar{x}_1\bar{y}}, g^{\bar{x}_2\bar{y}})],$$

where the decision oracle 2DH$(\cdot,\cdot,\cdot)$ inputs (g^y, g^{z_1}, g^{z_2}) and outputs 1 if $(\bar{x}_1 y = z_1) \wedge (\bar{x}_2 y = z_2)$ and 0 otherwise.

The st2DH assumption was proven equivalent to the CDH assumption [16].

Theorem 1. *[16] For any PPT adversary $\mathcal{A}$ against the st2DH problem, there exists a PPT algorithm $\mathcal{B}$ against the CDH problem such that $\mathsf{Adv}^{\mathsf{st2DH}}_{\mathbb{G},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G},\mathcal{B}}(\lambda) + Q/q$, where Q is the maximum number of decision oracle queries.*

In the following sessions, we also use the notations $\mathsf{CDH}(g^x, g^y) = g^{xy}$, and $\mathsf{2DH}(g^{x_1}, g^{x_2}, g^y) = (g^{x_1 y}, g^{x_2 y})$ for arbitrary elements $g^x, g^y, g^{x_1}, g^{x_2}$ in $\mathbb{G}$.

3 PAKE with Tight Security in the UC Framework

3.1 UC Framework for PAKE

We assume basic familiarity with the Universally Composable framework (UC framework, a.k.a. UC model) for PAKE. The ideal functionality $\mathcal{F}_{\mathsf{pake}}$ is shown in Fig. 1. We mainly follow the definition by Shoup in [41], which is a modified version of [15] by Canetti et al. For a full understanding of UC framework, we refer [15, 41] for details.

Overview of the UC framework. The ideal functionality $\mathcal{F}_{\mathsf{pake}}$ plays the role of a trusted authority in the ideal world. A client and a server first share the same password when registration, after which $\mathcal{F}_{\mathsf{pake}}$ records the password privately. When initializing a new PAKE session, both the two parties send a query to $\mathcal{F}_{\mathsf{pake}}$, and the client additionally sends a password (since it is very possible for a client to mistype the password, see the description below). Then $\mathcal{F}_{\mathsf{pake}}$ verifies whether the password from the client matches the (correct) password stored by the server. If yes, these two parties are "matched" and they will get the same random session key from $\mathcal{F}_{\mathsf{pake}}$. Otherwise, they are "dismatched" and the execution of PAKE fails (the output may be arbitrary in this case). Security in this ideal model holds inherently, since nothing except the identities of involved parties is leaked to the simulator/adversary Sim in the ideal world, and the only attack Sim can apply, is an online attack.

The security target of a PAKE protocol Π, is to emulates the ideal functionality $\mathcal{F}_{\mathsf{pake}}$ in the real world. More precisely, consider an environment $\mathcal{Z}$ that controls passwords for all parties[4], and it aims to distinguish the real world from the ideal world, i.e., distinguish the case where outputs including session keys are produced via executions of Π compelled by an adversary $\mathcal{A}$, from the case where outputs are obtained from $\mathcal{F}_{\mathsf{pake}}$ and an simulator Sim interacting with $\mathcal{F}_{\mathsf{pake}}$. If for any PPT environment $\mathcal{Z}$, the distinguishing advantage is negligible, we say PAKE protocol Π securely emulates $\mathcal{F}_{\mathsf{pake}}$.

Now we describe $\mathcal{F}_{\mathsf{pake}}$ in more detail.

Password storage and sessions. We require two parties involved in a PAKE execution have different roles (client or server), and each party has a unique identity, namely, $\mathsf{C}^{(i)}$ or $\mathsf{S}^{(j)}$. In the registration stage, the environment $\mathcal{Z}$ allocates a password $\hat{\mathsf{pw}}$ for each client-server pair $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)})$. The functionality $\mathcal{F}_{\mathsf{pake}}$ then records this password after a $\mathsf{StorePWFile}$ query from $\mathsf{C}^{(i)}$ or $\mathsf{S}^{(j)}$. Without loss of generality, we assume each pair of $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ has only one password.

For a party P, we call an execution of protocol a (session) instance, and index it with an instance identity iid. After registration, P can initialize a new session instance via a $\mathsf{NewClient}$ or $\mathsf{NewServer}$ query to $\mathcal{F}_{\mathsf{pake}}$. For a server $\mathsf{S}^{(j)}$, the password pw used in this instance is set to be the correct password $\hat{\mathsf{pw}}$ pre-shared between $\mathsf{C}^{(i)}$ and $\mathsf{S}^{(j)}$. For a client $\mathsf{C}^{(i)}$, it is possible that $\mathsf{pw} \neq \hat{\mathsf{pw}}$ due to a mistyped/misremembered password.

Following the definition in [41], we explicitly model mistyped or misremembered passwords in $\mathcal{F}_{\mathsf{pake}}$, instead of absorbing it into an active attack by the adversary $\mathcal{A}$ (though this is enough from the perspective of PAKE security, i.e., preventing a bad client from logging into the server). Actually, a mistyped password is very close to the correct password, and an accidental mismatch would not compromise this nearly-identical password to $\mathcal{A}$.

Active attacks. To capture online attacks in the real world, $\mathcal{F}_{\mathsf{pake}}$ allows the simulator Sim in the ideal world to make a password guess per instance via the interface TestPW. If the guess is correct, then the session instance is marked as

[4] Let the environment deciding passwords captures the security in case users' passwords are arbitrarily distributed and correlated. This is one aspect in which the UC framework is superior to the IND model.

Functionality $\mathcal{F}_{\mathsf{pake}}$

The functionality $\mathcal{F}_{\mathsf{pake}}$ is parameterized by a security parameter λ. It interacts with a simulator Sim and a set of parties via the following queries:

Password Storage

Upon receiving a query $(\mathsf{StorePWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}})$ **from a client** $\mathsf{C}^{(i)}$ **or a server** $\mathsf{S}^{(j)}$:

If there exists a record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \cdot \rangle$, ignore this query.
Otherwise, record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}} \rangle$, and send $(\mathsf{StorePWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ to Sim.

Sessions

Upon receiving a query $(\mathsf{NewClient}, iid^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$ **from a client** $\mathsf{C}^{(i)}$:

Retrieve the record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}} \rangle$. Send $(\mathsf{NewClient}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw} = \hat{\mathsf{pw}}?)$ to Sim. Record $(\mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$ and mark it as fresh.
In this case, $\mathsf{S}^{(j)}$ is called the intended partner of $(\mathsf{C}^{(i)}, iid^{(i)})$.

Upon receiving a query $(\mathsf{NewServer}, iid^{(j)}, \mathsf{C}^{(i)})$ **from a server** $\mathsf{S}^{(j)}$:

Retrieve the record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}} \rangle$. Send $(\mathsf{NewServer}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)})$ to Sim. Set $\mathsf{pw} := \hat{\mathsf{pw}}$, record $(\mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)}, \mathsf{pw})$ and mark it as fresh.
In this case, $\mathsf{C}^{(i)}$ is called the intended partner of $(\mathsf{S}^{(j)}, iid^{(j)})$.

Two instances $(\mathsf{C}^{(i)}, iid^{(i)})$ and $(\mathsf{S}^{(j)}, iid^{(j)})$ are said to be partnered, if there are two fresh records $(\mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$ and $(\mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)}, \mathsf{pw})$ sharing the same pw.

Active Session Attacks

Upon receiving a query $(\mathsf{TestPW}, P, iid, \mathsf{pw}')$ **from** Sim:

If there is a fresh record $(P, iid, \cdot, \mathsf{pw})$:
- If $\mathsf{pw}' = \mathsf{pw}$, mark the record $\mathsf{compromised}$ and reply to Sim with "correct guess".
- If $\mathsf{pw}' \neq \mathsf{pw}$, mark the record $\mathsf{interrupted}$ and replay with "wrong guess".

Key Generation

Upon receiving a query $(\mathsf{FreshKey}, P, iid, sid)$ **from** Sim:

If 1) there is a fresh or $\mathsf{interrupted}$ record (P, iid, Q, pw); and 2) sid has never been assigned to P's any other instance (P, iid'):
Pick a new random key k, mark the record (P, iid, Q, pw) as $\mathsf{completed}$, assign it with sid, send (iid, sid, k) to P, and record (P, Q, sid, k).

Upon receiving a query $(\mathsf{CopyKey}, P, iid, sid)$ **from** Sim:

If 1) there is a fresh record (P, iid, Q, pw) and a $\mathsf{completed}$ record $(Q, iid^*, P, \mathsf{pw})$ s.t. (P, iid) and (Q, iid^*) are partnered; and 2) sid has never been assigned to P's any other instance (P, iid'); and 3) there is a unique (Q, iid^*) that has been assigned with sid:
Retrieve the record (Q, P, sid, k), mark the record (P, iid, Q, pw) as $\mathsf{completed}$, assign it with sid, and send (iid, sid, k) to P.

Upon receiving a query $(\mathsf{CorruptKey}, P, iid, sid, k)$ **from** Sim:

If 1) there is a $\mathsf{compromised}$ record $(P, iid, \mathcal{Q}, \mathsf{pw})$; and 2) sid has never been assigned to P's any other instance (P, iid'):
Mark the record $(P, iid, \mathcal{Q}, \mathsf{pw})$ as $\mathsf{completed}$, assign it with sid, and send (iid, sid, k) to P.

Fig. 1. The PAKE functionality $\mathcal{F}_{\mathsf{pake}}$ [41].

$\mathsf{compromised}$, which means that the adversary succeeds in attacking this instance and can affect the generation of the session key. If the guess is wrong, then the instance is marked as $\mathsf{interrupted}$, indicating a failed online attack, and the session key is chosen at random.

Via (static) corruptions, a real world adversary can learn the password hold by a party and control its behaviour completely. To make the view of the environment consistent, the simulator Sim in the ideal world also obtains the password of that party, and simulates what it outputs in an indistinguishable way. Note that the corruption process is not explicitly presented in $\mathcal{F}_{\mathsf{pake}}$ in Fig 1.

Key generation. For an instance (P, iid), when the protocol execution is completed, $\mathcal{F}_{\mathsf{pake}}$ will assign to the instance a key and a session identity sid which is determined by Sim. And sid is required to uniquely index this completed instance (the two parties in a session would share the same sid if there is no active attack). Furthermore, $\mathcal{F}_{\mathsf{pake}}$ provides three types of interfaces for key generation.

- $\mathsf{FreshKey}$. When a successful protocol execution finishes and one instance needs to output a session key first, or the passwords do not match (including the case of a failed password guess), the instance is assigned with an independent and random key.
- $\mathsf{CopyKey}$. If there are two instances that match with each other, and a fresh key has been assigned to one instance before, then a copy of the session key is passed to the other instance.
- $\mathsf{CorruptKey}$. If one of the participating parties is corrupted, or the adversary successfully guesses the password, then the session key is totally determined by Sim.

Remark 1 (Session identities). $\mathcal{F}_{\mathsf{pake}}$ implicitly assumes that sid allocated by the simulator differs for each instance (even for two different instances of the same party) except for the two partnered instances. As we will see, this is indeed the case in 2DH-EKE, since sid connects the identities of the client, the server, and the session transcripts, and each instance contributes its own randomness to transcripts. So once an instance is completed and has been assigned with (sid, k), the information of sid is sufficient to locate the unique and partnered pair (P, iid, Q, pw) and $(Q, iid^*, P, \mathsf{pw})$, when dealing with $\mathsf{CopyKey}$ queries.

Remark 2 (Corruptions). Our PAKE framework deals with static corruptions, i.e., the adversary can corrupt some parties and get their passwords prior to the protocol execution. Note that there is a stronger model that supports adaptive corruptions, where the adversary can corrupt parties adaptively throughout the execution, and obtain not only the passwords but also the internal states. Almost all UC frameworks [15, 41] for PAKE are defined in the way of static corruptions.

3.2 The 2DH-EKE Protocol

The EKE compiler/protocol was proposed by Bellovin and Merritt in [8], and formally proved later by Bellare et al. in the IND model [7], and by Dupont et al.

in the UC framework [17]. The security proof is based on the CDH assumption in the IC and RO model, and has a security loss $L = Q_h \cdot N \cdot \theta$, with Q_h, N, θ the maximum numbers of hash queries, client-server pairs, and protocol executions per client-server pair, respectively.

In this subsection, we present a variant of EKE, named 2DH-EKE protocol, and prove its tight security based on the strong 2DH assumption (equivalently, the CDH assumption) in the UC framework.

The 2DH-EKE protocol is shown in Fig. 2. Here $(\mathsf{E}_1, \mathsf{D}_1)$ is a symmetric encryption with key space $\mathcal{PW}$, plaintext space $\mathbb{G}^2$ and ciphertext space $\mathcal{E}_1$, and $(\mathsf{E}_2, \mathsf{D}_2)$ is a symmetric encryption with key space $\mathcal{PW}$, plaintext space $\mathbb{G}$ and ciphertext space $\mathcal{E}_2$. Hash function H is defined as $\mathsf{H} : \{0,1\}^* \mapsto \mathcal{K}$ with $\mathcal{K}$ the space of session keys. C, S are identities of Client and Server.

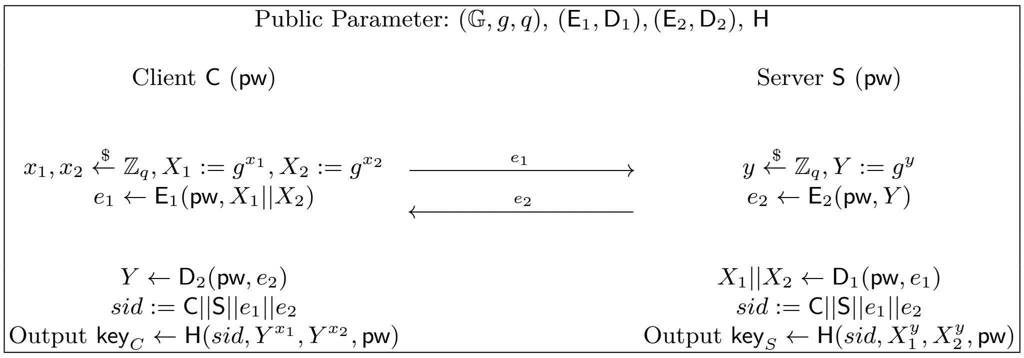

Public Parameter: $(\mathbb{G}, g, q)$, $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$, H
Client C (pw)
Server S (pw)
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q, X_1 := g^{x_1}, X_2 := g^{x_2}$
$e_1 \leftarrow \mathsf{E}_1(\mathsf{pw}, X_1||X_2)$
$\xrightarrow{e_1}$
$\xleftarrow{e_2}$
$y \xleftarrow{\$} \mathbb{Z}_q, Y := g^y$
$e_2 \leftarrow \mathsf{E}_2(\mathsf{pw}, Y)$
$Y \leftarrow \mathsf{D}_2(\mathsf{pw}, e_2)$
$sid := \mathsf{C}||\mathsf{S}||e_1||e_2$
Output $\mathsf{key}_C \leftarrow \mathsf{H}(sid, Y^{x_1}, Y^{x_2}, \mathsf{pw})$
$X_1||X_2 \leftarrow \mathsf{D}_1(\mathsf{pw}, e_1)$
$sid := \mathsf{C}||\mathsf{S}||e_1||e_2$
Output $\mathsf{key}_S \leftarrow \mathsf{H}(sid, X_1^y, X_2^y, \mathsf{pw})$

Fig. 2. The 2DH-EKE protocol.

Remark 3. The 2DH-EKE protocol can be modified to a variant protocol by interchanging the operations of Client and Server: the client sends $e_1 = \mathsf{E}_1(\mathsf{pw}, X)$ and the server sends $e_2 = \mathsf{E}_2(\mathsf{pw}, Y_1||Y_2)$. In this way, the computational cost of Client is reduced, but Server has to initiate the session. In this paper we do not take this variant, since Client will start a session in general cases.

Remark 4 (Ideal ciphers on group elements). The ideal cipher in the 2DH-EKE protocol can be accomplished with a block cipher like AES. Take $e_1 = \mathsf{E}_1(\mathsf{pw}, X)$ as an example. First, the group element X is mapped to an n-bit string through a quasi bijection [21], and then the encryption algorithm encrypts the n-bit string with the password. The decryption algorithm D_1 can be similarly defined. For more details on implementations of IC, see [21].

Remark 5 (Comparisons with the twin DH protocol [16] and KC-SPAKE2 [41]). Note that Cash et al. [16] extended the DH key exchange protocol to a twin DH version and proved its tight security. In the twin DH protocol, one party publishes (X_1, X_2) and the other party publishes (Y_1, Y_2), and the session key is the hash value $H(g^{x_1 y_1}, g^{x_1 y_2}, g^{x_2 y_1}, g^{x_2 y_2})$. In contrast, the server's (plain) message in our 2DH-EKE protocol consists of only one element Y, which greatly decreases the computation/communication cost.

In [41], Shoup showed the (non-tight) security of KC-SPAKE2 based on the CDH assumption, and argued that the reduction is tight under the Gap DH assumption. In contrast, our tight reduction of 2DH-EKE is based on the standard CDH assumption.

3.3 Security Analysis

Theorem 2 (Security of 2DH-EKE). *If the st2DH assumption (equivalently, the CDH assumption) holds in $\mathbb{G}$, $(\mathsf{E}_1, \mathsf{D}_1)$ and $(\mathsf{E}_2, \mathsf{D}_2)$ work as ideal ciphers, and H works as a random oracle, then the 2DH-EKE protocol in Fig. 2 securely emulates $\mathcal{F}_{\mathsf{pake}}$. More precisely, for any PPT environment $\mathcal{Z}$ and real world adversary $\mathcal{A}$ which has access to ideal ciphers $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$ and random oracle H, there exist a PPT simulator Sim, which has access to the ideal functionality $\mathcal{F}_{\mathsf{pake}}$, and algorithms $\mathcal{B}, \mathcal{B}'$, s.t. the advantage of $\mathcal{Z}$ in distinguishing the real world running with $\mathcal{A}$ and the ideal world running with Sim is bounded by*

$$\begin{aligned}\mathsf{Adv}_{\mathsf{2DH\text{-}EKE},\mathcal{Z}}(\lambda) \leq & 2\mathsf{Adv}^{\mathsf{st2DH}}_{\mathbb{G},\mathcal{B}}(\lambda) + \frac{Q_{ic}^2}{|\mathcal{E}_1|} + \frac{Q_{ic}^2}{|\mathcal{E}_2|} + 2^{-\Omega(\lambda)} \\ \leq & 2\mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G},\mathcal{B}'}(\lambda) + 2^{-\Omega(\lambda)}.\end{aligned}$$

where Q_{ic} denotes the maximum number of IC queries.

Proof. The main task of the proof, is to construct a PPT simulator Sim, which has access to the ideal functionality $\mathcal{F}_{\mathsf{pake}}$ and interactions with the environment $\mathcal{Z}$, and simulates the real world 2DH-EKE protocol interactions among the adversary $\mathcal{A}$, parties, and the environment $\mathcal{Z}$. To this end, Sim needs to simulate honestly generated messages from real parties, respond adversarial messages approximately, and simulate ideal functions $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$, and H, as shown in Fig. 3. The functionality $\mathcal{F}_{\mathsf{pake}}$ provides information to Sim through interfaces including TestPW, $\mathsf{NewClient}$, $\mathsf{NewServer}$, $\mathsf{FreshKey}$, $\mathsf{CopyKey}$, and $\mathsf{CorruptKey}$, as defined in Fig. 1. Recall that Sim has no secret inputs (i.e., passwords).

The full description of the simulator Sim is given in Fig. 4. Let $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$ be the real experiment where environment $\mathcal{Z}$ interacts with real parties and adversary $\mathcal{A}$, and $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$ be the ideal experiment where $\mathcal{Z}$ interacts with simulator Sim. We prove that $|\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}} \Rightarrow 1]|$ is negligible via a series of games **Game 0 − 5**, where **Game 0** is $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$, **Game 5** is $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$, and argue that the adjacent two games are indistinguishable from $\mathcal{Z}$'s prospective of view.

We consider the scenario of multi-users and multi instances. Let $\mathsf{C}^{(i)}$ (resp., $\mathsf{S}^{(j)}$) denote clients (resp., servers) with superscript (i) (resp., (j)) indexing different clients (resp., servers). Let $(\mathsf{C}^{(i)}, iid^{(i)})$ denote client instances of $\mathsf{C}^{(i)}$ with $iid^{(i)}$ indexing its different instances. Similarly, let $(\mathsf{S}^{(j)}, iid^{(j)})$ denote server instances of $\mathsf{S}^{(j)}$ with $iid^{(j)}$ indexing its different instances. For better presentation of the proof, we give some definitions as follows.

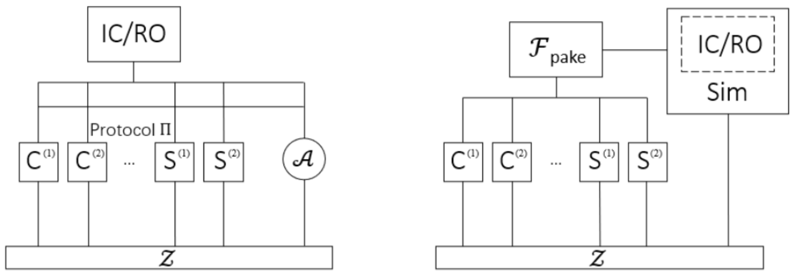


Fig. 3. The real world execution (left) and the ideal world execution (right).

Good/Bad client instance. We call a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ a *good* (resp., *bad*) one, if the password pw used in this instance equals (resp., differs from) the correct password $\hat{\mathsf{pw}}$ shared between $\mathsf{C}^{(i)}$ and its intended partner $\mathsf{S}^{(j)}$. Note that a bad client instance indicates the case that the client mistypes its password.

Linked instances. We say that a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ is linked to a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ (no matter good or bad), if e_1 generated by $(\mathsf{C}^{(i)}, iid^{(i)})$ is received by one instance $(\mathsf{S}^{(j)}, iid^{(j)})$ of its intended partner $\mathsf{S}^{(j)}$. Similarly, we say a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ is linked to a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$, if e_2 generated by $(\mathsf{S}^{(j)}, iid^{(j)})$ is received by one instance $(\mathsf{C}^{(i)}, iid^{(i)})$ of its intended partner $\mathsf{C}^{(i)}$. If the two instances are linked to each other, then they are called linked instances.

Game 0. This is the real experiment $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$. In this experiment, $\mathcal{Z}$ initializes a password for each client-server pair, sees the interactions among clients, servers and adversary $\mathcal{A}$, and also obtains the corresponding session keys of protocol instances. Here $\mathcal{A}$ may implement attacks like view, modify, insert, or drop messages over the network. We have

$$\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}} \Rightarrow 1] = \Pr[\mathbf{Game\ 0} \Rightarrow 1].$$

Game 1. (Add an ideal layout.) From this game on, we add an ideal layout Sim[5], which is only a toy construction in **Game 1**, but will be complete with games going on and arrive at the final Sim defined in Fig. 4. In **Game 1**, Sim still needs to take passwords as inputs. With the help of passwords, it perfectly simulates the executions in $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$, except that the encryption of IC is simulated in a collision-free way. Meanwhile, Sim also necessarily keeps the exponent values of the decrypted group elements from D_1 and D_2. More precisely, it maintains lists $\mathcal{L}_{\mathsf{IC}_1}, \mathcal{L}_{\mathsf{IC}_2}, \mathcal{T}_{\mathsf{IC}_1}, \mathcal{T}_{\mathsf{IC}_2}, \mathcal{L}_{\mathsf{H}}, \mathcal{DL}$ (all initialized to be empty sets) and works as follows.

[5] The simulators in **Game** $\mathbf{1-4}$ are semi-manufactured, which help us to analyze the differences between the real world and the ideal world step by step. For simplicity, we still use the same notation Sim in **Game** $\mathbf{1-4}$.

Sim maintains lists $\mathcal{L}_{\mathsf{IC}_1}, \mathcal{L}_{\mathsf{IC}_2}, \mathcal{T}_{\mathsf{IC}_1}, \mathcal{T}_{\mathsf{IC}_2}, \mathcal{L}_{\mathsf{H}}, \mathcal{T}, \mathcal{DL}$ (all initialized to be empty) in the simulation.

- $\mathcal{L}_{\mathsf{IC}_1}, \mathcal{L}_{\mathsf{IC}_2}, \mathcal{T}_{\mathsf{IC}_1}, \mathcal{T}_{\mathsf{IC}_2}$: store records w.r.t. ideal ciphers $(\mathsf{E}_1, \mathsf{D}_1)$ and $(\mathsf{E}_2, \mathsf{D}_2)$.
- $\mathcal{L}_{\mathsf{H}}$: store records w.r.t. random oracle H.
- $\mathcal{T}$: store messages sent by client/server instances.
- $\mathcal{DL}$: store discrete logarithms.

PAKE Sessions

```
on $(\mathsf{NewClient}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, b)$ from $\mathcal{F}_{\mathsf{pake}}$:
    $e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_{\mathsf{IC}_1}$, $\mathcal{T}_{\mathsf{IC}_1} := \mathcal{T}_{\mathsf{IC}_1} \cup \{e_1\}$, $\mathcal{T} := \mathcal{T} \cup \{(\mathsf{C}^{(i)}, iid^{(i)}, e_1)\}$, send $e_1$ from $\mathsf{C}^{(i)}$ to $\mathcal{A}$.
    If $b = 1$: mark $(\mathsf{C}^{(i)}, iid^{(i)})$ as $\mathsf{correct\text{-}pw}$. // client $\mathsf{C}^{(i)}$ correctly inputs the password
on $(\mathsf{NewServer}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)})$ from $\mathcal{F}_{\mathsf{pake}}$ and $e_1$ from $\mathcal{A}$ as a client message from $\mathsf{C}^{(i)}$ to $(\mathsf{S}^{(j)}, iid^{(j)})$:
    $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_{\mathsf{IC}_2}$, $\mathcal{T}_{\mathsf{IC}_2} := \mathcal{T}_{\mathsf{IC}_2} \cup \{e_2\}$, $\mathcal{T} := \mathcal{T} \cup \{(\mathsf{S}^{(j)}, iid^{(j)}, e_2)\}$, send $e_2$ from $\mathsf{S}^{(j)}$ to $\mathcal{A}$.
    $sid := \mathsf{C}^{(i)} || \mathsf{S}^{(j)} || e_1 || e_2$.
    If $\exists(\mathsf{pw}', X_1 || X_2, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$: ask $(\mathsf{TestPW}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{pake}}$, and if $\mathcal{F}_{\mathsf{pake}}$ returns "correct guess":
        Let $X_1 || X_2 \leftarrow \mathsf{D}_1(\mathsf{pw}', e_1)$ and $Y \leftarrow \mathsf{D}_2(\mathsf{pw}', e_2)$, retrieve item $(Y, y) \in \mathcal{DL}$, $Z_1 := X_1^y$, $Z_2 := X_2^y$,
        $\mathsf{key} \leftarrow \mathsf{H}(sid, Z_1, Z_2, \mathsf{pw}')$, send $(\mathsf{CorruptKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{pake}}$.
    In other cases: send $(\mathsf{FreshKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{pake}}$.
on $e_2$ from $\mathcal{A}$ as a server message from $\mathsf{S}^{(j)}$ to $(\mathsf{C}^{(i)}, iid^{(i)})$:
    Retrieve $(\mathsf{C}^{(i)}, iid^{(i)}, e_1) \in \mathcal{T}$, $sid := \mathsf{C}^{(i)} || \mathsf{S}^{(j)} || e_1 || e_2$.
    If $(\mathsf{C}^{(i)}, iid^{(i)})$ is $\mathsf{correct\text{-}pw}$, $\exists(\mathsf{S}^{(j)}, \cdot, e_2) \in \mathcal{T}$, and $\mathsf{Sim}$ has queried $(\mathsf{FreshKey}, \mathsf{S}^{(j)}, \cdot, sid)$:
        Send $(\mathsf{CopyKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$ to $\mathcal{F}_{\mathsf{pake}}$.
    If $\exists(\mathsf{pw}', Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$: ask $(\mathsf{TestPW}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{pake}}$, and if $\mathcal{F}_{\mathsf{pake}}$ returns "correct guess":
        Let $X_1 || X_2 \leftarrow \mathsf{D}_1(\mathsf{pw}', e_1)$ and $Y \leftarrow \mathsf{D}_2(\mathsf{pw}', e_2)$, retrieve item $(X_1 || X_2, x_1 || x_2) \in \mathcal{DL}$, $Z_1 := Y^{x_1}$,
        $Z_2 := Y^{x_2}$, $\mathsf{key} \leftarrow \mathsf{H}(sid, Z_1, Z_2, \mathsf{pw}')$, send $(\mathsf{CorruptKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{pake}}$.
    In other cases: send $(\mathsf{FreshKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$ to $\mathcal{F}_{\mathsf{pake}}$.
```

On Ideal Ciphers and Random Oracles

```
on $\mathsf{E}_1(\mathsf{pw}, X_1 || X_2)$ from $\mathcal{A}$:
    If $\exists(\mathsf{pw}, X_1 || X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$: return $e_1$.
    Otherwise: $e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_{\mathsf{IC}_1}$, $\mathcal{L}_{\mathsf{IC}_1} := \mathcal{L}_{\mathsf{IC}_1} \cup \{(\mathsf{pw}, X_1 || X_2, e_1, enc)\}$, $\mathcal{T}_{\mathsf{IC}_1} := \mathcal{T}_{\mathsf{IC}_1} \cup \{e_1\}$, return $e_1$.
on $\mathsf{D}_1(\mathsf{pw}, e_1)$ from $\mathcal{A}$:
    If $\exists(\mathsf{pw}, X_1 || X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$: return $X_1 || X_2$.
    Otherwise: $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$, $X_1 := g^{x_1}$, $X_2 := g^{x_2}$, $\mathcal{L}_{\mathsf{IC}_1} := \mathcal{L}_{\mathsf{IC}_1} \cup \{(\mathsf{pw}, X_1 || X_2, e_1, dec)\}$, $\mathcal{DL} :=$
    $\mathcal{DL} \cup \{(X_1 || X_2, x_1 || x_2)\}$, return $X_1 || X_2$.
on $\mathsf{E}_2(\mathsf{pw}, Y)$ from $\mathcal{A}$:
    If $\exists(\mathsf{pw}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$: return $e_2$.
    Otherwise: $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_{\mathsf{IC}_2}$, $\mathcal{L}_{\mathsf{IC}_2} := \mathcal{L}_{\mathsf{IC}_2} \cup \{(\mathsf{pw}, Y, e_2, enc)\}$, $\mathcal{T}_{\mathsf{IC}_2} := \mathcal{T}_{\mathsf{IC}_2} \cup \{e_2\}$, return $e_2$.
on $\mathsf{D}_2(\mathsf{pw}, e_2)$ from $\mathcal{A}$:
    If $\exists(\mathsf{pw}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$: return $Y$.
    Otherwise: $y \xleftarrow{\$} \mathbb{Z}_q$, $Y := g^y$, $\mathcal{L}_{\mathsf{IC}_2} := \mathcal{L}_{\mathsf{IC}_2} \cup \{(\mathsf{pw}, Y, e_2, dec)\}$, $\mathcal{DL} := \mathcal{DL} \cup \{(Y, y)\}$, return $Y$.
on $\mathsf{H}(\mathsf{C}, \mathsf{S}, e_1, e_2, Z_1, Z_2, \mathsf{pw})$ from $\mathcal{A}$:
    $sid := \mathsf{C} || \mathsf{S} || e_1 || e_2$.
    If $\exists(sid, Z_1, Z_2, \mathsf{pw}, \mathsf{key}) \in \mathcal{L}_{\mathsf{H}}$ for some $\mathsf{key}$: return $\mathsf{key}$.
    Otherwise: $\mathsf{key} \xleftarrow{\$} \mathcal{K}$, record $(sid, Z_1, Z_2, \mathsf{pw}, \mathsf{key})$ in $\mathcal{L}_{\mathsf{H}}$, and return $\mathsf{key}$.
```

Fig. 4. Simulator Sim for 2DH-EKE in the proof of Theorem 2.

- On $\mathsf{E}_1(\mathsf{pw}, X_1||X_2)$: If there exists $(\mathsf{pw}, X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$, return e_1. Otherwise, $e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_{\mathsf{IC}_1}$, add $(\mathsf{pw}, X_1||X_2, e_1, enc)$ in $\mathcal{L}_{\mathsf{IC}_1}$, add e_1 in $\mathcal{T}_{\mathsf{IC}_1}$, and return e_1. Here "*enc*" indicates that the record is created in encryption.
- On $\mathsf{D}_1(\mathsf{pw}, e_1)$: If there exists $(\mathsf{pw}, X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$, return $X_1||X_2$. Otherwise, $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$, $X_1 := g^{x_1}$, $X_2 := g^{x_2}$, add $(\mathsf{pw}, X_1||X_2, e_1, dec)$ in $\mathcal{L}_{\mathsf{IC}_1}$, add $(X_1||X_2, x_1||x_2)$ in $\mathcal{DL}$, and return $X_1||X_2$. Here "*dec*" indicates that the record is created in decryption.
- On $\mathsf{E}_2(\mathsf{pw}, Y)$: If there exists $(\mathsf{pw}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$, return e_2. Otherwise, $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_{\mathsf{IC}_2}$, add $(\mathsf{pw}, Y, e_2, enc)$ in $\mathcal{L}_{\mathsf{IC}_2}$, add e_2 in $\mathcal{T}_{\mathsf{IC}_2}$, and return e_2.
- On $\mathsf{D}_2(\mathsf{pw}, e_2)$: If there exists $(\mathsf{pw}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$, return Y. Otherwise, $y \xleftarrow{\$} \mathbb{Z}_q$, $Y := g^y$, add $(\mathsf{pw}, Y, e_2, dec)$ in $\mathcal{L}_{\mathsf{IC}_2}$, add (Y, y) in $\mathcal{DL}$, and return Y.
- On $\mathsf{H}(\mathsf{C}, \mathsf{S}, e_1, e_2, Z_1, Z_2, \mathsf{pw})$: Let $sid := \mathsf{C}||\mathsf{S}||e_1||e_2$. If there exists $(sid, Z_1, Z_2, \mathsf{pw}, \mathsf{key}) \in \mathcal{L}_{\mathsf{H}}$, return key. Otherwise, $\mathsf{key} \xleftarrow{\$} \mathcal{K}$, add $(sid, Z_1, Z_2, \mathsf{pw}, \mathsf{key})$ in $\mathcal{L}_{\mathsf{H}}$ and return key.

According to the ideal functionality of ideal ciphers, we know that distinct inputs of E_1 (and E_2) collide to the same ciphertext with probability $1/|\mathcal{E}_1|$ (and $1/|\mathcal{E}_2|$). By union bound, we have

$$|\Pr[\textbf{Game 1} \Rightarrow 1] - \Pr[\textbf{Game 0} \Rightarrow 1]| \le \frac{Q_{ic}^2}{|\mathcal{E}_1|} + \frac{Q_{ic}^2}{|\mathcal{E}_2|},$$

where Q_{ic} denotes the maximum number of IC queries.

Game 2. (Randomize keys for passively attacked instances.) In this game, for any session, if $\mathcal{A}$ only eavesdrops on the protocol instance, then Sim returns a random key instead of the real session key (the hash value of H). More precisely, **Game 2** is changed as follows.

(1) If server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ is linked to a good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$, then Sim generates a random session key for $(\mathsf{S}^{(j)}, iid^{(j)})$.

(2) If a good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ and a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ are linked to each other, and $(\mathsf{S}^{(j)}, iid^{(j)})$ has already been assigned with a random key, then Sim copies the key as the session key for $(\mathsf{C}^{(i)}, iid^{(i)})$.

Define bad_1 as the event that there exists a passively attacked session w.r.t. a good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ and a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$, and $\mathcal{A}$ ever asks a hash query on $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \hat{Z}_1, \hat{Z}_2, \hat{\mathsf{pw}})$ such that

$$(\hat{Z}_1, \hat{Z}_2) = \mathsf{2DH}(\mathsf{D}_1(\hat{\mathsf{pw}}, e_1), \mathsf{D}_2(\hat{\mathsf{pw}}, e_2)),$$

where e_1 and e_2 are the transcripts, and $\hat{\mathsf{pw}}$ is the correct password pre-shared between them.

Obviously $\mathcal{A}$ will not detect the change in **Game 2** unless bad_1 happens. We show that if bad_1 happens, then we can construct an algorithm $\mathcal{B}_1$ to solve the strong 2DH problem. Due to the page limitation, we provide the reduction in our full version [34]. Consequently we have

$$|\Pr[\textbf{Game 2} \Rightarrow 1] - \Pr[\textbf{Game 1} \Rightarrow 1]| \le \mathsf{Adv}^{\mathsf{st2DH}}_{\mathbb{G}, \mathcal{B}_1}(\lambda).$$

Game 3. (Randomize simulated messages.) In this game, Sim directly samples random messages to simulate the transcripts e_1 and e_2, and postpones the usage of ideal ciphers $(\mathsf{E}_1, \mathsf{D}_1)$ and $(\mathsf{E}_2, \mathsf{D}_2)$ until necessary (like the generation of session keys). More precisely, **Game 3** is now simulated by Sim as follows.

– For the simulation of a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ generating the first message e_1, Sim chooses a random $e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_{\mathsf{IC}_1}$ (without any encryption) as the output message and adds e_1 in $\mathcal{T}_{\mathsf{IC}_1}$.
– For the simulation of a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ generating the second message e_2 and the session key, Sim chooses a random $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_{\mathsf{IC}_2}$ (without any encryption) as the output message and adds e_2 in $\mathcal{T}_{\mathsf{IC}_2}$. Let e_1 be the message that $\mathsf{S}^{(j)}$ has received .
  - If $(\mathsf{S}^{(j)}, iid^{(j)})$ is linked to some good client instance, then the session key is set to be random, just like **Game 2**.
  - If $(\mathsf{S}^{(j)}, iid^{(j)})$ is not linked to any good client instance, then Sim invokes $Y \leftarrow \mathsf{D}_2(\hat{\mathsf{pw}}, e_2)$ by sampling $y \xleftarrow{\$} \mathbb{Z}_q$, computing $Y := g^y$ and adding $(\hat{\mathsf{pw}}, Y, e_2, dec)$ to $\mathcal{L}_{\mathsf{IC}_2}$. The session key is generated by $\mathsf{key} \leftarrow \mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \mathsf{2DH}(\mathsf{D}_1(\hat{\mathsf{pw}}, e_1), Y), \hat{\mathsf{pw}})$ with the knowledge of y, where $\mathsf{C}^{(i)}$ is the intended partner of $(\mathsf{S}^{(j)}, iid^{(j)})$ and $\hat{\mathsf{pw}}$ is the (correct) password. In this way, the session key is the same hash value as that in **Game 2**.

– For the simulation of a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ that sends e_1 out and receives e_2, if $(\mathsf{C}^{(i)}, iid^{(i)})$ is bad or e_2 was adversarially generated, then Sim invokes $(X_1, X_2) \leftarrow \mathsf{D}_1(\mathsf{pw}, e_1)$ by sampling $x_1, x_1, \xleftarrow{\$} \mathbb{Z}_q$, computing $X_1 := g^{x_1}$, $X_2 := g^{x_2}$ and adding $(\mathsf{pw}, X_1 || X_2, e_1, dec)$ to $\mathcal{L}_{\mathsf{IC}_1}$. The session key is generated as $\mathsf{key} \leftarrow \mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \mathsf{2DH}(X_1, X_2, \mathsf{D}_2(\mathsf{pw}, e_2)), \mathsf{pw})$ with the knowledge of x_1, x_2, where $\mathsf{S}^{(j)}$ is the intended partner of $(\mathsf{C}^{(i)}, iid^{(i)})$ and pw is the (possible incorrect) password used in this instance. In this way, the session key is the same hash value as that in **Game 2**.

Recall that in **Game 2**, the transcripts e_1 and e_2 are randomly distributed via the simulation of E_1 and E_2, so they have the same distribution as that in **Game 3**. As shown above, the generation of all session keys in **Game 3** is also the same as that in **Game 2**. Therefore, we have

$$\Pr[\textbf{Game 3} \Rightarrow 1] = \Pr[\textbf{Game 2} \Rightarrow 1].$$

Game 4. (Randomize keys for actively attacked server/client instances in case of incorrect password guesses.) In **Game 4**, the simulator further changes the session key generation of server/client instances.

For any server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ that receives e_1, let $\mathsf{C}^{(i)}$ be its intended partner and $\mathsf{pw}(= \hat{\mathsf{pw}})$ be the (correct) password used in this instance. Sim generates the session key for it in the following way.

Case (S.1). If $(\mathsf{S}^{(j)}, iid^{(j)})$ is linked to some good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$, then Sim generates a random key for $(\mathsf{S}^{(j)}, iid^{(j)})$, just as that in **Game 3**.

Case (S.2). $(\mathsf{S}^{(j)}, iid^{(j)})$ is not linked to any good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$. We further divide it into the following two subcases.

Case (S.2.1). If there exists a record $(\mathsf{pw}' = \mathsf{pw}, X_1||X_2, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$, then Sim sets $\mathsf{key} \leftarrow \mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \mathsf{2DH}(X_1, X_2, \mathsf{D}_2(\mathsf{pw}, e_2)), \mathsf{pw})$ as the session key, just like that in **Game 3**. Note that there exists at most one such record in $\mathcal{L}_{\mathsf{IC}_1}$, since E_1 is simulated in a collision-free way.

Case (S.2.2). If there does not exist a record $(\mathsf{pw}' = \mathsf{pw}, X_1||X_2, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$, then Sim generates a random key for $(\mathsf{S}^{(j)}, iid^{(j)})$.

For any client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ that sends e_1 out and receives e_2, let $\mathsf{S}^{(j)}$ be the intended partner and pw be the (possibly incorrect) password used in this instance. Sim generates the session key for it in the following way.

Case (C.1). If $(\mathsf{C}^{(i)}, iid^{(i)})$ and some server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ are linked to each other, and $(\mathsf{C}^{(i)}, iid^{(i)})$ is good, then Sim assigns the same random session key of $(\mathsf{S}^{(j)}, iid^{(j)})$ to $(\mathsf{C}^{(i)}, iid^{(i)})$, just as that in **Game 3**.

Case (C.2). If $(\mathsf{C}^{(i)}, iid^{(i)})$ is not linked to any server instance, or $(\mathsf{C}^{(i)}, iid^{(i)})$ is bad. We further divide it into the following two subcases.

Case (C.2.1). If there exists a record $(\mathsf{pw}' = \mathsf{pw}, Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$, then Sim sets $\mathsf{key} \leftarrow \mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \mathsf{2DH}(\mathsf{D}_1(\mathsf{pw}, e_1), Y), \mathsf{pw})$ as the session key, just like that in **Game 3**. Note that there exists at most one such record in $\mathcal{L}_{\mathsf{IC}_2}$, since E_2 is simulated in a collision-free way.

Case (C.2.2). If there does not exist a record $(\mathsf{pw}' = \mathsf{pw}, Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$, then Sim generates a random key for $(\mathsf{C}^{(i)}, iid^{(i)})$.

Note that the differences between **Game 3** and **Game 4** lie in Cases (S.2.2) and (C.2.2), since in **Game 3** the session keys are the hash values (rather than random elements) in Cases (S.2.2) and (C.2.2.).

We define bad_2 as the event that there exists a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ in Case (S.2.2), or a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ in Case (C.2.2), and $\mathcal{A}$ ever asks a hash query on $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \hat{Z}_1, \hat{Z}_2, \mathsf{pw})$ such that

$$(\hat{Z}_1, \hat{Z}_2) = \mathsf{2DH}(\mathsf{D}_1(\mathsf{pw}, e_1), \mathsf{D}_2(\mathsf{pw}, e_2)),$$

where e_1 and e_2 are the transcripts w.r.t. $(\mathsf{S}^{(j)}, iid^{(j)})$ or $(\mathsf{C}^{(i)}, iid^{(i)})$, and pw is the password used in this instance.

Obviously **Game 4** and **Game 3** are the same unless bad_2 happens. We show that if bad_2 happens, then we can construct a reduction algorithm $\mathcal{B}_2$ to solve the strong 2DH problem. Due to the page limitation, we provide the reduction in our full version [34].

$$|\Pr[\textbf{Game 4} \Rightarrow 1] - \Pr[\textbf{Game 3} \Rightarrow 1]| \leq \mathsf{Adv}^{\mathsf{st2DH}}_{\mathbb{G},\mathcal{B}_2}(\lambda) + 2^{-\Omega(\lambda)}.$$

Now in **Game 4**, Sim does not use pw any more, except the case of session key generation when the adversary $\mathcal{A}$ correctly guesses the password pw and actively engages into a client/server instance, i.e., there exists a record

$(\mathsf{pw}, X_1||X_2, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$ or $(\mathsf{pw}, Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$. Now we are ready to introduce the complete simulator in Fig. 4, which helps us stepping to the ideal experiment $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$.

Game 5. (Use $\mathcal{F}_{\mathsf{pake}}$ interfaces.) In the final game we introduce the ideal functionality $\mathcal{F}_{\mathsf{pake}}$. By using interfaces to interact with $\mathcal{F}_{\mathsf{pake}}$, the simulator Sim can perfectly simulates **Game 4** as follows.

– It simulates $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$, and H as described in **Game 4**.
– When Sim receives $(\mathsf{NewClient}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, b)$ from $\mathcal{F}_{\mathsf{pake}}$, it marks this instance as $\mathsf{correct\text{-}pw}$ if $b = 1$, indicating that $\mathsf{C}^{(i)}$ inputs the correct password in this client instance. Meanwhile, Sim chooses a random $e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_{\mathsf{IC}_1}$ as the output message and adds e_1 in $\mathcal{T}_{\mathsf{IC}_1}$.
– When server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ receives e_1 and $(\mathsf{NewServer}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)})$ from $\mathcal{F}_{\mathsf{pake}}$, Sim chooses a random $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_{\mathsf{IC}_2}$ as the output message and adds e_2 in $\mathcal{T}_{\mathsf{IC}_2}$. Meanwhile, it sets the session identity to be $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||e_1||e_2$ and checks whether $(\mathsf{S}^{(j)}, iid^{(j)})$ is linked to a good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$.
  • If it is the case, Sim allocates a random key to $(\mathsf{S}^{(j)}, iid^{(j)})$ by directly asking a query $(\mathsf{FreshKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{pake}}$. According to the definition of $\mathsf{FreshKey}$ interface, this performs identically as that in **Game 4**.
  • Otherwise, Sim checks whether there exists a record $(\mathsf{pw}', \cdot, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$. If such a record exists, Sim issues a TestPW query $(\mathsf{TestPW}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{pw}')$ to ask $\mathcal{F}_{\mathsf{pake}}$ whether $\mathsf{pw}' = \mathsf{pw}$, where pw is the (correct) password used in $(\mathsf{S}^{(j)}, iid^{(j)})$.
    ∗ If the record exists and $\mathcal{F}_{\mathsf{pake}}$ returns "correct guess" (i.e., $\mathsf{pw}' = \mathsf{pw}$), then Sim computes the session key as $\mathsf{key} \leftarrow \mathsf{H}(sid, \mathsf{2DH}(\mathsf{D}_1(\mathsf{pw}, e_1), \mathsf{D}_2(\mathsf{pw}, e_2)), \mathsf{pw})$, and allocates sid and key to $(\mathsf{S}^{(j)}, iid^{(j)})$ via a query $(\mathsf{CorruptKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{pake}}$. According to the definition of $\mathsf{CorruptKey}$ interface, the environment $\mathcal{Z}$ has the same view as that in **Game 4**.
    ∗ If the record does not exist, or $\mathcal{F}_{\mathsf{pake}}$ returns "wrong guess" (i.e., $\mathsf{pw}' \neq \mathsf{pw}$), then Sim allocates sid and a random key to $(\mathsf{S}^{(j)}, iid^{(j)})$ by asking a query $(\mathsf{FreshKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{pake}}$. According to the definition of $\mathsf{FreshKey}$, this results in the same view to the environment $\mathcal{Z}$ as that in **Game 4**.
– When client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ receives e_2, let e_1 be the message sent out and $\mathsf{S}^{(j)}$ be its intended partner. Sim sets the session identity to be $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||e_1||e_2$ and checks whether $(\mathsf{C}^{(i)}, iid^{(i)})$ and a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ are linked to each other, and $(\mathsf{C}^{(i)}, iid^{(i)})$ is marked as $\mathsf{correct\text{-}pw}$.
  • If it is the case, then sid and a random key key must have been assigned to $(\mathsf{S}^{(j)}, iid^{(j)})$. Sim assigns the same sid and key to $(\mathsf{C}^{(i)}, iid^{(i)})$ via a query $(\mathsf{CopyKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$ to $\mathcal{F}_{\mathsf{pake}}$. According to the definition of $\mathsf{CopyKey}$, this performs identically as that in **Game 4**.

- Otherwise, Sim retrieves the record $(\mathsf{pw}', Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$ if it exists, and uses the TestPW interface provided by $\mathcal{F}_{\mathsf{pake}}$ to check whether $\mathsf{pw}' = \mathsf{pw}$, where pw is the (possible incorrect) password used in $(\mathsf{C}^{(i)}, iid^{(i)})$.
  * If the record exists and $\mathcal{F}_{\mathsf{pake}}$ returns "correct guess" (i.e., $\mathsf{pw}' = \mathsf{pw}$), then Sim computes the session key as $\mathsf{key} \leftarrow \mathsf{H}(sid, \mathsf{2DH}(\mathsf{D}_1(\mathsf{pw}, e_1), \mathsf{D}_2(\mathsf{pw}, e_2)), \mathsf{pw})$, and allocates sid and key to $(\mathsf{C}^{(i)}, iid^{(i)})$ via a query $(\mathsf{CorruptKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{pake}}$. According to the definition of $\mathsf{CorruptKey}$ interface, the environment $\mathcal{Z}$ has the same view as that in **Game 4**.
  * If the record does not exist, or $\mathcal{F}_{\mathsf{pake}}$ returns "wrong guess" (i.e., $\mathsf{pw}' \neq \mathsf{pw}$), then Sim allocates sid and a random key to $(\mathsf{C}^{(i)}, iid^{(i)})$ by asking a query $(\mathsf{FreshKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$ to $\mathcal{F}_{\mathsf{pake}}$. According to the definition of $\mathsf{FreshKey}$, this results in the same view to the environment $\mathcal{Z}$ as that in **Game 4**.

The full description of Sim is shown in Fig. 4. From the analysis above we know **Game 4** and **Game 5** are conceptually identical. Furthermore, one can easily see that **Game 5** is just the experiment in the ideal world. Therefore, we have

$$\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}} = \mathbf{Game\ 5} = \mathbf{Game\ 4}.$$

Theorem 2 follows immediately from **Game 0** to **Game 5**, and Theorem 1.

4 Asymmetric PAKE with Optimal Tightness in the UC Framework

4.1 UC Framework for aPAKE

In aPAKE, the server stores a password file (usually a hash of the password) rather than the password in plain. This somehow protects the password even if the server is compromised. If the server's password file is obtained by the adversary due to compromise, the adversary can implement offline attacks to guess the password, or impersonate the server to run the aPAKE protocol with the client. However, it is still infeasible for the adversary to impersonate the client to log in the server, if it fails to find the correct password and actively engage into one protocol execution.

To capture the attacks due to server compromise[6] in the asymmetric setting, the ideal functionality $\mathcal{F}_{\mathsf{apake}}$ is augmented with more interfaces like $\mathsf{StealPWFile}$ and $\mathsf{OfflineTestPW}$, compared with $\mathcal{F}_{\mathsf{pake}}$. Meanwhile, the $\mathsf{CorruptKey}$ interface also takes into consideration the case of server compromise. Furthermore, we add a new interface Abort to deal with the case that the explicit authentication fails. The augments of $\mathcal{F}_{\mathsf{apake}}$ are shown below.

[6] In the real world, the server continues to faithfully execute protocols as normal after a compromise of password files.

Functionality $\mathcal{F}_{\mathsf{apake}}$

The functionality $\mathcal{F}_{\mathsf{apake}}$ is parameterized by a security parameter λ. It interacts with an adversary Sim and a set of parties (clients and servers) via the following queries:

Password Storage

Upon receiving a query $(\mathsf{StorePWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}})$ **from a client** $\mathsf{C}^{(i)}$ **or a server** $\mathsf{S}^{(j)}$:

If there exists a record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \cdot \rangle$, ignore this query.
Otherwise, record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}} \rangle$, mark it as fresh, and send $(\mathsf{StorePWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ to Sim.

Stealing Password File

Upon receiving a query $(\mathsf{StealPWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ **from server** $\mathsf{S}^{(j)}$:

Mark the password data record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}} \rangle$ as $\mathsf{compromised}$, and send $(\mathsf{StealPWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ to Sim.
If there is a record $\langle \mathsf{offline}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}} \rangle$, then send $\hat{\mathsf{pw}}$ to Sim.

Upon receiving a query $(\mathsf{OfflineTestPW}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}')$ **from** Sim:

If there exists a record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}} \rangle$ marked $\mathsf{compromised}$, check whether $\mathsf{pw}' = \hat{\mathsf{pw}}$: return "correct guess" if yes, and "wrong guess" otherwise.
Else, store $\langle \mathsf{offline}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}' \rangle$.

Sessions

Upon receiving a query $(\mathsf{NewClient}, iid^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$ **from a client** $\mathsf{C}^{(i)}$:

Retrieve the record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}} \rangle$. Send $(\mathsf{NewClient}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw} = \hat{\mathsf{pw}}?)$ to Sim. Record $(\mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$ and mark it as fresh.
In this case, $\mathsf{S}^{(j)}$ is called the intended partner of $(\mathsf{C}^{(i)}, iid^{(i)})$.

Upon receiving a query $(\mathsf{NewServer}, iid^{(j)}, \mathsf{C}^{(i)})$ **from a server** $\mathsf{S}^{(j)}$:

Retrieve the record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}} \rangle$. Send $(\mathsf{NewServer}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)})$ to Sim. Set $\mathsf{pw} = \hat{\mathsf{pw}}$, record $(\mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)}, \mathsf{pw})$ and mark it as fresh.
In this case, $\mathsf{C}^{(i)}$ is called the intended partner of $(\mathsf{S}^{(j)}, iid^{(j)})$.

Two instances $(\mathsf{C}^{(i)}, iid^{(i)})$ and $(\mathsf{S}^{(j)}, iid^{(j)})$ are said to be partnered, if there are two fresh records $(\mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$ and $(\mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)}, \mathsf{pw})$ sharing the same pw.

Active Session Attacks

Upon receiving a query $(\mathsf{TestPW}, P, iid, \mathsf{pw}')$ **from** Sim:

If there is a fresh record $(P, iid, \cdot, \mathsf{pw})$:
- If $\mathsf{pw}' = \mathsf{pw}$, mark the record $\mathsf{compromised}$ and reply to Sim with "correct guess".
- If $\mathsf{pw}' \neq \mathsf{pw}$, mark the record $\mathsf{interrupted}$ and replay with "wrong guess".

Key Generation

Upon receiving a query $(\mathsf{FreshKey}, P, iid, sid)$ **from** Sim:

If 1) there is a fresh or $\mathsf{interrupted}$ record (P, iid, Q, pw); and 2) sid has never been assigned to P's any other instance (P, iid'):
Pick a new random key k, mark the record (P, iid, Q, pw) as $\mathsf{completed}$, assign it with sid, send (iid, sid, k) to P, and record (P, Q, sid, k).

Upon receiving a query $(\mathsf{CopyKey}, P, iid, sid)$ **from** Sim:

If 1) there is a fresh record (P, iid, Q, pw) and a $\mathsf{completed}$ record $(Q, iid^*, P, \mathsf{pw})$ s.t. (P, iid) and (Q, iid^*) are partnered; and 2) sid has never been assigned to P's any other instance (P, iid'); and 3) there is a unique (Q, iid^*) that has been assigned with sid:
Retrieve the record (Q, P, sid, k), mark the record (P, iid, Q, pw) as $\mathsf{completed}$, assign it with sid, and send (iid, sid, k) to P.

Upon receiving a query $(\mathsf{CorruptKey}, P, iid, sid, k)$ **from** Sim:

If 1) sid has never been assigned to some record (P, iid'); and 2) either: 2.1) there is a $\mathsf{compromised}$ record (P, iid, Q, pw), or 2.2) there is a fresh record (P, iid, Q, pw) with P a client, and there is a $\mathsf{compromised}$ record $\langle \mathsf{file}, P, Q, \hat{\mathsf{pw}} \rangle$ such that $\mathsf{pw} = \hat{\mathsf{pw}}$:
Mark the record $(P, iid, \cdot, \mathsf{pw})$ as $\mathsf{completed}$, assign it with sid, and send (iid, sid, k) to P.

Upon receiving a query (Abort, P, iid) **from** Sim:

If P is a server: mark the record $(P, iid, \cdot, \mathsf{pw})$ as $\mathsf{completed}$, and send $(iid, \perp)$ to P.

Fig. 5. The aPAKE functionality $\mathcal{F}_{\mathsf{apake}}$ [41].

– The $\mathsf{StealPWFile}$ interface. The server may send a $\mathsf{StealPWFile}$ query to $\mathcal{F}_{\mathsf{apake}}$, indicating that the password file stored in it has been compromised by the adversary. Then $\mathcal{F}_{\mathsf{apake}}$ will pass this query message to the simulator Sim (so that Sim "simulates" a password file for the adversary).
– The $\mathsf{OfflineTestPW}$ interface. Sim issues $\mathsf{OfflineTestPW}$ together with a password guess, and $\mathcal{F}_{\mathsf{apake}}$ tests whether the guess is the pre-image of the password file and returns the test result to Sim.[7].
– The $\mathsf{CorruptKey}$ interface. Beyond the cases considered in $\mathcal{F}_{\mathsf{pake}}$, if the password file has been compromised by the adversary, Sim also assigns a key to a client instance by issuing a $\mathsf{CorruptKey}$ query[8].
– The Abort interface. If the explicit authentication from the client to the server fails, Sim assigns the session key $k = \perp$ to the server instance via an Abort query, indicating that the execution of aPAKE fails.

The functionality of $\mathcal{F}_{\mathsf{apake}}$ is shown in Fig. 5. We mainly follow the definition by Shoup in [41], which is a modified version of [19] by Gentry et al. and [25] by Hesse.

Remark 6. Perfect Forward Security [22] (PFS, a.k.a. perfect forward secrecy) requires that once a party has been corrupted at some moment, the session keys completed before the corruption remain hidden from the adversary. An aPAKE protocol with implicit authentication cannot achieve PFS due to the following reason. For the adversary who steals the password file and actively engages into one session as the client, it can always stage a (successful) offline dictionary attack, to find out the correct password, and hence obtain the "completed" session key. A canonical approach to PFS is to add an explicit authentication from the client to the server. And the server will output a specific key $k = \perp$ to terminate the session, once the authentication fails.

4.2 The 2DH-aEKE Protocol

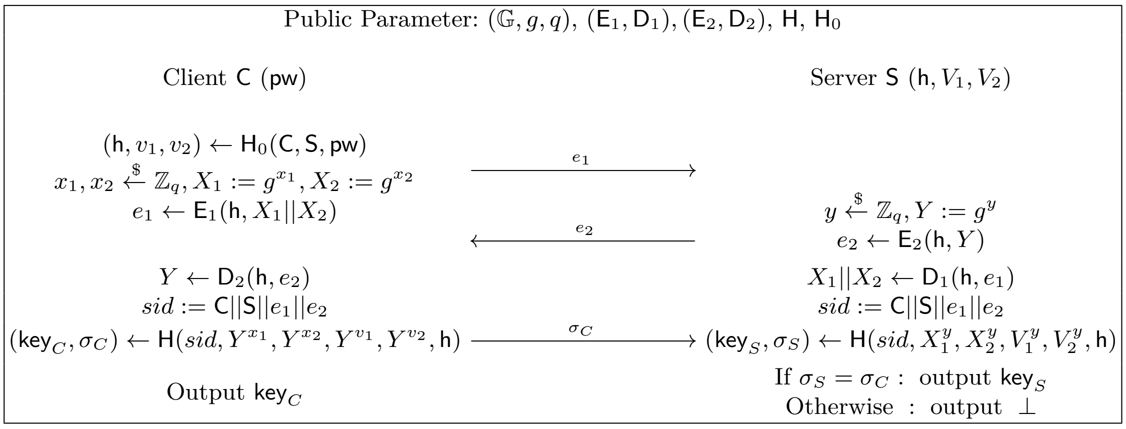
In this section, we provide an asymmetric variant of 2DH-EKE, named 2DH-aEKE. The 2DH-aEKE protocol meets the optimal reduction loss factor $L = N$, the maximum number of client-server pairs. A formal proof for the optimality is shown in Section 5.

The 2DH-aEKE protocol is shown in Fig. 6. Here $(\mathsf{E}_1, \mathsf{D}_1)$ is a symmetric encryption with key space $\mathcal{H}$, plaintext space $\mathbb{G}^2$ and ciphertext space $\mathcal{E}_1$, and $(\mathsf{E}_2, \mathsf{D}_2)$ is a symmetric encryption with key space $\mathcal{H}$, plaintext space $\mathbb{G}$ and ciphertext space $\mathcal{E}_2$. Two hash functions are defined as: $\mathsf{H} : \{0,1\}^* \mapsto \mathcal{K}$ with $\mathcal{K}$ the space of session keys, and $\mathsf{H}_0 : \{0,1\}^* \times \mathcal{PW} \mapsto \mathcal{H} \times \mathbb{Z}_q^2$. And C, S are identities of Client and Server.

[7] Such definitions seem reasonable only in a hybrid world where random oracles or ideal ciphers exist. See further discussions in [21, 41, 25].

[8] More precisely, a (corrupted) session key is assigned via $\mathsf{CorruptKey}$, if $\langle \mathsf{file}, P, Q, \hat{pw} \rangle$ is compromised and the password pw used in the client instance is correct. If pw is incorrect, then Sim would assign a random key for this client instance via $\mathsf{FreshKey}$.

In the registration stage, Server stores the password file $\mathsf{S.file[C]} := (\mathsf{h}, V_1, V_2)$, where $(\mathsf{h}, v_1, v_2) \leftarrow \mathsf{H}_0(\mathsf{C}, \mathsf{S}, \mathsf{pw})$, and $V_1 := g^{v_1}$, $V_2 := g^{v_2}$.



Public Parameter: $(\mathbb{G}, g, q)$, $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$, H, H_0
Client C (pw)
Server S (h, V_1, V_2)
$(\mathsf{h}, v_1, v_2) \leftarrow \mathsf{H}_0(\mathsf{C}, \mathsf{S}, \mathsf{pw})$
$x_1, x_2 \overset{\$}{\leftarrow} \mathbb{Z}_q, X_1 := g^{x_1}, X_2 := g^{x_2}$
$e_1 \leftarrow \mathsf{E}_1(\mathsf{h}, X_1 || X_2)$
e_1
$y \overset{\$}{\leftarrow} \mathbb{Z}_q, Y := g^y$
$e_2 \leftarrow \mathsf{E}_2(\mathsf{h}, Y)$
e_2
$Y \leftarrow \mathsf{D}_2(\mathsf{h}, e_2)$
$sid := \mathsf{C}||\mathsf{S}||e_1||e_2$
$(\mathsf{key}_C, \sigma_C) \leftarrow \mathsf{H}(sid, Y^{x_1}, Y^{x_2}, Y^{v_1}, Y^{v_2}, \mathsf{h})$
σ_C
$X_1||X_2 \leftarrow \mathsf{D}_1(\mathsf{h}, e_1)$
$sid := \mathsf{C}||\mathsf{S}||e_1||e_2$
$(\mathsf{key}_S, \sigma_S) \leftarrow \mathsf{H}(sid, X_1^y, X_2^y, V_1^y, V_2^y, \mathsf{h})$
Output key_C
If $\sigma_S = \sigma_C$: output key_S
Otherwise : output $\perp$

Fig. 6. The 2DH-aEKE protocol.

4.3 Security Analysis

Theorem 3 (Security of 2DH-aEKE). *If the st2DH assumption (equivalently, the CDH assumption) holds in $\mathbb{G}$, $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$ work as ideal ciphers, and H, H_0 work as random oracles, then the 2DH-aEKE protocol in Fig. 6 securely emulates $\mathcal{F}_{\mathsf{apake}}$. More precisely, for any PPT environment $\mathcal{Z}$ and real world adversary $\mathcal{A}$ which has access to ideal ciphers $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$ and random oracles H, H_0, there exist a PPT simulator Sim, which has access to the ideal functionality $\mathcal{F}_{\mathsf{apake}}$, and algorithms $\mathcal{B}, \mathcal{B}'$, s.t. that advantage of $\mathcal{Z}$ in distinguishing the real world running with $\mathcal{A}$ and the ideal world running with Sim is bounded by*

$$\begin{aligned}\mathsf{Adv}_{\mathsf{2DH\text{-}aEKE},\mathcal{Z}}(\lambda) \le& (N+3) \cdot \mathsf{Adv}^{\mathsf{st2DH}}_{\mathbb{G},\mathcal{B}}(\lambda) + \frac{Q_{ic}^2}{|\mathcal{E}_1|} + \frac{Q_{ic}^2}{|\mathcal{E}_2|} + \frac{Q_{\mathsf{H}_0}^2}{|\mathcal{H}|} + 2^{-\Omega(\lambda)} \\ \le& (N+3) \cdot \mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G},\mathcal{B}'}(\lambda) + 2^{-\Omega(\lambda)},\end{aligned}$$

where Q_{ic} and Q_{H_0} denote the maximum numbers of IC and H_0 queries, and N denotes the number of client-server pairs.

The proof is shown in the full version [34].

Remark 7 (On the optimal tightness of 2DH-aEKE). As we can see, the security reduction in Theorem 3 has a loss factor of N. Actually, such a loose factor is unavoidable in the scenario of aPAKE, since the correct password is committed by the hash value to the adversary in the form of password file, and it can be adaptively revealed via offline dictionary attacks (i.e., password hash queries). In Section 5 we give a formal proof to show that, the loss factor $L = N$ is essentially optimal — at least for "simple" reductions.

Nevertheless, the optimal factor N is superior to a loose factor $(Q_h \cdot N \cdot \theta)$ (the maximum numbers of hash queries, client-server pairs, and protocol executions per client-server pair, respectively). Usually there are thousands of protocol executions per user (especially for the server), and $Q_h N\theta \gg N$ in general.

5 Optimal Reduction Loss in aPAKE

In this section we show that the security loss of $L = N$ in Theorem 3 is essentially optimal, at least for "simple" reductions. Here "simple" means that the reduction algorithm runs a single copy of the adversary only once. Almost all known security reductions (for PAKE and aPAKE) are either of this type, or use the forking lemma (e.g., KHAPE-HMQV [21]).

We consider the class of DH-type aPAKE protocols defined as follows.

Definition 3 (DH-Type aPAKE Protocol). *An asymmetric PAKE protocol Π is DH-type, if it satisfies the following properties.*

1. *In the phase of password storage (registration), the server stores a password file* file *based on the pre-shared password* pw *(and some salts, perhaps).*
2. *In an execution of Π, the honest client first obtains a secret input* si *from the identities of the two parties, the password* pw *(and the first message by the server, perhaps). In this case, we say* si *is matched with the password file* file *(stored in the server).*
3. *For each* file*, there exists only one matching secret input* si*. And there exists an efficiently comutable function* $\mathsf{R}(\mathsf{file}, \mathsf{si})$*, to check whether* si *is matched with* file.
4. *There exists an efficiently computable function* F *that inputs the identities of the two parties, the password* pw*, and the password file* file *(stored in the server), and outputs the matching secret input* si.
5. *With secret input* si*, an adversary can impersonate the client to communicate with the server and compute the session key.*

We take 2DH-aEKE protocol in Fig. 6 as an example, to show how it satisfies the definition of DH-type aPAKE protocol.

1. Let pw be the password shared between C and S. The password file stored in S is $\mathsf{file} = (\mathsf{h}, V_1, V_2)$.
2. In the execution, C first obtains the secret input $(\mathsf{h}, v_1, v_2) \leftarrow \mathsf{H}_0(\mathsf{C}, \mathsf{S}, \mathsf{pw})$.
3. For each $\mathsf{file} = (\mathsf{h}, V_1, V_2)$, there exists only one matching $\mathsf{si} = (\mathsf{h}, v_1, v_2)$. And the matching relation can be efficiently verified.
4. Given identities C, S, pw, and $\mathsf{file} = (\mathsf{h}, V_1, V_2)$, the secret input si can be efficiently obtained by computing $\mathsf{H}_0(\mathsf{C}, \mathsf{S}, \mathsf{pw})$.
5. The last property is self-evident.

Apart from 2DH-aEKE, a large number of existing aPAKE protocols, including KC-SPAKE2+ [41], KHAPE-HMQV [21], aEKE-HMQV and OKAPE-HMQV [39], fall into the DH-type class.

Definition 4 (Simple Reduction). *A simple reduction $\mathcal{R}$ to a problem class $\mathcal{P}$ interacts with an adversary/environment $\mathcal{Z}$ as follows.*

1. *$\mathcal{R}$ receives a problem instance $P \in \mathcal{P}$ from its own challenger, it also has access to an oracle $\mathcal{O}$ provided by the challenger.*
2. *$\mathcal{R}$ randomly samples a bit $\beta \xleftarrow{\$} \{0,1\}$. If $\beta = 0$, then $\mathcal{R}$ simulates the real world running for $\mathcal{Z}$. And if $\beta = 1$, then $\mathcal{R}$ simulates the ideal world running for $\mathcal{Z}$.*
3. *$\mathcal{R}$ outputs its solution s.*

We say $\mathcal{R}$ is a simple $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, \epsilon_{\mathcal{Z}})$-reduction, if it runs in time at most $t_{\mathcal{R}}$, and for any adversary/environment $\mathcal{Z}$ with distinguishing advantage $\epsilon_{\mathcal{Z}}$, the output s is a solution to P with probability at least $\epsilon_{\mathcal{R}}$.

The specification of oracle $\mathcal{O}$ depends on the problem class $\mathcal{P}$ (and of cause $\mathcal{O}$ can be defined as NULL). In this paper we consider the strong twin DH problem, where a problem instance is $P = (\bar{X}_1, \bar{X}_2, \bar{Y})$, and $\mathcal{O}$ takes (Y, Z_1, Z_2) as inputs and outputs whether $(Z_1, Z_2) = \mathsf{2DH}(\bar{X}_1, \bar{X}_2, Y)$.

Theorem 4. *Let Π be a DH-type aPAKE protocol, and $\mathcal{K}$ be the session key space of Π. For any simple $(t_{\mathcal{R}}, \epsilon_{\mathcal{R}}, 1 - 1/|\mathcal{K}|)$-reduction $\mathcal{R}$ from the security of Π defined in Subsec. 4.1 to the hardness of $\mathcal{P}$, there exists a meta-reduction algorithm $\mathcal{M}$ that solves $\mathcal{P}$ in time $t_{\mathcal{M}}$ and with success probability $\epsilon_{\mathcal{M}}$, such that $t_{\mathcal{M}} \approx N \cdot t_{\mathcal{R}}$, and*

$$|\epsilon_{\mathcal{R}} - \epsilon_{\mathcal{M}}| \leq 1/N,$$

where N denotes the total number of client-server pairs.

The proof is shown in the full version [34] due to the page limitation.

From the inequality $|\epsilon_{\mathcal{R}} - \epsilon_{\mathcal{M}}| \leq 1/N$ we know $\epsilon_{\mathcal{M}} \geq \epsilon_{\mathcal{R}} - 1/N$. Namely, even with a "perfect" adversary $\mathcal{Z}$ whose advantage is overwhelming, the success probability $\epsilon_{\mathcal{R}}$ of $\mathcal{R}$ cannot significantly exceed $1/N$, as otherwise there exists an efficient algorithm $\mathcal{M}$ against the hard problem $\mathcal{P}$ (e.g., the strong 2DH problem). This implies that the reduction of $\mathcal{R}$ leads to a loss factor at least N.

6 Tight Security for 2DH-SPAKE2 in the Relaxed UC Framework

In [1], Abdalla et al. relaxed the definition of PAKE functionality to a so-called lazy-extraction PAKE (lePAKE), and proved some widely used PAKE protocols, like SPEKE [29], SPAKE2 [4], and TBPEKE [36], are secure under this relaxed model. We provide the definition of lazy-extraction UC PAKE functionality $\mathcal{F}_{\textsf{le-pake}}$ in our full version [34]. Informally, $\mathcal{F}_{\textsf{le-pake}}$ allows the adversary/simulator in the ideal world to postpone its password guess until *after* the session is completed.

In this section, we show how our technique can be extended to get tightly secure and ideal cipher-free protocols in the relaxed UC framework. We take 2DH-SPAKE2 (Fig. 7) as an example. Here randomly sampled (M_1, M_2, N_1, M_2) servers as the common reference string (CRS), and hash function H is defined as: $\mathsf{H} : \{0,1\}^* \mapsto \mathcal{K}$ with $\mathcal{K}$ the space of session keys. C, S are identities of Client and Server.

Public Parameter: $(\mathbb{G}, g, q), M_1, M_2, N_1, N_2, \mathsf{H}$

$$\begin{array}{ccc}
\text{Client } \mathsf{C}\ (\mathsf{pw}) & & \text{Server } \mathsf{S}\ (\mathsf{pw}) \\
x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q, X_1 := g^{x_1}, X_2 := g^{x_2} & \xrightarrow{X_1^*, X_2^*} & y_1, y_2 \xleftarrow{\$} \mathbb{Z}_q, Y_1 := g^{y_1}, Y_2 := g^{y_2} \\
X_1^* := X_1 \cdot M_1^{\mathsf{pw}}, X_2^* := X_2 \cdot M_2^{\mathsf{pw}} & \xleftarrow{Y_1^*, Y_2^*} & Y_1^* := Y_1 \cdot N_1^{\mathsf{pw}}, Y_2^* := Y_2 \cdot N_2^{\mathsf{pw}} \\
Z_{1,1} := (\frac{Y_1^*}{N_1^{\mathsf{pw}}})^{x_1}, Z_{1,2} := (\frac{Y_2^*}{N_2^{\mathsf{pw}}})^{x_1} & & Z_{1,1} := (\frac{X_1^*}{M_1^{\mathsf{pw}}})^{y_1}, Z_{1,2} := (\frac{X_1^*}{M_1^{\mathsf{pw}}})^{y_2} \\
Z_{2,1} := (\frac{Y_1^*}{N_1^{\mathsf{pw}}})^{x_2}, Z_{2,2} := (\frac{Y_2^*}{N_2^{\mathsf{pw}}})^{x_2} & & Z_{2,1} := (\frac{X_2^*}{M_2^{\mathsf{pw}}})^{y_1}, Z_{2,2} := (\frac{X_2^*}{M_2^{\mathsf{pw}}})^{y_2} \\
sid := \mathsf{C}||\mathsf{S}||X_1^*||X_2^*||Y_1^*||Y_2^* & & sid := \mathsf{C}||\mathsf{S}||X_1^*||X_2^*||Y_1^*||Y_2^* \\
\text{Output } \mathsf{key}_C \leftarrow \mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}) & & \text{Output } \mathsf{key}_S \leftarrow \mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})
\end{array}$$

Fig. 7. The 2DH-SPAKE2 Protocol.

Theorem 5 (Security of 2DH-SPAKE2). *If the CDH assumption holds in* $\mathbb{G}$, H *works as a random oracle, then the 2DH-SPAKE2 protocol in Fig. 7 securely emulates* $\mathcal{F}_{\textsf{le-pake}}$. *More precisely, for any PPT environment* $\mathcal{Z}$ *and real world adversary* $\mathcal{A}$ *which has access to random oracle* H, *there exist a PPT simulator* Sim, *which has access to the ideal functionality* $\mathcal{F}_{\textsf{le-pake}}$, *and an algorithm* $\mathcal{B}$, *s.t. that advantage of* $\mathcal{Z}$ *in distinguishing the real world running with* $\mathcal{A}$ *and the ideal world running with* Sim *is bounded by*

$$\mathsf{Adv}_{\textsf{2DH-SPAKE2},\mathcal{Z}}(\lambda) \leq 3\mathsf{Adv}_{\mathbb{G},\mathcal{B}}^{\mathsf{CDH}}(\lambda) + 2^{-\Omega(\lambda)}.$$

The proof is shown in the full version [34].

Note that the technique can be further used to extend PAKE protocol PPK [35] to 2DH-PPK, that achieves tight security in the relaxed UC framework. We omit the details due to the similarity.

Acknowledgments. We would like to thank the anonymous reviewers for their constructive comments, especially on the perfect forward security of 2DH-aEKE.

Shengli Liu and Xiangyu Liu were partially supported by National Natural Science Foundation of China (NSFC No. 61925207), Guangdong Major Project of Basic and Applied Basic Research (2019B030302008), and the National Key R&D Program of China under Grant 2022YFB2701500. Shuai Han was partially supported by National Natural Science Foundation of China (Grant No. 62002223), Shanghai Sailing Program (20YF1421100), Young Elite Scientists Sponsorship Program by China Association for Science and Technology (YESS20200185), and Ant Group through CCF-Ant Research Fund (CCF-AFSG RF20220224). Dawu Gu is partially supported by the National Key Research and Development Project (Grant No. 2020YFA0712302).

References

[1] Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. In: Advances in Cryptology - CRYPTO 2020. vol. 12170, pp. 278–307. Springer (2020)

[2] Abdalla, M., Barbosa, M., Rønne, P.B., Ryan, P.Y.A., Sala, P.: Security characterization of J-PAKE and its variants. IACR Cryptol. ePrint Arch. p. 824

[3] Abdalla, M., Haase, B., Hesse, J.: Security analysis of cpace. In: Advances in Cryptology - ASIACRYPT 2021. vol. 13093, pp. 711–741. Springer (2021)

[4] Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) Topics in Cryptology - CT-RSA 2005. vol. 3376, pp. 191–208. Springer (2005)

[5] Anderson, T.: Local-use ipv4/ipv6 translation prefix. RFC **8215**, 1–7 (2017)

[6] Becerra, J., Iovino, V., Ostrev, D., Sala, P., Skrobot, M.: Tightly-secure PAK(E). In: Capkun, S., Chow, S.S.M. (eds.) Cryptology and Network Security. vol. 11261, pp. 27–48. Springer (2017)

[7] Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Advances in Cryptology - EUROCRYPT 2000. vol. 1807, pp. 139–155. Springer (2000)

[8] Bellovin, S.M., Merritt, M.: Encrypted key exchange: password-based protocols secure against dictionary attacks. In: 1992 IEEE Computer Society Symposium on Research in Security and Privacy. pp. 72–84. IEEE Computer Society (1992)

[9] Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In: CCS '93. pp. 244–250. ACM (1993)

[10] Benhamouda, F., Blazy, O., Chevalier, C., Pointcheval, D., Vergnaud, D.: New techniques for sphfs and efficient one-round PAKE protocols. In: Advances in Cryptology - CRYPTO 2013. vol. 8042, pp. 449–475. Springer (2013)

[11] Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: New models and constructions. IACR Cryptol. ePrint Arch. p. 833 (2013)

[12] Bresson, E., Chevassut, O., Pointcheval, D.: Security proofs for an efficient password-based key exchange. In: CCS 2003. pp. 241–250. ACM (2003)

[13] Bresson, E., Chevassut, O., Pointcheval, D.: New security results on encrypted key exchange. In: Public Key Cryptography - PKC 2004. vol. 2947, pp. 145–158. Springer (2004)

[14] Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA. pp. 136–145. IEEE Computer Society (2001)

[15] Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Advances in Cryptology - EUROCRYPT 2005. vol. 3494, pp. 404–421. Springer (2005)
[16] Cash, D., Kiltz, E., Shoup, V.: The twin diffie-hellman problem and applications. In: Advances in Cryptology - EUROCRYPT 2008. vol. 4965, pp. 127–145. Springer (2008)
[17] Dupont, P., Hesse, J., Pointcheval, D., Reyzin, L., Yakoubov, S.: Fuzzy password-authenticated key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018. vol. 10822, pp. 393–424. Springer (2018)
[18] Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. In: Advances in Cryptology - EUROCRYPT 2003. vol. 2656, pp. 524–543. Springer (2003)
[19] Gentry, C., MacKenzie, P.D., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Advances in Cryptology - CRYPTO 2006. vol. 4117, pp. 142–159. Springer (2006)
[20] Groce, A., Katz, J.: A new framework for efficient password-based authenticated key exchange. In: CCS 2010. pp. 516–525. ACM (2010)
[21] Gu, Y., Jarecki, S., Krawczyk, H.: KHAPE: asymmetric PAKE from key-hiding key exchange. In: Advances in Cryptology - CRYPTO 2021. vol. 12828, pp. 701–730. Springer (2021)
[22] Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J., Vandewalle, J. (eds.) EUROCRYPT 1989. vol. 434, pp. 29–37. Springer (1989)
[23] Hao, F., Ryan, P.Y.A.: J-PAKE: authenticated key exchange without PKI. Trans. Comput. Sci. **11**, 192–206 (2010)
[24] Harkins, D.: Dragonfly key exchange. RFC **7664**, 1–18 (2015)
[25] Hesse, J.: Separating symmetric and asymmetric password-authenticated key exchange. In: SCN 2020. vol. 12238, pp. 579–599. Springer (2020)
[26] Hwang, J.Y., Jarecki, S., Kwon, T., Lee, J., Shin, J.S., Xu, J.: Round-reduced modular construction of asymmetric password-authenticated key exchange. In: SCN 2018. vol. 11035, pp. 485–504. Springer (2018)
[27] ISO/IEC: Iso/iec 11770-4:2017 information technology — security techniques — key management — part 4: Mechanisms based on weak secrets, https://www.iso.org/standard/67933.html
[28] Jablon, D.P.: Strong password-only authenticated key exchange. Comput. Commun. Rev. **26**(5), 5–26 (1996)
[29] Jablon, D.P.: Extended password key exchange protocols immune to dictionary attacks. In: 6th Workshop on Enabling Technologies (WET-ICE '97). pp. 248–255. IEEE Computer Society (1997)
[30] Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: an asymmetric PAKE protocol secure against pre-computation attacks. In: Advances in Cryptology - EUROCRYPT 2018. vol. 10822, pp. 456–486. Springer (2018)
[31] Katz, J., Ostrovsky, R., Yung, M.: Efficient password-authenticated key exchange using human-memorable passwords. In: Advances in Cryptology - EUROCRYPT 2001. vol. 2045, pp. 475–494. Springer (2001)
[32] Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Theory of Cryptography - 8th Theory of Cryptography Conference, TCC 2011. vol. 6597, pp. 293–310. Springer (2011)
[33] Kwon, J.O., Sakurai, K., Lee, D.H.: One-round protocol for two-party verifier-based password-authenticated key exchange. In: CMS 2006. vol. 4237, pp. 87–96. Springer (2006)

[34] Liu, X., Liu, S., Han, S., Gu, D.: Eke meets tight security in the universally composable framework. Cryptology ePrint Archive, Paper 2023/170 (2023), https://eprint.iacr.org/2023/170

[35] Mackenzie, P.: The pak suite: Protocols for password-authenticated key exchange (12 2002)

[36] Pointcheval, D., Wang, G.: VTBPEKE: verifier-based two-basis password exponential key exchange. In: AsiaCCS 2017. pp. 301–312. ACM (2017)

[37] Rescorla, E.: The transport layer security (TLS) protocol version 1.3. RFC **8446**, 1–160 (2018)

[38] RFC: Crypto forum (cfrg), https://datatracker.ietf.org/rg/cfrg/documents/

[39] Santos, B.F.D., Gu, Y., Jarecki, S., Krawczyk, H.: Asymmetric PAKE with low computation and communication. In: EUROCRYPT 2022. vol. 13276, pp. 127–156. Springer (2022)

[40] Shin, S., Kobara, K.: Efficient augmented password-only authentication and key exchange for ikev2. RFC **6628**, 1–20 (2012)

[41] Shoup, V.: Security analysis of SPAKE2+. IACR Cryptol. ePrint Arch. p. 313 (2020)

[42] Tanwar, S., Vora, J., Kaneriya, S., Tyagi, S., Kumar, N., Sharma, V., You, I.: Human arthritis analysis in fog computing environment using bayesian network classifier and thread protocol. IEEE Consumer Electronics Magazine **9**(1), 88–94 (2020)

[43] Williams, M., Benfield, C., Warner, B., Zadka, M., Mitchell, D., Samuel, K., Tardy, P.: Magic Wormhole, pp. 253–284. Apress, Berkeley, CA (2019)

[44] Yu, J., Lian, H., Zhao, Z., Tang, Y., Wang, X.: Provably secure verifier-based password authenticated key exchange based on lattices. Adv. Comput. **120**, 121–156 (2021)