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 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) [10] 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.

[*] A preliminary version of this paper was accepted by PKC 2023 and this is the full version.

For PAKE, the server has to store all clients' passwords and once compromised, all clients are in high risk. Asymmetric PAKE (aPAKE, a.k.a. verifier-based PAKE) [11, 24] is a variant of PAKE that considers security against server compromise. In the scenario of aPAKE, 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 [10, 11], (a)PAKE has been studied extensively, and a variety of protocols have been proposed over the past decades. For example, SPEKE [35], PPK/PAK [41], SPAKE2 [4], Dragonfly [31], J-PAKE [30], KOY [38], KV [39] for PAKE, and VB-PAKE [40], OPAQUE [37], KC-SPAKE2+ [47], KHAPE [27], YLZT [50], aEKE and OKAPE [45] for aPAKE. Among these protocols, SPAKE2, J-PAKE, OPAQUE are under the process of standardization [46, 5, 34, 44]. (a)PAKE protocols have also been increasingly applied to numerous settings, including TLS [37, 43], ad hoc networks [49], and the Internet of Things [48].

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 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 [9] for PAKE and [40, 12, 13] for aPAKE) and the simulation-based security in the Universally Composable (UC) framework (see [17] for PAKE and [24] 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 (or at least have a min-entropy) for the sake of security proof, e.g., [9, 38].
– 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 [16] in the UC framework.
– PAKE with UC security implies simulation-based security of secure-channel protocols built on PAKE [17]. In contrast, it is not sure for the IND security [47].

Tight security. The security of (a)PAKE (in both the IND an the UC models) is achieved by a security reduction under proper assumptions. Security reduction transforms the ability of a successful adversary $\mathcal{A}$ to an algorithm $\mathcal{B}$ solving some well-known hard problem with almost the same running time. If $\mathcal{A}$ attacks with successful 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. [8] proved that the security of the PAK protocol [41] 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 challenge 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 [10], 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 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 only one extra round of message authentications w.r.t. communication.
4. As a byproduct, we further apply our technique to PAKE protocols like SPAKE2 [4] and PPK [41] 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 [10], and proposed the well-known EKE compiler/protocol. The security of EKE was formally proved later by Bellare et al. [9] in the IND model, and by Dupont et al. [22] in the UC framework. Most of the efficient PAKE constructions ([41, 15, 4, 14, 42], to name a few) rely on Random Oracles (RO), and they can be viewed as different variants of the classical EKE compiler [10]. There are some works [38, 23, 26, 39] 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 [26] or NIZK [39], and hence are less efficient.

Given the advantages of the UC framework over the IND model, a large amount of (a)PAKE protocols [47, 37, 27, 45] 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 [36], SPAKE2 [4], and TBPEKE [42] are UC-secure.

The only two works considering tight security of PAKE are [8] by Becerra et al., and [3] by Abdalla et al. (both of them are in the random oracle 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 [10] (Fig. 15 in Appendix D). 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, 8] did. However, [1, 8] 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 [18], 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 [18], 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 successfully engages a brute-force attack to recover the pre-image pw. In this paper, we develop our 2DH-EKE to an aPAKE protocol 2DH-aEKE, with only one extra round of message authentication codes.

2DH-aEKE inherits the idea of the generic CDH-based compiler in [33], 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 on $\mathsf{H}_0(\mathsf{pw})$ 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.

Recall that almost all previous aPAKE schemes [33, 47, 27] 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 be applied to some IC-free protocols like SPAKE2 [4] and PPK [41], 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 $\mathsf{key} = H(\mathsf{aux}, g^{x_1y_1}, g^{x_1y_2}, g^{x_2y_1}g^{x_2y_2}, \mathsf{pw})$. 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 Perfect Forward Security [28] (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. 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, propose the asymmetric variant 2DH-aEKE protocol, and prove its security. The optimal reduction loss in aPAKE is proved in Section 5. Consequently, we extend our technique to SPAKE2 to obtain 2DH-SPAKE2 in Section 6, with its proof in Appendix C. The functionalities of ideal ciphers, random oracles, and $\mathcal{F}_{\mathsf{le\text{-}pake}}$ (relaxed UC framework) are shown in Appendixes A and B, respectively. Some more protocols are shown in Appendix D.

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 $x \overset{\$}{\leftarrow} \mathcal{X}$ 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.

For the functionalities of ideal ciphers and random oracles, see Appendix A.

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*

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

In [18], 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. *[18] For any adversary $\mathcal{A}$, its advantage in solving the st2DH problem is defined as*

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

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 [18].

Theorem 1. *[18] 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 [47], which is a modified version of [17] by Canetti et al. For a full understanding of UC framework, we refer [17, 47] 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 distinguish 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 $\mathsf{S}^{(j)}$. While for a client $\mathsf{C}^{(i)}$, it is possible that $\mathsf{pw} \neq \hat{\mathsf{pw}}$ due to a mistyped/misremembered password.

Following by the definition in [47], 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 pass-

[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 $\mathcal{F}_{\mathsf{pake}}$ [47].

word 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 $\mathsf{compromised}$, 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 a session identity sid and a key for it. Here sid is determined by Sim, and it 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 terminates 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.

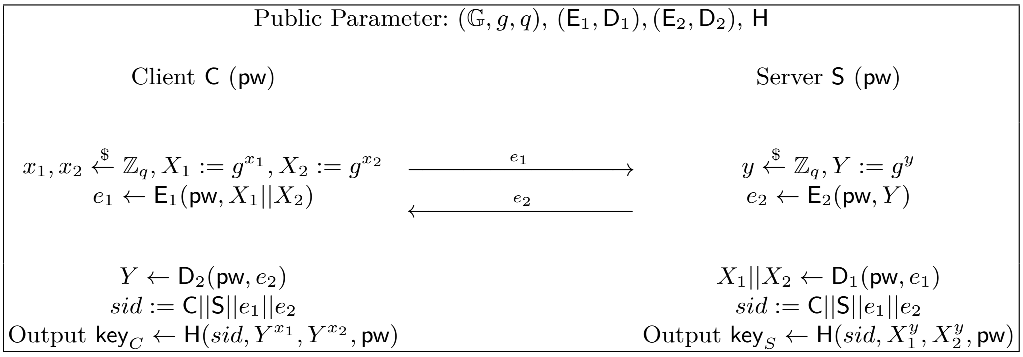
Remark 2 (Corruptions). Our PAKE framework deals with static corruptions, i.e., the adversary can corrupt some parties and get their passwords prior to the protocol execution. Note that there is a stronger model that supports adaptive corruptions, where the adversary can corrupt parties adaptively throughout the execution, and obtain not only the passwords but also the internal states. Almost all UC frameworks [17, 47] for PAKE are defined in the way of static corruptions.

3.2 The 2DH-EKE Protocol

The EKE compiler/protocol (Fig. 15 in Appendix D) was proposed by Bellovin and Merritt in [10], and formally proved later by Bellare et al. in the IND model [9], and by Dupont et al. in the UC framework [22]. 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 [27], 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 [27].

Remark 5 (Comparisons with the twin DH protocol [18] and KC-SPAKE2 [47]). Note that Cash et al. [18] 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 $H(g^{x_1y_1}, g^{x_1y_2}, g^{x_2y_1}, g^{x_2y_2})$. 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 [47], 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*

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

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, response 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** $\mathbf{0-5}$, 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

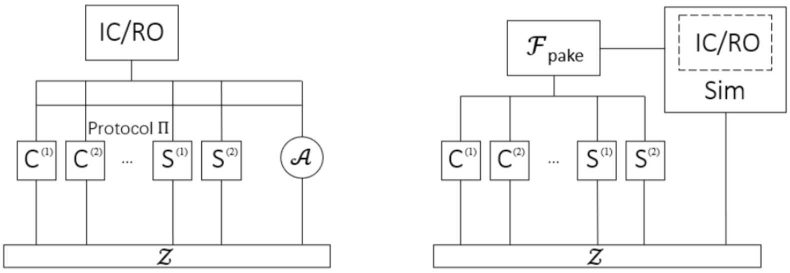


Fig. 3. The real world execution (left) and the ideal world execution (right).

$iid^{(i)}$ 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.

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

[5] The simulators in **Game 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 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 (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 correct-pw. // client $\mathsf{C}^{(i)}$ correctly inputs the password
on (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 correct-pw, $\exists(\mathsf{S}^{(j)}, \cdot, e_2) \in \mathcal{T}$, and 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 key: return key.
    Otherwise: $\mathsf{key} \xleftarrow{\$} \mathcal{K}$, record $(sid, Z_1, Z_2, \mathsf{pw}, \mathsf{key})$ in $\mathcal{L}_{\mathsf{H}}$, and return key.
```

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

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 $\mathcal{L}_{\mathsf{IC}_1}, \mathcal{L}_{\mathsf{IC}_2}, \mathcal{T}_{\mathsf{IC}_1}, \mathcal{T}_{\mathsf{IC}_2}, \mathcal{DL}, \mathcal{L}_{\mathsf{H}}$ (all initialized to be empty sets) and works as follows.

- 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

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

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.

$\mathcal{B}_1$ works as follows. It receives the 2DH challenge $(\bar{X}_1, \bar{X}_2, \bar{Y}) = (g^{\bar{x}_1}, g^{\bar{x}_2}, g^{\bar{y}})$, as well as an oracle 2DH which inputs (Y, Z_1, Z_2) and outputs whether $(Z_1, Z_2) = \mathsf{2DH}(\bar{X}_1, \bar{X}_2, Y)$. Then it simulates **Game 2** as below.

- The simulation of $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2), \mathsf{H}$ is the same as that in **Game 2**.
- For the simulation of good client instance $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ generating the first message e_1: $\mathcal{B}_1$ samples $a_{s,1}^{(i)}, a_{s,2}^{(i)} \xleftarrow{\$} \mathbb{Z}_q$, and sets $e_1 \leftarrow \mathsf{E}_1(\hat{\mathsf{pw}}, X_1 || X_2)$, where $X_1 := \bar{X}_1 g^{a_{s,1}^{(i)}} = g^{\bar{x}_1 + a_{s,1}^{(i)}}$ and $X_2 := \bar{X}_2 g^{a_{s,2}^{(i)}} = g^{\bar{x}_2 + a_{s,2}^{(i)}}$.
- For the simulation of server instance $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ generating the second message e_2 and the session key: Let e_1 be the received message. There are two cases.
  - If $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ is linked to some good instance $(\mathsf{C}^{(i)}, iid^{(i)} = t)$, it samples $b_t^{(j)} \xleftarrow{\$} \mathbb{Z}_q$, $Y := g^{\bar{y} + b_t^{(j)}}$, and sets the session key to be random.
  - Otherwise, either $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ is linked to some bad client instance, or e_1 is adversarially generated. In this case, it samples $y \xleftarrow{\$} \mathbb{Z}_q$, $Y := g^y$, and computes the session key according to the protocol specification.

  In either case, it outputs $e_2 \leftarrow \mathsf{E}_2(\hat{\mathsf{pw}}, Y)$.
- For the simulation of good client instance $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ generating the session key (after generating e_1 and receiving e_2), there are two cases.
  - If $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ and a server instance $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ are linked to each other, then $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ must have been assigned with a random key key. In this case, $\mathcal{B}_1$ assigns the same key for $(\mathsf{C}^{(i)}, iid^{(i)} = s)$.
  - If $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ is not linked to any server instance, then $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ must have received an adversarially generated message e_2. $\mathcal{B}_1$ retrieves $(\hat{\mathsf{pw}}, X_1 || X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$ and $(\hat{\mathsf{pw}}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$ ($\mathcal{B}_1$ generates the items if they do not exist), and checks whether there exists $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, Z_1, Z_2, \mathsf{pw}, \mathsf{key}) \in \mathcal{L}_{\mathsf{H}}$, such that $\mathsf{2DH}(X_1, X_2, Y) = (Z_1, Z_2)$. Namely, it uses the trapdoor information $a_{s,1}^{(i)}, a_{s,1}^{(i)}$ and the decisional oracle 2DH, to check whether

    $$\mathrm{2DH}(Y, Z_1/Y^{a_{s,1}^{(i)}}, Z_2/Y^{a_{s,2}^{(i)}}) = 1.$$

    If so, $\mathcal{B}_1$ assigns key as the session key of $(\mathsf{C}^{(i)}, iid^{(i)} = s)$. Otherwise, $\mathcal{B}_1$ randomly samples a key and "views" it as the hash output for the correct input $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \mathsf{2DH}(X_1, X_2, Y) = (?, ?), \hat{\mathsf{pw}})$, where $\mathsf{2DH}(X_1, X_2, Y) = (?, ?)$ means that the values of $\mathsf{2DH}(X_1, X_2, Y)$ are to be determined. If $\mathcal{A}$ later asks $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, Z_1, Z_2, \hat{\mathsf{pw}})$, then $\mathcal{B}_1$ checks whether $Z_1 || Z_2$ are the correct 2DH values via the decisional oracle 2DH, i.e., it checks whether

    $$\mathrm{2DH}(Y, Z_1/Y^{a_{s,1}^{(i)}}, Z_2/Y^{a_{s,2}^{(i)}}) = 1.$$

If yes, $\mathcal{B}_1$ reprograms the random oracle s.t. $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, Z_1, Z_2, \hat{\mathsf{pw}}) = \mathsf{key}$ by replacing $(?, ?)$ with (Z_1, Z_2). In this way, the view of $\mathcal{A}$ is consistent.

– The simulation of bad client instances is the same as that in **Game 2**.

Suppose that bad_1 happens w.r.t. instances $(\mathsf{C}^{(i)}, s)$ and $(\mathsf{S}^{(j)}, t)$ with transcripts $e_1||e_2$, then $\mathcal{A}$ must have asked $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \hat{Z}_1, \hat{Z}_2, \hat{\mathsf{pw}})$ s.t.

$$(\hat{Z}_1, \hat{Z}_2) = \mathsf{2DH}(g^{\bar{x}_1+a^{(i)}_{s,1}}, g^{\bar{x}_2+a^{(i)}_{s,2}}, g^{\bar{y}+b^{(j)}_t}).$$

Note that $\mathcal{B}_1$ can detect bad_1 with the help of oracle 2DH and trapdoors $a^{(i)}_{s,1}, a^{(i)}_{s,2}, b^{(j)}_t$. Then it makes use of $\hat{Z}_1||\hat{Z}_2$ to extract

$$\hat{Z}_1/g^{\bar{x}_1 b^{(j)}_t+\bar{y}a^{(i)}_{s,1}+a^{(i)}_{s,1}b^{(j)}_t} = g^{\bar{x}_1\bar{y}} \text{ and } \hat{Z}_2/g^{\bar{x}_2 b^{(j)}_t+\bar{y}a^{(i)}_{s,2}+a^{(i)}_{s,2}b^{(j)}_t} = g^{\bar{x}_2\bar{y}},$$

which is obviously the solution to the strong 2DH problem.

Therefore, we have

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

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.

$\mathcal{B}_2$ receives the 2DH challenge $(\bar{X}_1, \bar{X}_2, \bar{Y}) = (g^{\bar{x}_1}, g^{\bar{x}_2}, g^{\bar{y}})$, as well as a decisional oracle 2DH. Then it simulates **Game 4** as follows.

- The simulation of $\mathsf{E}_1, \mathsf{E}_2$, and H is the same as that in **Game 4**.
- Simulation of $\mathsf{D}_1(\mathsf{pw}', e_1)$.
  - If there exists $(\mathsf{pw}', X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$, return $X_1||X_2$.
  - Otherwise, sample $a_1, a_2 \xleftarrow{\$} \mathbb{Z}_q$, $X_1 := \bar{X}_1 g^{a_1} = g^{\bar{x}_1 + a_1}$, $X_2 := \bar{X}_2 g^{a_2} = g^{\bar{x}_2 + a_2}$, add $(\mathsf{pw}', X_1||X_2, e_1, dec)$ in $\mathcal{L}_{\mathsf{IC}_1}$, and return $X_1||X_2$.
- Simulation of $\mathsf{D}_2(\mathsf{pw}', e_2)$.
  - If there exists $(\mathsf{pw}', Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$, return Y.
  - Otherwise, if $\mathsf{D}_2(\mathsf{pw}', e_2)$ is invoked in Case (S.2.1), $\mathcal{B}_2$ samples $y \xleftarrow{\$} \mathbb{Z}_q$, $Y := g^y$, adds $(\mathsf{pw}' = \hat{\mathsf{pw}}, Y, e_2, dec)$ in $\mathcal{L}_{\mathsf{IC}_2}$, adds (Y, y) in $\mathcal{DL}$ and returns Y. Note that with overwhelming probability, there exists no record $(\cdot, \cdot, e_2, \cdot)$ in $\mathcal{L}_{\mathsf{IC}_2}$ before $\mathsf{D}_2(\hat{\mathsf{pw}}, e_2)$ is invoked in Case (S.2.1), since e_2 is randomly sampled by the server instance $(\mathsf{S}^{(j)}, iid^{(j)})$.

    In other cases, $\mathcal{B}_2$ samples $b \xleftarrow{\$} \mathbb{Z}_q$, $Y := \bar{Y} g^b = g^{\bar{y}+b}$, adds $(\mathsf{pw}', Y, e_2, dec)$ in $\mathcal{L}_{\mathsf{IC}_2}$ and returns Y.
- The simulation of transcripts e_1 and e_2 is the same as that in **Game 4**.
- The simulation of key generation for server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ is the same as that in **Game 4**. Specially in Case (S.2.1), $\mathcal{B}_2$ is able to compute the correct hash value with the knowledge of y obtained in the simulation of D_2.
- Simulation of key generation for client instance $(\mathsf{C}^{(i)}, iid^{(i)})$.
  - 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 $\mathcal{B}_2$ assigns the same random session key for it, just as that in **Game 4**.
  - Otherwise, $\mathcal{B}_2$ computes the session key with the help of the decisional oracle 2DH. More preciesly, let e_1, e_2 be the messages sent and received by $(\mathsf{C}^{(i)}, iid^{(i)})$, and pw be the password used in it. $\mathcal{B}_2$ retrieves $(\mathsf{pw}, X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$ and $(\mathsf{pw}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$ ($\mathcal{B}_2$ generates the items if they do not exist), and checks whether there exists $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, Z_1,$

$Z_2, \mathsf{pw}, \mathsf{key}) \in \mathcal{L}_{\mathsf{H}}$, such that $\mathsf{2DH}(X_1, X_2, Y) = (Z_1, Z_2)$. If so, $\mathcal{B}_2$ assigns key as the session key of $(\mathsf{C}^{(i)}, iid^{(i)})$. Otherwise, $\mathcal{B}_2$ randomly samples a key and "views" it as the hash value for the correct input $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \mathsf{2DH}(X_1, X_2, Y) = (?, ?), \mathsf{pw})$, where $\mathsf{2DH}(X_1, X_2, Y) = (?, ?)$ means that the values of $\mathsf{2DH}(X_1, X_2, Y)$ are to be determined. If $\mathcal{A}$ later asks $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, Z_1, Z_2, \mathsf{pw})$, then $\mathcal{B}_2$ checks whether $Z_1||Z_2$ are the correct 2DH values via the decisional oracle 2DH, i.e., it checks whether

$$\mathrm{2DH}(Y, Z_1/Y^{a_1}, Z_2/Y^{a_2}) = 1,$$

with the knowledge of a_1, a_2. If yes, $\mathcal{B}_2$ reprograms the random oracle s.t. $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, Z_1, Z_2, \mathsf{pw}) = \mathsf{key}$ by replacing $(?, ?)$ with (Z_1, Z_2). In this way, the view of $\mathcal{A}$ is consistent.

If bad_2 happens, then 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 $\mathcal{A}$ 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

$$(\hat{Z}_1, \hat{Z}_2) = \mathsf{2DH}(\mathsf{D}_1(\mathsf{pw}, e_1), \mathsf{D}_2(\mathsf{pw}, e_2)) = \mathsf{2DH}(g^{\bar{x}_1 + a_1}, g^{\bar{x}_2 + a_2}, g^{\bar{y} + b}).$$

Note that $\mathcal{B}_2$ can detect bad_2 with the help of oracle 2DH and trapdoors a_1, a_2, b. Then, it can make use of $\hat{Z}_1||\hat{Z}_2$ to extract

$$\hat{Z}_1/g^{\bar{x}_1 b + \bar{y} a_1 + a_1 b} \text{ and } \hat{Z}_2/g^{\bar{x}_2 b + \bar{y} a_2 + a_2 b},$$

which is the solution to the strong 2DH problem.

Therefore, we have

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

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 $e_2 \overset{\$}{\leftarrow} \mathcal{E}_2 \backslash \mathcal{T}_{\mathsf{IC}_2}$ 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 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 StealPWFile and OfflineTestPW, compared with $\mathcal{F}_{\mathsf{pake}}$. Meanwhile, the 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.

- The StealPWFile interface. The server may send a 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 OfflineTestPW interface. Sim issues 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].

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

[7] Such definitions seem reasonable only in a hybrid world where random oracles or ideal ciphers exist. See further discussions in [27, 47, 32].

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 $\mathcal{F}_{\mathsf{apake}}$ [47].

- 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 [47], which is a modified version of [24] by Gentry et al. and [32] by Hesse.

Remark 6. <u>P</u>erfect <u>F</u>orward <u>S</u>ecurity [28] (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, 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.

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

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*

[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 $\mathsf{FreshKey}$.

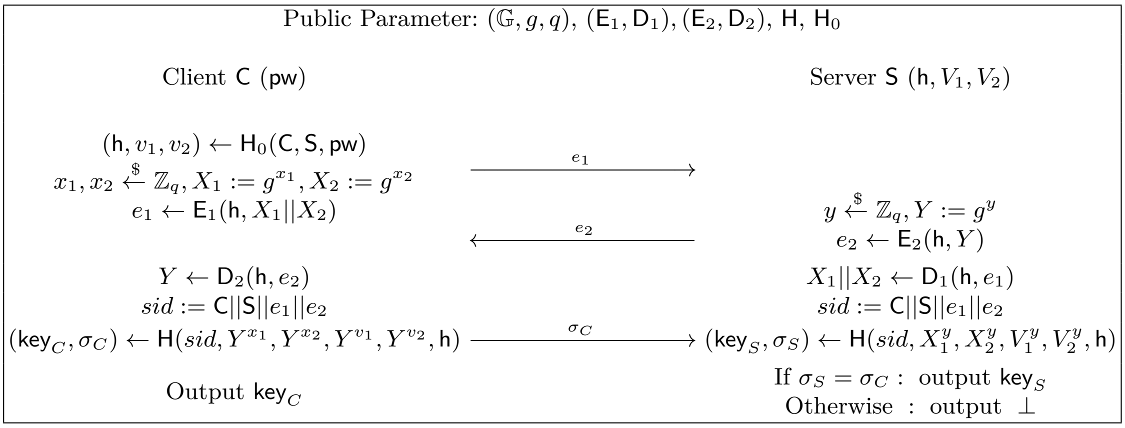

Public Parameter: $(\mathbb{G}, g, q)$, $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$, H, H_0
Client C (pw)
Server S (h, V_1, V_2)
$(\mathsf{h}, v_1, v_2) \leftarrow \mathsf{H}_0(\mathsf{C}, \mathsf{S}, \mathsf{pw})$
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q, X_1 := g^{x_1}, X_2 := g^{x_2}$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

Fig. 6. The 2DH-aEKE protocol.

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*

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

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.

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.

Proof. Similar to that in Theorem 2, the main task of the proof, is to construct a PPT simulator Sim, which has access to the ideal functionality $\mathcal{F}_{\mathsf{apake}}$ and interactions with the environment $\mathcal{Z}$, and simulates the real world 2DH-aEKE 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, response adversarial messages approximately, and simulate ideal functions $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$, and H_0, H. The functionality $\mathcal{F}_{\mathsf{apake}}$ provides information to Sim through interfaces including $\mathsf{StealPWFile}$[9], $\mathsf{OfflineTestPW}$, TestPW, $\mathsf{NewClient}$, $\mathsf{NewServer}$, $\mathsf{FreshKey}$, $\mathsf{CopyKey}$, $\mathsf{CorruptKey}$, and Abort, as defined in Fig. 5. Recall that Sim has no secret inputs (i.e., passwords).

The full description of the simulator Sim is given in Fig. 7 and 8. 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 − 6**, where **Game 0** is $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$, **Game 6** 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 as before, and use $(\mathsf{C}^{(i)}, iid^{(i)})$ (resp., $(\mathsf{S}^{(j)}, iid^{(j)})$) to specify a client (resp., a server) instance. The definitions of good/bad client instances are the same as those in Section 3. Besides, due to the explicit authentication from the client to the server in 2DH-aEKE, we redefine linked instances as follows.

Linked instances. We say that a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ is linked (resp., partially linked) to a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ (no matter good or bad), if e_1 and σ (resp., e_1) generated by $(\mathsf{C}^{(i)}, iid^{(i)})$ are/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 $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$. It is defined just like the real experiment for 2DH-aEKE in the proof of Theorem 2, except that the adversary can implement more attacks like "steal password" ($\mathsf{StealPWFile}$) and "offline attacks" ($\mathsf{OfflineTestPW}$) on the password file, and perfectly impersonates the server after compromising the password file. 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, 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. 7 and 8. 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 and H_0 are 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 $\mathcal{L}_{\mathsf{IC}_1}, \mathcal{L}_{\mathsf{IC}_2}, \mathcal{T}_{\mathsf{IC}_1}, \mathcal{T}_{\mathsf{IC}_2}, \mathcal{L}_{\mathsf{H}_0}, \mathcal{T}_{\mathsf{H}_0}, \mathcal{L}_{\mathsf{H}}, \mathcal{DL}$ (all initialized to be empty sets) and works as follows.

[9] The query $(\mathsf{StealPWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ is invoked by $\mathsf{S}^{(j)}$ under $\mathcal{Z}$'s instruction, indicating that the password data record $\langle \mathsf{file}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \cdot \rangle$ is compromised. Then Sim obtains this compromise information via a message $(\mathsf{StealPWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ from $\mathcal{F}_{\mathsf{apake}}$.

Sim maintains lists $\mathcal{L}_{\mathsf{IC}_1}, \mathcal{L}_{\mathsf{IC}_2}, \mathcal{T}_{\mathsf{IC}_1}, \mathcal{T}_{\mathsf{IC}_2}, \mathcal{L}_{\mathsf{H}_0}, \mathcal{T}_{\mathsf{H}_0}, \mathcal{L}_{\mathsf{H}}, \mathcal{T}, \mathcal{DL}$ (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 $(\mathsf{E}_2, \mathsf{D}_2)$.
- $\mathcal{L}_{\mathsf{H}_0}, \mathcal{T}_{\mathsf{H}_0}, \mathcal{L}_{\mathsf{H}}$: store records w.r.t. random oracles H_0 and H.
- $\mathcal{T}$: store messages sent by client/server instances.
- $\mathcal{DL}$: store discrete logarithms.

Password Storage

on $(\mathsf{StorePWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ from $\mathcal{F}_{\mathsf{apake}}$:

Sample $\hat{\mathsf{h}} \xleftarrow{\$} \mathcal{H} \backslash \mathcal{T}_{\mathsf{H}_0}, \hat{v}_1, \hat{v}_2 \xleftarrow{\$} \mathbb{Z}_q$, $\hat{V}_1 := g^{\hat{v}_1}$, $\hat{V}_2 := g^{\hat{v}_2}$, $\mathcal{T}_{\mathsf{H}_0} := \mathcal{T}_{\mathsf{H}_0} \cup \{\hat{\mathsf{h}}\}$, $\mathcal{DL} := \mathcal{DL} \cup \{(\hat{V}_1 || \hat{V}_2, \hat{v}_1 || \hat{v}_2)\}$, and record $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] := (\hat{\mathsf{h}}, \hat{V}_1, \hat{V}_2)$.

Stealing Password Data

on $(\mathsf{StealPWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ from $\mathcal{F}_{\mathsf{apake}}$:

Mark $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$ as $\mathsf{compromised}$ and return it to $\mathcal{A}$.
If $\mathcal{F}_{\mathsf{apake}}$ additionally returns $\hat{\mathsf{pw}}$, pass $\hat{\mathsf{pw}}$ to $\mathcal{A}$.

aPAKE Sessions

on $(\mathsf{NewClient}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, b)$ from $\mathcal{F}_{\mathsf{apake}}$:

$e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_{\mathsf{IC}_1}$, $\mathcal{T}_{\mathsf{IC}_1} := \mathcal{T}_{\mathsf{IC}_1} \cup \{e_1\}$, $\mathcal{T} := \mathcal{T} \cup \{(\mathsf{C}^{(i)}, iid^{(i)}, e_1)\}$, 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 $\mathcal{F}_{\mathsf{apake}}$ and e_1 from $\mathcal{A}$ as a client message to $(\mathsf{S}^{(j)}, iid^{(j)})$:

$e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_{\mathsf{IC}_2}$, $\mathcal{T}_{\mathsf{IC}_2} := \mathcal{T}_{\mathsf{IC}_2} \cup \{e_2\}$, $\mathcal{T} := \mathcal{T} \cup \{(\mathsf{S}^{(j)}, iid^{(j)}, e_1, e_2)\}$, send e_2 from $\mathsf{S}^{(j)}$ to $\mathcal{A}$.

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 either 1) $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$ is $\mathsf{compromised}$ and $(\mathsf{C}^{(i)}, iid^{(i)})$ is $\mathsf{correct\text{-}pw}$; or 2) $\exists(\mathsf{h}', Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$ and $\exists(\mathsf{C}^{(i)}, \mathsf{S}^{(i)}, \mathsf{pw}', \mathsf{h}', v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$, and $\mathcal{F}_{\mathsf{apake}}$ returns "correct guess" upon query $(\mathsf{TestPW}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{pw}')$:

Let $X_1 || X_2 \leftarrow \mathsf{D}_1(\mathsf{pw}', e_1)$ and $Y \leftarrow \mathsf{D}_2(\mathsf{pw}', e_2)$, retrieve items $(X_1 || X_2, x_1 || x_2) \in \mathcal{DL}$ and $(V_1 || V_2, v_1 || v_2) \in \mathcal{DL}$, $Z_1 := Y^{x_1}$, $Z_2 := Y^{x_2}$, $Z_3 := Y^{v_1}$, $Z_4 := Y^{v_2}$, $(\mathsf{key}, \sigma) \leftarrow \mathsf{H}(sid, Z_1, Z_2, Z_3, Z_4, \mathsf{h}')$, send σ from $\mathsf{C}^{(i)}$ to $\mathcal{A}$, $\mathcal{T} := \mathcal{T} \cup \{(\mathsf{C}^{(i)}, iid^{(i)}, e_1, e_2, \sigma)\}$, and send $(\mathsf{CorruptKey}, \mathsf{C}^{(i)}, iid^{(j)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{apake}}$.

In other cases: $\sigma \xleftarrow{\$} \{0,1\}^{\lambda}$, send σ from $\mathsf{C}^{(i)}$ to $\mathcal{A}$, $\mathcal{T} := \mathcal{T} \cup \{(\mathsf{C}^{(i)}, iid^{(i)}, e_1, e_2, \sigma)\}$, and send $(\mathsf{FreshKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$ to $\mathcal{F}_{\mathsf{apake}}$.

on σ from $\mathcal{A}$ as a client message from $\mathsf{C}^{(i)}$ to $(\mathsf{S}^{(j)}, iid^{(j)})$:

Retrieve $(\mathsf{S}^{(j)}, iid^{(j)}, e_1, e_2) \in \mathcal{T}$, $sid := \mathsf{C}^{(i)} || \mathsf{S}^{(j)} || e_1 || e_2$.
If $\exists(\mathsf{C}^{(i)}, iid^{(i)}, e_1, \cdot, \sigma' = \sigma) \in \mathcal{T}$, $(\mathsf{C}^{(i)}, iid^{(i)})$ is $\mathsf{correct\text{-}pw}$, and Sim has queried $(\mathsf{FreshKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$:

Send $(\mathsf{CopyKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{apake}}$.

If $\exists(\mathsf{h}', X_1 || X_2, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$ and $\exists(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}', \mathsf{h}', v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$: ask $(\mathsf{TestPW}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{pw}')$, and if $\mathcal{F}_{\mathsf{apake}}$ returns "correct guess":

$Y \leftarrow \mathsf{D}_2(\mathsf{h}', e_2)$, retrieve item $(Y, y) \in \mathcal{DL}$, $Z_1 := X_1^y$, $Z_2 := X_2^y$, $Z_3 := g^{v_1 y}$, $Z_4 := g^{v_2 y}$, $(\mathsf{key}, \sigma') \leftarrow \mathsf{H}(sid, Z_1, Z_2, Z_3, Z_4, \mathsf{h}')$: if $\sigma' = \sigma$: send $(\mathsf{CorruptKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{apake}}$.

In other cases: send $(\mathsf{Abort}, \mathsf{S}^{(j)}, iid^{(j)})$ to $\mathcal{F}_{\mathsf{apake}}$.

Fig. 7. Simulator Sim for 2DH-aEKE in the proof of Theorem 3, part 1.

```
$\mathsf{Sim}$ maintains lists $\mathcal{L}_{\mathsf{IC}_1}, \mathcal{L}_{\mathsf{IC}_2}, \mathcal{T}_{\mathsf{IC}_1}, \mathcal{T}_{\mathsf{IC}_2}, \mathcal{L}_{\mathsf{H}_0}, \mathcal{T}_{\mathsf{H}_0}, \mathcal{L}_{\mathsf{H}}, \mathcal{T}, \mathcal{DL}$ (all initialized to be empty) in the simulation

On Ideal Ciphers and Random Oracles
on $\mathsf{E}_1(\mathsf{h}, X_1||X_2)$ from $\mathcal{A}$:
    If $\exists(\mathsf{h}, X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$: return $e_1$.
    Otherwise: $e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_{\mathsf{IC}_1}$, $\mathcal{L}_{\mathsf{IC}_1} := \mathcal{L}_{\mathsf{IC}_1} \cup (\mathsf{h}, X_1||X_2, e_1, enc)$, $\mathcal{T}_{\mathsf{IC}_1} := \mathcal{T}_{\mathsf{IC}_1} \cup \{e_1\}$, return $e_1$.
on $\mathsf{D}_1(\mathsf{h}, e_1)$ from $\mathcal{A}$:
    If $\exists(\mathsf{h}, X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$: return $X_1||X_2$.
    Otherwise: $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$, $X_1 := g^{x_1}$, $X_2 := g^{x_2}$, $\mathcal{L}_{\mathsf{IC}_1} := \mathcal{L}_{\mathsf{IC}_1} \cup \{(\mathsf{h}, X_1||X_2, e_1, dec)\}$, $\mathcal{DL} := \mathcal{DL} \cup \{(X_1||X_2, x_1||x_2)\}$, return $X_1||X_2$.
on $\mathsf{E}_2(\mathsf{h}, Y)$ from $\mathcal{A}$:
    If $\exists(\mathsf{h}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$: return $e_2$.
    Otherwise: $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_{\mathsf{IC}_2}$, $\mathcal{L}_{\mathsf{IC}_2} := \mathcal{L}_{\mathsf{IC}_2} \cup \{(\mathsf{h}, Y, e_2, enc)\}$, $\mathcal{T}_{\mathsf{IC}_2} := \mathcal{T}_{\mathsf{IC}_2} \cup \{e_2\}$, return $e_2$.
on $\mathsf{D}_2(\mathsf{h}, e_2)$ from $\mathcal{A}$:
    If $\exists(\mathsf{h}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$: return $Y$.
    Otherwise: $y \xleftarrow{\$} \mathbb{Z}_q$, $Y := g^y$, $\mathcal{L}_{\mathsf{IC}_2} := \mathcal{L}_{\mathsf{IC}_2} \cup \{(\mathsf{h}, Y, e_2, dec)\}$, $\mathcal{DL} := \mathcal{DL} \cup \{(Y, y)\}$, return $Y$.
on $\mathsf{H}_0(\mathsf{C}, \mathsf{S}, \mathsf{pw})$ from $\mathcal{A}$:
    If $\exists(\mathsf{C}, \mathsf{S}, \mathsf{pw}, \mathsf{h}, v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$: return $(\mathsf{h}, v_1, v_2)$.
    Otherwise, send $(\mathsf{OfflineTestPW}, \mathsf{C}, \mathsf{S}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{apake}}$:
        If $\mathcal{F}_{\mathsf{apake}}$ returns "correct guess": retrieve $\mathsf{S}.\mathsf{file}[\mathsf{C}] = (\mathsf{h}, V_1, V_2)$ and $(V_1||V_2, v_1||v_2) \in \mathcal{DL}$.
        Else: $\mathsf{h} \xleftarrow{\$} \mathcal{H} \backslash \mathcal{T}_{\mathsf{H}_0}$, $\mathcal{T}_{\mathsf{H}_0} := \mathcal{T}_{\mathsf{H}_0} \cup \{\mathsf{h}\}$, $v_1, v_2 \xleftarrow{\$} \mathbb{Z}_q$, $V_1 := g^{v_1}$, $V_2 := g^{v_2}$, $\mathcal{DL} := \mathcal{DL} \cup \{(V_1||V_2, v_1||v_2)\}$.
    Record $(\mathsf{C}, \mathsf{S}, \mathsf{pw}, \mathsf{h}, v_1, v_2)$ in $\mathcal{L}_{\mathsf{H}_0}$, and return $(\mathsf{h}, v_1, v_2)$.
on $\mathsf{H}(\mathsf{C}, \mathsf{S}, e_1, e_2, Z_1, Z_2, Z_3, Z_4, \mathsf{h})$ from $\mathcal{A}$:
    $sid := \mathsf{C}||\mathsf{S}||e_1||e_2$.
    If $\exists(sid, Z_1, Z_2, Z_3, Z_4, \mathsf{h}, \mathsf{key}, \sigma) \in \mathcal{L}_{\mathsf{H}}$ for some $\mathsf{key}$ and $\sigma$: return $(\mathsf{key}, \sigma)$.
    Otherwise: $\mathsf{key} \xleftarrow{\$} \mathcal{K}$, $\sigma \xleftarrow{\$} \{0,1\}^\lambda$, record $(sid, Z_1, Z_2, Z_3, Z_4, \mathsf{h}, \mathsf{key}, \sigma)$ in $\mathcal{L}_{\mathsf{H}}$, and return $(\mathsf{key}, \sigma)$.
```

Fig. 8. Simulator Sim for 2DH-aEKE in the proof of Theorem 3, part 2.

- On $\mathsf{E}_1(\mathsf{h}, X_1||X_2)$: If there exists $(\mathsf{h}, X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$, return e_1. Otherwise, $e_1 \xleftarrow{\$} \mathcal{E}_1 \backslash \mathcal{T}_{\mathsf{IC}_1}$, add $(\mathsf{h}, X_1||X_2, e_1, enc)$ 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 $\mathsf{D}_1(\mathsf{h}, e_1)$: If there exists $(\mathsf{h}, X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$, 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 $(\mathsf{h}, X_1||X_2, e_1, dec)$ 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 $\mathsf{E}_2(\mathsf{h}, Y)$: If there exists $(\mathsf{h}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$, return e_2. Otherwise, $e_2 \xleftarrow{\$} \mathcal{E}_2 \backslash \mathcal{T}_{\mathsf{IC}_2}$, add $(\mathsf{h}, Y, e_2, enc)$ in $\mathcal{L}_{\mathsf{IC}_2}$, add e_2 in $\mathcal{T}_{\mathsf{IC}_2}$, and return e_2.
- On $\mathsf{D}_2(\mathsf{h}, e_2)$: If there exists $(\mathsf{h}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$, return Y. Otherwise, $y \xleftarrow{\$} \mathbb{Z}_q$, $Y := g^y$, add $(\mathsf{h}, Y, e_2, dec)$ in $\mathcal{L}_{\mathsf{IC}_2}$, add (Y, y) in $\mathcal{DL}$, and return Y.

- On $\mathsf{H}_0(\mathsf{C}, \mathsf{S}, \mathsf{pw})$: If there exists $(\mathsf{C}, \mathsf{S}, \mathsf{pw}, \mathsf{h}, v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$, return (h, v_1, v_2). Otherwise, $\mathsf{h} \xleftarrow{\$} \mathcal{H} \backslash \mathcal{T}_{\mathsf{H}_0}$, add h in $\mathcal{T}_{\mathsf{H}_0}$, $v_1, v_2 \xleftarrow{\$} \mathbb{Z}_q$, $V_1 := g^{v_1}$, $V_2 := g^{v_2}$, add $(\mathsf{C}, \mathsf{S}, \mathsf{pw}, \mathsf{h}, v_1, v_2)$ in $\mathcal{L}_{\mathsf{H}_0}$, add $(V_1||V_2, v_1||v_2)$ in $\mathcal{DL}$, and return (h, v_1, v_2).
- On $\mathsf{H}(\mathsf{C}, \mathsf{S}, e_1, e_2, Z_1, Z_2, Z_3, Z_4, \mathsf{h})$: Let $sid := \mathsf{C}||\mathsf{S}||e_1||e_2$. If there exists $(sid, Z_1, Z_2, Z_3, Z_4, \mathsf{h}, \mathsf{key}, \sigma) \in \mathcal{L}_{\mathsf{H}}$, return (key, σ). Otherwise, $\mathsf{key} \xleftarrow{\$} \mathcal{K}$, $\sigma \xleftarrow{\$} \{0,1\}^{\lambda}$, add $(sid, Z_1, Z_2, Z_3, Z_4, \mathsf{h}, \mathsf{key}, \sigma)$ in $\mathcal{L}_{\mathsf{H}}$ and return (key, σ).
- Simulation of clients' registration, password file storage, and password compromise. When a client $\mathsf{C}^{(i)}$ makes a registration to a server $\mathsf{S}^{(j)}$ with password $\hat{\mathsf{pw}}$, Sim invokes $\mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}})$ to obtain $(\hat{\mathsf{h}}, \hat{v}_1, \hat{v}_2)$, and computes $\hat{V}_1 := g^{\hat{v}_1}, \hat{V}_2 := g^{\hat{v}_2}$. Then it sets the password file as $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] := (\hat{\mathsf{h}}, \hat{V}_1, \hat{V}_2)$. Whenever $\mathcal{A}$ compromises the password file, Sim returns $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$.

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

$$|\Pr[\textbf{Game 1} \Rightarrow 1] - \Pr[\textbf{Game 0} \Rightarrow 1]| \le \frac{Q_{ic}^2}{|\mathcal{E}_1|} + \frac{Q_{ic}^2}{|\mathcal{E}_2|} + \frac{Q_{\mathsf{H}_0}^2}{|\mathcal{H}|},$$

where Q_{ic} and Q_{H_0} denote the maximum numbers of IC and H_0 queries, respectively.

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 good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ is linked to a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$, then Sim generates a random session key for $(\mathsf{C}^{(i)}, iid^{(i)})$, and samples a random σ as the third message.

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

Similar to that in the proof of Theorem 2, we show that **Game 1** and **Game 2** are indisintugishable due to the strong 2DH assumption.

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 $\mathsf{H}(sid, \hat{Z}_1, \hat{Z}_2, \hat{Z}_3, \hat{Z}_4, \hat{\mathsf{h}})$ such that

$$(\hat{Z}_1, \hat{Z}_2) = \mathsf{2DH}(\mathsf{D}_1(\hat{\mathsf{h}}, e_1), \mathsf{D}_2(\hat{\mathsf{h}}, e_2)) \ \wedge \ (\hat{Z}_3, \hat{Z}_4) = \mathsf{2DH}(g^{\hat{v}_1}, g^{\hat{v}_2}, \mathsf{D}_2(\hat{\mathsf{h}}, e_2)),$$

where $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||e_1||e_2$ with e_1 and e_2 the transcripts, $(\hat{\mathsf{h}}, \hat{v}_1, \hat{v}_2) \leftarrow \mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}})$ with $\hat{\mathsf{pw}}$ 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.

The reduction works as follows. $\mathcal{B}_1$ receives the 2DH challenge $(\bar{X}_1, \bar{X}_2, \bar{Y}) = (g^{\bar{x}_1}, g^{\bar{x}_2}, g^{\bar{y}})$, as well as an oracle 2DH which inputs (Y, Z_1, Z_2) and outputs whether $(Z_1, Z_2) = \mathsf{2DH}(\bar{X}_1, \bar{X}_2, Y)$. Let $\mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}}) = (\hat{\mathsf{h}}, \hat{v}_1, \hat{v}_2)$ and $\hat{\mathsf{pw}}$ is the correct password pre-shared between $\mathsf{C}^{(i)}$ and $\mathsf{S}^{(j)}$. Then $\mathcal{B}_1$ simulates **Game 2** as below.

- The simulation of $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2), \mathsf{H}_0, \mathsf{H}$ is the same as that in **Game 2**.
- The simulation of clients' registrations, password storage, responses to $\mathcal{A}$'s offline attacks and compromise is the same as that in **Game 2**.
- For the simulation of good client instance $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ generating the first message e_1: $\mathcal{B}_1$ samples $a_{s,1}^{(i)}, a_{s,2}^{(i)} \xleftarrow{\$} \mathbb{Z}_q$, and sets $e_1 \leftarrow \mathsf{E}_1(\hat{\mathsf{h}}, X_1 || X_2)$, where $X_1 := \bar{X}_1 g^{a_{s,1}^{(i)}} = g^{\bar{x}_1 + a_{s,1}^{(i)}}$ and $X_2 := \bar{X}_2 g^{a_{s,2}^{(i)}} = g^{\bar{x}_2 + a_{s,2}^{(i)}}$.
- For the simulation of server instance $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ generating the second message e_2: Let e_1 be the received message. There are two cases.
  - If $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ is partially linked to some good instance $(\mathsf{C}^{(i)}, iid^{(i)} = s)$, it samples $b_t^{(j)} \leftarrow \mathbb{Z}_q$, $Y := g^{\bar{y} + b_t^{(j)}}$.
  - Otherwise, either $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ is partially linked to some bad client instance, or e_1 is adversarially generated. In this case, $\mathcal{B}_1$ samples $y \leftarrow \mathbb{Z}_q$, $Y := g^y$.

  In either case, it outputs $e_2 \leftarrow \mathsf{E}_2(\hat{\mathsf{h}}, Y)$.
- For the simulation of good client instance $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ generating the session key and the third message σ, let $\mathsf{S}^{(j)}$ be its intended partner and $e_1 || e_2$ be the transcripts. There are two cases.
  - If $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ and server instance $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ are linked to each other, $\mathcal{B}_1$ assigns a random key for $(\mathsf{C}^{(i)}, iid^{(i)} = s)$, and samples a random σ as the third message.
  - If $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ is not linked to any server instance, then $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ must have received an adversarially generated message e_2. Let $sid := \mathsf{C}^{(i)} || \mathsf{S}^{(j)} || e_1 || e_2$. $\mathcal{B}_1$ retrieves $(\hat{\mathsf{h}}, X_1 || X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$, $(\hat{\mathsf{h}}, Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$ ($\mathcal{B}_1$ generates the items if they do not exist) and $(\hat{V}_1 || \hat{V}_2, \hat{v}_1, \hat{v}_2) \in \mathcal{DL}$, and checks whether there exists $(sid, Z_1, Z_2, Z_3 = Y^{\hat{v}_1}, Z_4 = Y^{\hat{v}_2}, \mathsf{h}, \mathsf{key}, \sigma) \in \mathcal{L}_{\mathsf{H}}$, such that $\mathsf{2DH}(X_1, X_2, Y) = (Z_1, Z_2)$. If yes, $\mathcal{B}_1$ assigns key as the session key of $(\mathsf{C}^{(i)}, iid^{(i)} = s)$, and sends σ out. Otherwise, $\mathcal{B}_1$ randomly samples (key, σ), and "views" it as the hash output for the correct input $\mathsf{H}(sid, \mathsf{2DH}(X_1,\ X_2, Y) = (?, ?), Z_3 = Y^{\hat{v}_1}, Z_4 = Y^{\hat{v}_2}, \hat{\mathsf{h}})$, where $\mathsf{2DH}(X_1, X_2, Y) = (?, ?)$ means that the values of $\mathsf{2DH}(X_1, X_2, Y)$ are to be determined. If $\mathcal{A}$ later asks $\mathsf{H}(sid, Z_1, Z_2, Z_3 = Y^{\hat{v}_1}, Z_4 = Y^{\hat{v}_2}, \hat{\mathsf{h}})$, then $\mathcal{B}_1$ checks whether $Z_1 || Z_2$ are the correct 2DH values via the decisional oracle 2DH, i.e., it checks whether

$$2\mathrm{DH}(Y, Z_1 / Y^{a_{s,1}^{(i)}}, Z_2 / Y^{a_{s,2}^{(i)}}) = 1.$$

    If yes, $\mathcal{B}_1$ reprograms the random oracle s.t. $\mathsf{H}(sid, Z_1, Z_2, Z_3, Z_4, \hat{\mathsf{h}}) = (\mathsf{key}, \sigma)$ by replacing $(?, ?)$ with (Z_1, Z_2). In this way, the view of $\mathcal{A}$ is consistent.

- For the simulation of server instance $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ generating the session key, there are two cases.
  - If $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ and good client instance $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ are linked to each other, then $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ must has been assigned with a random key key. In this case Sim copies key as the session key of $(\mathsf{S}^{(j)}, iid^{(j)} = t)$.
  - Otherwise, Sim computes the session key according to the protocol specification.
- The simulation of bad client instances is the same as that in **Game 2**.

Suppose that bad_1 happens w.r.t. instances $(\mathsf{C}^{(i)}, s)$ and $(\mathsf{S}^{(j)}, t)$ with transcripts $e_1||e_2$, then $\mathcal{A}$ must have asked $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \hat{Z}_1, \hat{Z}_2, \hat{Z}_3, \hat{Z}_4, \hat{\mathsf{h}})$ s.t.

$$(\hat{Z}_1, \hat{Z}_2) = \mathsf{2DH}(g^{\bar{x}_1 + a_{s,1}^{(i)}}, g^{\bar{x}_2 + a_{s,2}^{(i)}}, g^{\bar{y} + b_t^{(j)}}) \wedge (\hat{Z}_3, \hat{Z}_4) = ((g^{\bar{y} + b_t^{(j)}})^{\hat{v}_1}, (g^{\bar{y} + b_t^{(j)}})^{\hat{v}_2}).$$

Note that $\mathcal{B}_1$ can detect bad_1 with the help of oracle 2DH and trapdoors $a_{s,1}^{(i)}, a_{s,2}^{(i)}, b_t^{(j)}$. Then $\mathcal{B}_1$ makes use of $\hat{Z}_1||\hat{Z}_2$ to extract

$$\hat{Z}_1 / g^{\bar{x}_1 b_t^{(j)} + \bar{y} a_{s,1}^{(i)} + a_{s,1}^{(i)} b_t^{(j)}} = g^{\bar{x}_1 \bar{y}} \text{ and } \hat{Z}_2 / g^{\bar{x}_2 b_t^{(j)} + \bar{y} a_{s,2}^{(i)} + a_{s,2}^{(i)} b_t^{(j)}} = g^{\bar{x}_2 \bar{y}},$$

which is obviously the solution to the strong 2DH problem.

Therefore, we have

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

Game 3. (Randomize simulated messages.) In this game, Sim directly samples random values to simulate the hash output $(\hat{\mathsf{h}}, \hat{v}_1, \hat{v}_2)$ in password file storage, and samples random messages to simulate the transcripts e_1 and e_2. Meanwhile, it postpones the usage of random oracle H_0 and ideal ciphers $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$ until necessary (for example, until $\mathcal{A}$ issues queries to H_0, or D_1 and D_2 are invoked for the generation of session keys). More precisely, **Game 3** is now simulated by Sim as follows.

- For the simulation of password file storage w.r.t. a client $\mathsf{C}^{(i)}$ and a server $\mathsf{S}^{(j)}$, let $\hat{\mathsf{pw}}$ be the correct password. Sim samples $\hat{\mathsf{h}} \xleftarrow{\$} \mathcal{H} \backslash \mathcal{T}_{\mathsf{H}_0}, \hat{v}_1, \hat{v}_2 \xleftarrow{\$} \mathbb{Z}_q$, computes $\hat{V}_1 := g^{\hat{v}_1}, \hat{V}_2 := g^{\hat{v}_2}$, adds $\hat{\mathsf{h}}$ in $\mathcal{T}_{\mathsf{H}_0}$, adds $(\hat{V}_1 || \hat{V}_2, \hat{v}_1 || \hat{v}_2)$ in $\mathcal{DL}$, and records $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] := (\hat{\mathsf{h}}, \hat{V}_1, \hat{V}_2)$. Later whenever there is a hash query on $\mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}})$, Sim patches the list $\mathcal{L}_{\mathsf{H}_0}$ by adding $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}}, \hat{\mathsf{h}}, \hat{v}_1, \hat{v}_2)$ in it, and returns $(\hat{\mathsf{h}}, \hat{v}_1, \hat{v}_2)$.
- 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, 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 $\mathcal{T}_{\mathsf{IC}_2}$.

- For the simulation of a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ that sends e_1 out and receives e_2, there are three subcases.
  - If good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ is linked to some server instance $(\mathsf{S}^{(j)}, iid^{(j)})$, then Sim assigns a random key for $(\mathsf{C}^{(i)}, iid^{(i)})$, and samples a random σ as the third message, just like **Game 2**.
  - If $(\mathsf{C}^{(i)}, iid^{(i)})$ is good and e_2 is adversarially generated, then Sim retrieves $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] = (\mathsf{h}, V_1, V_2) = (\hat{\mathsf{h}}, \hat{V}_1, \hat{V}_2)$ and $(V_1||V_2, v_1||v_2) \in \mathcal{DL}$, invokes $(X_1, X_2) \leftarrow \mathsf{D}_1(\mathsf{h}, e_1)$, and retrieves $(X_1||X_2, x_1||x_2) \in \mathcal{DL}$ (due to the simulation of D_1, such records always exist).
  - If $(\mathsf{C}^{(i)}, iid^{(i)})$ is bad, i.e., the password pw used in this instance is incorrect, Sim invokes $(\mathsf{h}, v_1, v_2) \leftarrow \mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$, $(X_1, X_2) \leftarrow \mathsf{D}_1(\mathsf{h}, e_1)$, and retrieves $(X_1||X_2, x_1||x_2) \in \mathcal{DL}$.

  Let $Y \leftarrow \mathsf{D}_2(\mathsf{h}, e_2)$. In the last two cases, with the knowledge of x_1, x_2, v_1, v_2, Sim can compute $(\mathsf{key}, \sigma) \leftarrow \mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, Y^{x_1}, Y^{x_2}, Y^{v_1}, Y^{v_2}, \mathsf{h})$, and send σ as the third message, just like **Game 2**.
- For the simulation of a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ generating the session key after receiving σ. Let e_1 and e_2 be the first two messages.
  - If $(\mathsf{S}^{(j)}, iid^{(j)})$ and some good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ are linked to each other, then Sim assigns the same session key of $(\mathsf{C}^{(i)}, iid^{(i)})$ to $(\mathsf{S}^{(j)}, iid^{(j)})$, just like **Game 2**.
  - Otherwise, Sim invokes $Y \leftarrow \mathsf{D}_2(\hat{\mathsf{h}}, e_2)$ and retrieves $(Y, y) \in \mathcal{DL}$. Then it computes $(\mathsf{key}, \sigma') \leftarrow \mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \mathsf{2DH}(\mathsf{D}_1(\hat{\mathsf{h}}, e_1), Y), \hat{V}_1^y, \hat{V}_2^y, \hat{\mathsf{h}})$ with the knowledge of y. Here $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] = (\hat{\mathsf{h}}, \hat{V}_1, \hat{V}_2)$. If $\sigma = \sigma'$, Sim assigns key as the session key for $(\mathsf{S}^{(j)}, iid^{(j)})$; otherwise, Sim assigns $\perp$. In this way, the session key is the same as that in **Game 2**.

Recall that in **Game 2**, the hash output $(\hat{\mathsf{h}}, \hat{v}_1, \hat{v}_2) \leftarrow \mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}})$ is randomly sampled, which performs identically as that in **Game 3**. Meanwhile, the transcripts e_1 and e_2 in **Game 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 client instances if the password guess is incorrect or the server is not compromised.) In this game, the simulator changes the session key generation of client instances as follows.

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. If $(\mathsf{C}^{(i)}, iid^{(i)})$ is good, then Sim retrieves the stored password file $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] = (\mathsf{h}, V_1, V_2)$ and $(V_1||V_2, v_1||v_2) \in \mathcal{DL}$. And if $(\mathsf{C}^{(i)}, iid^{(i)})$ is bad, Sim invokes $\mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$ to get (h, v_1, v_2). Let $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||e_1||e_2$ and $Y \leftarrow \mathsf{D}_2(\mathsf{h}, e_2)$, Sim generates the session key in the following way.

Case (C.1) If $(\mathsf{C}^{(i)}, iid^{(i)})$ is good and linked to some server instance $(\mathsf{S}^{(j)}, iid^{(j)})$, then Sim assigns a random key for $(\mathsf{C}^{(i)}, iid^{(i)})$, and samples a random σ as the third message, 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 three subcases.

Case (C.2.1). If $(\mathsf{C}^{(i)}, iid^{(i)})$ is good and the password file $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$ has been stolen, then Sim computes $(\mathsf{key}, \sigma) \leftarrow \mathsf{H}(sid, \mathsf{2DH}(\mathsf{D}_1(\mathsf{h}, e_1), Y), Y^{v_1}, Y^{v_2}, \mathsf{h})$, and sends σ out, just like that in **Game 3**.

Case (C.2.2). If there exists a record $(\mathsf{h}, Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$, and a record $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}' = \mathsf{pw}, \mathsf{h}, v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$, then Sim computes $(\mathsf{key}, \sigma) \leftarrow \mathsf{H}(sid, \mathsf{2DH}(\mathsf{D}_1(\mathsf{h}, e_1), Y), Y^{v_1}, Y^{v_2}, \mathsf{h})$, and sends σ out as the third message, just like that in **Game 3**. Note that there exist at most one such record in $\mathcal{L}_{\mathsf{IC}_2}$ and one such record in $\mathcal{L}_{\mathsf{H}_0}$, since E_2 and H_0 are simulated in a collision-free way.

Case (C.2.3). In any other case, Sim generates a random key for $(\mathsf{C}^{(i)}, iid^{(i)})$, and samples a random $\sigma \xleftarrow{\$} \{0,1\}^\lambda$.

Note that the differences between **Game 3** and **Game 4** lie only in cases (C.2.3).

Define bad_2 as the event that there exists a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ in Case (C.2.3), and $\mathcal{A}$ ever asks a hash query on $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \hat{Z}_1, \hat{Z}_2, \hat{Z}_3, \hat{Z}_4, \mathsf{h})$ such that

$$(\hat{Z}_1, \hat{Z}_2) = \mathsf{2DH}(\mathsf{D}_1(\mathsf{h}, e_1), \mathsf{D}_2(\mathsf{h}, e_2)) \ \wedge \ (\hat{Z}_3, \hat{Z}_4) = \mathsf{2DH}(V_1, V_2, \mathsf{D}_2(\mathsf{h}, e_2)).$$

Obviously **Game 3** and **Game 4** are the same unless bad_2 happens. Next we show that if bad_2 happens, then we can construct a reduction algorithm $\mathcal{B}_2$ to solve the strong 2DH problem.

$\mathcal{B}_2$ receives the 2DH challenge $(\bar{X}_1, \bar{X}_2, \bar{Y}) = (g^{\bar{x}_1}, g^{\bar{x}_2}, g^{\bar{y}})$, as well as a decisional oracle 2DH. Then it simulates **Game 4** as follows.

- The simulation of password file storage is the same as that in **Game 4**.
- For the simulation of stealing password file, $\mathcal{B}_2$ returns $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$ as that in **Game 4**.
- The simulation of $\mathsf{E}_1, \mathsf{E}_2$, H_0, and H is the same as that in **Game 4**.
- Simulation of $\mathsf{D}_1(\mathsf{h}', e_1)$.
  - If there exists $(\mathsf{h}', X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$, return $X_1||X_2$.
  - Otherwise, $\mathcal{B}_2$ samples $a_1, a_2 \xleftarrow{\$} \mathbb{Z}_q$, $X_1 := \bar{X}_1 g^{a_1} = g^{\bar{x}_1 + a_1}$, $X_2 := \bar{X}_2 g^{a_2} = g^{\bar{x}_2 + a_2}$, adds $(\mathsf{h}', X_1||X_2, e_1, dec)$ in $\mathcal{L}_{\mathsf{IC}_1}$, and returns $X_1||X_2$.
- Simulation of $\mathsf{D}_2(\mathsf{h}', e_2)$.
  - If there exists $(\mathsf{h}', Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$, return Y.
  - If $\mathsf{D}_2(\mathsf{h}', e_2)$ is invoked when a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ generating the session key in case of active attacks (including the case of bad client instances), $\mathcal{B}_2$ samples $y \xleftarrow{\$} \mathbb{Z}_q$, $Y := g^y$, adds $(\mathsf{h}', Y, e_2, dec)$ in $\mathcal{L}_{\mathsf{IC}_2}$, adds (Y, y) in $\mathcal{DL}$ and returns Y.

- In other cases, $\mathcal{B}_2$ samples $b \xleftarrow{\$} \mathbb{Z}_q$, $Y := \bar{Y}g^b = g^{\bar{y}+b}$, adds $(\mathsf{h}', Y, e_2, dec)$ in $\mathcal{L}_{\mathsf{IC}_2}$ and returns Y.

– The simulation of e_1 and e_2 is the same as that in **Game 4**.

– The simulation of key generation for server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ is the same as that in **Game 4**.

– Simulation of σ and key generation for client instance $(\mathsf{C}^{(i)}, iid^{(i)})$.

- If $(\mathsf{C}^{(i)}, iid^{(i)})$ is good and linked to some server instance $(\mathsf{S}^{(j)}, iid^{(j)})$, then $\mathcal{B}_2$ assigns a random key for it, and samples a random σ as the third message, just as that in **Game 4**.
- Otherwise, $\mathcal{B}_2$ computes the session key with the help of the decisional oracle 2DH. Concretely, let e_1 and e_2 be the transcripts, $\mathsf{S}^{(j)}$ be its intended partner, and pw be the (possibly incorrect) password used in $(\mathsf{C}^{(i)}, iid^{(i)})$. For good client instance, $\mathcal{B}_2$ directly retrieves the password file $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] = (\mathsf{h}, V_1, V_2)$, and the record $(V_1||V_2, v_1||v_2) \in \mathcal{DL}$. From the simulation we know, $(V_1||V_2, v_1||v_2) \in \mathcal{DL}$ always exists. And for bad client instance, $\mathcal{B}_2$ queries $(\mathsf{h}, v_1, v_2) \leftarrow \mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$. Let $X_1||X_2 \leftarrow \mathsf{D}_1(\mathsf{h}, e_1)$, $Y \leftarrow \mathsf{D}_2(\mathsf{h}, e_2)$, and $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||e_1||e_2$.

  $\mathcal{B}_2$ checks whether there exists $(sid, Z_1, Z_2, Z_3, Z_4, \mathsf{h}, \mathsf{key}, \sigma) \in \mathcal{L}_{\mathsf{H}}$, such that $(Z_1, Z_2) = \mathsf{2DH}(X_1, X_2, Y)$ and $Z_3 = Y^{v_1}, Z_4 = Y^{v_2}$. If so, $\mathcal{B}_2$ assigns key as the session key of $(\mathsf{C}^{(i)}, iid^{(i)})$, and sends σ out. Otherwise, $\mathcal{B}_2$ randomly samples (key, σ), and "views" it as the hash value for the correct input $\mathsf{H}(sid, \mathsf{2DH}(X_1, X_2, Y) = (?, ?), Y^{v_1}, Y^{v_2}, \mathsf{h})$, where $\mathsf{2DH}(X_1, X_2, Y) = (?, ?)$ means that the values of $\mathsf{2DH}(X_1, X_2, Y)$ are to be determined. If $\mathcal{A}$ later asks $\mathsf{H}(sid, Z_1, Z_2, Y^{v_1}, Y^{v_2}, \mathsf{h})$, then $\mathcal{B}_2$ checks whether $Z_1||Z_2$ are the correct 2DH values via the decisional oracle 2DH, i.e., it checks whether

  $$2\mathrm{DH}(Y, Z_1/Y^{a_1}, Z_2/Y^{a_2}) = 1,$$

  with the knowledge of a_1, a_2. If yes, $\mathcal{B}_2$ reprograms the random oracle s.t. $\mathsf{H}(sid, Z_1, Z_2, Y^{v_1}, Y^{v_2}, \mathsf{h}) = (\mathsf{key}, \sigma)$ by replacing $(?, ?)$ with (Z_1, Z_2). In this way, the view of $\mathcal{A}$ is consistent.

If bad_2 happens in Case (C.2.3) w.r.t. a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ with intended partner $\mathsf{S}^{(j)}$, then the following conditions satisfy:

(a) $(\mathsf{C}^{(i)}, iid^{(i)})$ is bad, or the password fill $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$ has not been stolen yet;

(b) there does not exist a record $(\mathsf{h}, Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$ or there does not exist a record $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}, \mathsf{h}, v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$;

(c) $\mathcal{A}$ asks a hash query on $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \hat{Z}_1, \hat{Z}_2, \hat{Z}_3, \hat{Z}_4, \mathsf{h})$ such that

$$(\hat{Z}_1, \hat{Z}_2) = \mathsf{2DH}(\mathsf{D}_1(\mathsf{h}, e_1), \mathsf{D}_2(\mathsf{h}, e_2)) = \mathsf{2DH}(g^{\bar{x}_1+a_1}, g^{\bar{x}_2+a_2}, g^{\bar{y}+b}),$$

and

$$(\hat{Z}_3, \hat{Z}_4) = \mathsf{2DH}(V_1, V_2, \mathsf{D}_2(\mathsf{h}, e_2)).$$

Conditioned on (a), if $\mathcal{A}$ never asks $\mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$, then h is perfectly hidden from $\mathcal{A}$, since h is chosen at random and no information of h is leaked via transcripts or hash queries. That is, with overwhelming probability, if there exists a record $(\mathsf{h}, Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$, then the record $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}, \mathsf{h}, v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$ must exist.

In the reduction above, $\mathcal{B}_2$ can detect bad_2 with the help of oracle 2DH and trapdoors a_1, a_2, b. Then, it can make use of $\hat{Z}_1 || \hat{Z}_2$ to extract

$$\hat{Z}_1 / g^{\bar{x}_1 b + \bar{y} a_1 + a_1 b} \text{ and } \hat{Z}_2 / g^{\bar{x}_2 b + \bar{y} a_2 + a_2 b},$$

which is the solution to the strong 2DH problem.

Therefore, we have

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

Game 5. (Randomize keys for actively attacked server instances in case of incorrect password guesses.) In this game, the simulator further changes the session key generation of server instances as follows.

For any server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ that receives e_1 and σ, let e_2 be the message sent out, $\mathsf{C}^{(i)}$ be its intended partner and $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] = (\hat{\mathsf{h}}, \hat{V}_1, \hat{V}_2)$ be the stored password file. Sim generates the session key in the following way.

Case (S.1). If $(\mathsf{S}^{(j)}, iid^{(j)})$ and some good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ are linked to each other, then Sim generates the same key of $(\mathsf{C}^{(i)}, iid^{(i)})$ for $(\mathsf{S}^{(j)}, iid^{(j)})$, just as that in **Game 4**.

Case (S.2). Otherwise, we further divide it into the following three subcases.

Case (S.2.1). If there exists a record $(\hat{\mathsf{h}}, X_1 || X_2, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$ and a record $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}}, \hat{\mathsf{h}}, \hat{v}_1, \hat{v}_2) \in \mathcal{L}_{\mathsf{H}_0}$, then Sim invokes $Y \leftarrow \mathsf{D}_2(\hat{\mathsf{h}}, e_2)$, retrieves $(Y, y) \in \mathcal{DL}$, and computes $(\mathsf{key}, \sigma') \leftarrow \mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \mathsf{2DH}(X_1, X_2, Y), \mathsf{2DH}(\hat{V}_1, \hat{V}_2, Y), \hat{\mathsf{h}})$ according to the protocol description, just like that in **Game 4**. If $\sigma = \sigma'$, then $(\mathsf{S}^{(j)}, iid^{(j)})$ outputs key; otherwise, it outputs $\perp$. Note that there exist at most one such record in $\mathcal{L}_{\mathsf{IC}_1}$ and one in $\mathcal{L}_{\mathsf{H}_0}$, since E_1 and H_0 are simulated in a collision-free way.

Case (S.2.2). If there exists a record $(\hat{\mathsf{h}}, X_1 || X_2, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$, but there does not exist a record $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \hat{\mathsf{pw}}, \hat{\mathsf{h}}, \hat{v}_1, \hat{v}_2) \in \mathcal{L}_{\mathsf{H}_0}$, then Sim assigns $\perp$ for $(\mathsf{S}^{(j)}, iid^{(j)})$.

Case (S.2.3). If there does not exist a record $(\hat{\mathsf{h}}, X_1 || X_2, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$, then Sim assigns $\perp$ for $(\mathsf{S}^{(j)}, iid^{(j)})$.

Note that the differences between **Game 4** and **Game 5** lie in Cases (S.2.2) and (S.2.3) only.

We analyse the difference in Case (S.2.3) first. Define bad_3' as the event that there exists a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ in Case (S.2.3), and $\mathcal{A}$ ever asks a hash query on $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, \hat{Z}_1, \hat{Z}_2, \hat{Z}_3, \hat{Z}_4, \hat{\mathsf{h}})$ such that

$$(\hat{Z}_1, \hat{Z}_2) = \mathsf{2DH}(\mathsf{D}_1(\hat{\mathsf{h}}, e_1), \mathsf{D}_2(\hat{\mathsf{h}}, e_2)) \ \wedge \ (\hat{Z}_3, \hat{Z}_4) = \mathsf{2DH}(\hat{V}_1, \hat{V}_2, \mathsf{D}_2(\hat{\mathsf{h}}, e_2)),$$

where e_1 and e_2 are the transcripts, and $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] = (\hat{\mathsf{h}}, \hat{V}_1, \hat{V}_2)$ is the stored password file.

Recall that $\mathsf{H}(\cdot)$ works as a random oracle. If $\mathcal{A}$ has not asked a hash query as above, then with overwhelming probability, $\sigma \neq \sigma'$ and $(\mathsf{S}^{(j)}, iid^{(j)})$ would output $\perp$. Next we show that, if bad_3' happens, then we can construct a reduction algorithm $\mathcal{B}_3'$ to solve the strong 2DH problem. Intuitively, in the reduction the 2DH challenge problem is embedded into $\mathsf{D}_1(\hat{\mathsf{h}}, e_1)$ and $\mathsf{D}_2(\hat{\mathsf{h}}, e_2)$, respectively. Due to the similarity with that in the proof of bad_2 (from **Game 3** to **Game 4**), we omit the details of reduction and draw the following conclusion.

$$\Pr[\mathsf{bad}_3'] \leq \mathsf{Adv}^{\mathsf{st2DH}}_{\mathbb{G},\mathcal{B}_3'}(\lambda) + 2^{-\Omega(\lambda)}.$$

Then we analyse the difference between **Game 4** and **Game 5** in Case (S.2.2). Define bad_3 as the event that there exists a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ in Case (S.2.2), and $\mathcal{A}$ ever asks a hash query on $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, e_1, e_2, Z_1, Z_2, \hat{Z}_3, \hat{Z}_4, \hat{\mathsf{h}})$ such that

$$(\hat{Z}_3, \hat{Z}_4) = \mathsf{2DH}(\hat{V}_1, \hat{V}_2, \mathsf{D}_2(\hat{\mathsf{h}}, e_2)),$$

where e_1 and e_2 are the transcripts, and $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] = (\hat{\mathsf{h}}, \hat{V}_1, \hat{V}_2)$ is the stored password file.

Similarly, since $\mathsf{H}(\cdot)$ is a random oracle, $\mathcal{A}$ can hardly generates a σ that passes the verification of $(\mathsf{S}^{(j)}, iid^{(j)})$, if it never asks the hash query above. Note that the definition of bad_3 omits the correctness check of Z_1, Z_2 in hash queries. Next we show that if bad_3 happens, we can construct a reduction algorithm $\mathcal{B}_3$ to solve the strong 2DH problem. The reduction has a optimal loss factor N, the total number of client-server pairs.

$\mathcal{B}_3$ receives the 2DH challenge $(\bar{X}_1, \bar{X}_2, \bar{Y}) = (g^{\bar{x}_1}, g^{\bar{x}_2}, g^{\bar{y}})$, as well as a decisional oracle 2DH. Then it simulates **Game 5** as follows.

- For the simulation of password file storage, $\mathcal{B}_3$ randomly samples a client-server pair $(\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$ among all N choices, hoping that bad_3 happens in one server instance $(\mathsf{S}^{(*)}, iid)$ with intended partner $\mathsf{C}^{(*)}$. It samples $\hat{\mathsf{h}} \xleftarrow{\$} \mathcal{H} \backslash \mathcal{T}_{\mathsf{H}_0}$, adds $\hat{\mathsf{h}}$ in $\mathcal{T}_{\mathsf{H}_0}$, and sets the password file to be $\mathsf{S}^{(*)}.\mathsf{file}[\mathsf{C}^{(*)}] := (\hat{\mathsf{h}}, \hat{V}_1 := \bar{X}_1, \hat{V}_2 := \bar{X}_2)$. If there is a hash query on $\mathsf{H}_0(\mathsf{C}^{(*)}, \mathsf{S}^{(*)}, \hat{\mathsf{pw}})$ with $\hat{\mathsf{pw}}$ the correct password shared between them, $\mathcal{B}_3$ aborts the simulation.
  The simulation of password file storage for other client-server pairs is the same as that in **Game 5**.
- For the simulation of stealing password file, $\mathcal{B}_3$ returns $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$ as that in **Game 5**.
- The simulation of $\mathsf{E}_1, \mathsf{E}_2$, and H_0 is the same as that in **Game 5**.
- The simulation of H is the same as that in **Game 5**. Specifically, $\mathcal{B}_3$ keeps detecting whether bad_3 happens in one instance between $\mathsf{C}^{(*)}$ and $\mathsf{S}^{(*)}$. Namely, whenever there is a query on $\mathsf{H}(\mathsf{C}^{(*)}, \mathsf{S}^{(*)}, e_1, e_2, Z_1, Z_2, Z_3, Z_4, \hat{\mathsf{h}})$, it checks whether

$$\mathrm{2DH}(\mathsf{D}(\hat{\mathsf{h}}, e_2), Z_3, Z_4) = 1,$$

with the help of decisional oracle 2DH, where $\mathsf{S}^{(*)}.\mathsf{file}[\mathsf{C}^{(*)}] = (\hat{\mathsf{h}}, \hat{V}_1 = \bar{X}_1, \hat{V}_2 = \bar{X}_2)$ as defined above.

– Simulation of $\mathsf{D}_1(\mathsf{h}', e_1)$.
  - If there exists $(\mathsf{h}', X_1||X_2, e_1, \cdot) \in \mathcal{L}_{\mathsf{IC}_1}$, return $X_1||X_2$.
  - Otherwise, $\mathcal{B}_3$ samples $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$, $X_1 := g^{x_1}$, $X_2 := g^{x_2}$, adds $(\mathsf{h}', X_1||X_2, e_1, dec)$ in $\mathcal{L}_{\mathsf{IC}_1}$, adds $(X_1||X_2, x_1||x_2)$ in $\mathcal{DL}$, and returns $X_1||X_2$.

– Simulation of $\mathsf{D}_2(\mathsf{h}', e_2)$.
  - If there exists $(\mathsf{h}', Y, e_2, \cdot) \in \mathcal{L}_{\mathsf{IC}_2}$, return Y.
  - Otherwise, if $\mathsf{D}_2(\mathsf{h}', e_2)$ is invoked in Case (S.2.1) or in Case (S.2.2) s.t. $\mathsf{S}^{(*)}.\mathsf{file}[\mathsf{C}^{(*)}] \neq (\mathsf{h}', \cdot, \cdot)$, then $\mathcal{B}_3$ samples $y \xleftarrow{\$} \mathbb{Z}_q$, $Y := g^y$, adds $(\mathsf{h}' = \hat{\mathsf{h}}, Y, e_2, dec)$ in $\mathcal{L}_{\mathsf{IC}_2}$, adds (Y, y) in $\mathcal{DL}$ and returns Y.

    In other cases, $\mathcal{B}_3$ samples $b \xleftarrow{\$} \mathbb{Z}_q$, $Y := \bar{Y} g^b = g^{\bar{y}+b}$, adds $(\mathsf{h}', Y, e_2, dec)$ in $\mathcal{L}_{\mathsf{IC}_2}$ and returns Y.

– The simulation of transcripts e_1 and e_2 is the same as that in **Game 5**.

– Simulation of σ and key generation for client instance $(\mathsf{C}^{(i)}, iid^{(i)})$.
  - If $(\mathsf{C}^{(i)}, iid^{(i)})$ is good and linked to some server instance $(\mathsf{S}^{(j)}, iid^{(j)})$, then $\mathcal{B}_3$ assigns a random key for it, and sends out a random $\sigma \xleftarrow{\$} \{0,1\}^\lambda$, just as that in **Game 5**.
  - Otherwise, $\mathcal{B}_3$ computes the session key with the help of the decisional oracle 2DH. Concretely, let e_1 and e_2 be the transcripts, $\mathsf{S}^{(j)}$ be the intended partner, and pw be the (possibly incorrect) password used in $(\mathsf{C}^{(i)}, iid^{(i)})$. For good client instance, $\mathcal{B}_3$ directly retrieves the password file $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] = (\mathsf{h}, V_1, V_2)$, and the record $(V_1||V_2, v_1||v_2) \in \mathcal{DL}$ if it exists. From the simulation we know, $(V_1||V_2, v_1||v_2) \in \mathcal{DL}$ always exists except the case $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}) = (\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$. And for bad client instance, $\mathcal{B}_3$ invokes $(\mathsf{h}, v_1, v_2) \leftarrow \mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$. Let $X_1||X_2 \leftarrow \mathsf{D}_1(\mathsf{h}, e_1)$, $Y \leftarrow \mathsf{D}_2(\mathsf{h}, e_2)$, and $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||e_1||e_2$. Meanwhile, $\mathcal{B}_3$ retrieves $(X_1||X_2, x_1||x_2)$ in $\mathcal{DL}$.
    - $*$ $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}) \neq (\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$. Since $\mathcal{B}_3$ knows all exponents x_1, x_2, v_1, v_2, it can compute the session key and σ as normal.
    - $*$ $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}) = (\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$. In this case, $V_1 = \bar{X}_1$ and $V_2 = \bar{X}_2$, and $\mathcal{B}_3$ cannot compute Y^{v_1}, Y^{v_2} as above since it does not know the exponents v_1 and v_2. Nevertheless, with the help of decisional oracle 2DH, $\mathcal{B}_3$ checks whether there exists $(sid, Z_1, Z_2, Z_3, Z_4, \mathsf{h}, \mathsf{key}, \sigma) \in \mathcal{L}_{\mathsf{H}}$, such that $(Z_1, Z_2) = (Y^{x_1}, Y^{x_2})$ and $(Z_1, Z_2) = \mathsf{2DH}(V_1, V_2, Y)$. If so, $\mathcal{B}_3$ assigns key as the session key of $(\mathsf{S}^{(j)}, iid^{(j)})$ and sends σ out. Otherwise, $\mathcal{B}_3$ randomly samples (key, σ), and “views” it as the hash value for the correct input $\mathsf{H}(sid, Y^{x_1}, Y^{x_2}, \mathsf{2DH}(V_1, V_2, Y) = (?, ?), \mathsf{h})$, where $\mathsf{2DH}(V_1, V_2, Y) = (?, ?)$ means that the values of $\mathsf{2DH}(V_1, V_2, Y)$ are to be determined. If $\mathcal{A}$ later asks $\mathsf{H}(sid, Y^{x_1}, Y^{x_2}, Z_3, Z_4, \mathsf{h})$, then $\mathcal{B}_3$ checks whether $Z_3||Z_4$ are the correct 2DH values via the decisional oracle 2DH, i.e., it checks whether

$$\mathrm{2DH}(Y, Z_3, Z_4) = 1.$$

If yes, $\mathcal{B}_3$ reprograms the random oracle s.t. $\mathsf{H}(sid, Y^{x_1}, Y^{x_2}, Z_3, Z_4, \mathsf{h}) = (\mathsf{key}, \sigma)$ by replacing $(?, ?)$ with (Z_3, Z_4). In this way, $\mathcal{B}_3$ can make the same simulation as in the case $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}) \neq (\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$, and the view of $\mathcal{A}$ is consistent.

- Simulation of key generation for server instance $(\mathsf{S}^{(j)}, iid^{(j)})$. Let $\mathsf{C}^{i)}$ be its intended partner and $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] = (\hat{\mathsf{h}}, \cdot, \cdot)$.
  - If it is in Case (S.1) (i.e., $(\mathsf{S}^{(j)}, iid^{(j)})$ and some good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ are linked to each other), then $\mathcal{B}_3$ assigns the same session key of $(\mathsf{C}^{(i)}, iid^{(i)})$ for $(\mathsf{S}^{(j)}, iid^{(j)})$, just as that in **Game 5**.
  - If it is in Case (S.2.3), then $\mathcal{B}_3$ assigns $\perp$ for $(\mathsf{S}^{(j)}, iid^{(j)})$.
  - If it is in Case (S.2.1), or in Case (S.2.2) s.t. $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}) \neq (\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$, then $\mathcal{B}_3$ computes the session key according to the protocol description. Note that $\mathcal{B}_3$ is able to compute the correct hash value with the knowledge of y obtained in the simulation of D_2.
  - If it is in Case (S.2.2) s.t. $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}) = (\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$, then $\mathcal{B}_3$ assigns $\perp$ for $(\mathsf{S}^{(j)}, iid^{(j)})$. Consequently, $\mathcal{B}_3$ keeps detecting whether bad_3 happens in one instance between $\mathsf{C}^{(*)}$ and $\mathsf{S}^{(*)}$ in the simulation of H.

If bad_3 happens in Case (S.2.2) w.r.t. server instance $(\mathsf{S}^{(*)}, iid)$ with intended partner $\mathsf{C}^{(*)}$, then $\mathcal{A}$ asks a hash query on $\mathsf{H}(\mathsf{C}^{(*)}, \mathsf{S}^{(*)}, e_1, e_2, Z_1, Z_2, \hat{Z}_3, \hat{Z}_4, \hat{\mathsf{h}})$ such that

$$(\hat{Z}_3, \hat{Z}_4) = \mathsf{2DH}(V_1, V_2, \mathsf{D}_2(\hat{\mathsf{h}}, e_2)) = \mathsf{2DH}(g^{\bar{x}_1}, g^{\bar{x}_2}, g^{\bar{y}+b}).$$

$\mathcal{B}_3$ can make use of $\hat{Z}_3||\hat{Z}_4$ to extract

$$\hat{Z}_3/g^{\bar{x}_1 b} \text{ and } \hat{Z}_4/g^{\bar{x}_2 b},$$

which is the solution to the strong 2DH problem. Note that $\mathcal{B}_3$ can detect bad_3 efficiently with the help of oracle 2DH and trapdoor b.

In the reduction above, $\mathcal{B}_3$ randomly chooses a client-server pair $(\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$. If bad_3 happens, then with probability $1/N$, it happens on one instance of $\mathsf{S}^{(*)}$ with intended partner $\mathsf{C}^{(*)}$. Therefore, we have

$$\Pr[\mathsf{bad}_3] \leq N \cdot \mathsf{Adv}^{\mathsf{st2DH}}_{\mathbb{G},\mathcal{B}_3}(\lambda) + 2^{-\Omega(\lambda)}.$$

In summary, we have

$$|\Pr[\textbf{Game 4} \Rightarrow 1] - \Pr[\textbf{Game 5} \Rightarrow 1]| \leq (N+1) \cdot \mathsf{Adv}^{\mathsf{st2DH}}_{\mathbb{G},\mathcal{B}_3}(\lambda) + 2^{-\Omega(\lambda)}.$$

Now in **Game 5**, Sim does not use pw any more, except the case of session key generation when the adversary $\mathcal{A}$ correctly guesses the password pw used in a client/server instance, i.e., there exists a record $(\mathsf{h}, X_1||X_2, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$ or $(\mathsf{h}, Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$, and a record $(\mathsf{C}, \mathsf{S}, \mathsf{pw}, \mathsf{h}, v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$. Now we are ready to introduce the complete simulator in Fig. 7 and 8, which helps us stepping to the ideal experiment $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$.

Game 6. (Use $\mathcal{F}_{\mathsf{apake}}$ interfaces.) In the final game we introduce the ideal functionality $\mathcal{F}_{\mathsf{apake}}$. By using interfaces to interact with $\mathcal{F}_{\mathsf{apake}}$, the simulator Sim can perfectly simulates **Game 5** as follows.

- It simulates $(\mathsf{E}_1, \mathsf{D}_1), (\mathsf{E}_2, \mathsf{D}_2)$, and H_0, H as described in **Game 5**.
- When Sim receives $(\mathsf{StorePWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ from $\mathcal{F}_{\mathsf{apake}}$, it randomly samples $\hat{\mathsf{h}} \xleftarrow{\$} \mathcal{H} \backslash \mathcal{T}_{\mathsf{H}_0}$, $\hat{v}_1, \hat{v}_2 \xleftarrow{\$} \mathbb{Z}_q$, computes $\hat{V}_1 := g^{\hat{v}_1}$, $\hat{V}_2 := g^{\hat{v}_2}$, and sets the password file as $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}] := (\hat{\mathsf{h}}, \hat{V}_1, \hat{V}_2)$. Meanwhile, it adds $\hat{\mathsf{h}}$ in $\mathcal{T}_{\mathsf{H}_0}$ and $(\hat{V}_1 || \hat{V}_2, \hat{v}_1 || \hat{v}_2)$ in $\mathcal{DL}$.
- When Sim receives $(\mathsf{StealPWFile}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)})$ from $\mathcal{F}_{\mathsf{apake}}$, it marks $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$ as $\mathsf{compromised}$ and sends it to $\mathcal{A}$. If $\mathcal{F}_{\mathsf{apake}}$ additionally returns $\hat{\mathsf{pw}}$ to Sim, which indicates that $\mathcal{A}$ employs a successful offline test and gets the password from the stolen password file, Sim passes the correct password $\hat{\mathsf{pw}}$ to $\mathcal{A}$.
- For the simulation of $\mathsf{H}_0(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$, if there already exists $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}, \mathsf{h}, v_1, v_2)$ in $\mathcal{L}_{\mathsf{H}_0}$, then Sim returns (h, v_1, v_2). Otherwise, it indicates an “offline test” query to the password file $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$, and Sim sends $(\mathsf{OfflineTestPW}, \mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw})$ to $\mathcal{F}_{\mathsf{apake}}$.
  - If $\mathcal{F}_{\mathsf{apake}}$ returns “correct guess”, i.e., pw is the correct password between $\mathsf{C}^{(i)}$ and $\mathsf{S}^{(j)}$, then Sim retrieves $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$ and $(V_1 || V_2, v_1 || v_2)$ from $\mathcal{DL}$.
  - Otherwise (pw is not the correct password, or the password file has not been compromised yet), Sim randomly samples $\mathsf{h} \xleftarrow{\$} \mathcal{H} \backslash \mathcal{T}_{\mathsf{H}_0}$, $v_1, v_2 \xleftarrow{\$} \mathbb{Z}_q$, computes $V_1 := g^{v_1}$, $V_2 := g^{v_2}$, adds h in $\mathcal{T}_{\mathsf{H}_0}$ and $(V_1 || V_2, v_1 || v_2)$ in $\mathcal{DL}$.

  At last, Sim adds $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}, \mathsf{h}, v_1, v_2)$ in $\mathcal{L}_{\mathsf{H}_0}$ and returns (h, v_1, v_2).
- When Sim receives $(\mathsf{NewClient}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, b)$ from $\mathcal{F}_{\mathsf{apake}}$, it marks this instance as $\mathsf{correct\text{-}pw}$ if $b = 1$, indicating that the client instance is good. 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 $\mathcal{F}_{\mathsf{apake}}$, 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 $\mathcal{T}_{\mathsf{IC}_2}$.
- 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 $(\mathsf{C}^{(i)}, iid^{(i)})$ is marked as $\mathsf{correct\text{-}pw}$ and linked to a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$.
  - If it is the case, Sim allocates a random key to $(\mathsf{C}^{(i)}, iid^{(i)})$ by directly asking a query $(\mathsf{FreshKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{apake}}$. Meanwhile, Sim randomly samples σ as the third message. According to the definition of $\mathsf{FreshKey}$ interface, this performs identically as that in **Game 5**.
  - Otherwise, Sim checks whether: (1) the password file $\mathsf{S}^{(j)}.\mathsf{file}[\mathsf{C}^{(i)}]$ has been compromised and $(\mathsf{C}^{(i)}, iid^{(i)})$ is good; or (2) there exists a record $(\mathsf{h}', Y, e_2, enc) \in \mathcal{L}_{\mathsf{IC}_2}$ and a record $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}', \mathsf{h}', v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$, and $\mathcal{F}_{\mathsf{apake}}$ returns “correct guess” after a query on $(\mathsf{TestPW}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{pw}')$.

    - If either (1) or (2) holds, Sim computes $X_1||X_2 \leftarrow \mathsf{D}_1(\mathsf{h}, e_1)$, $Y \leftarrow \mathsf{D}_2(\mathsf{h}, e_2)$, and $(\mathsf{key}, \sigma) \leftarrow \mathsf{H}(sid, \mathsf{2DH}(X_1, X_2, Y), \mathsf{2DH}(g^{v_1}, g^{v_2}, Y), \mathsf{h})$ (with the knowledge of x_1, x_2, v_1, v_2). Then it allocates sid and key to $(\mathsf{C}^{(i)}, iid^{(i)})$ via a query $(\mathsf{CorruptKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{apake}}$, and sends σ out. According to the definition of $\mathsf{CorruptKey}$ interface, the environment $\mathcal{Z}$ has the same view as that in **Game 5**.
    - Otherwise, 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 $\mathcal{F}_{\mathsf{apake}}$. Meanwhile, Sim randomly samples σ as the third message. According to the definition of $\mathsf{FreshKey}$, this results in the same view to the environment $\mathcal{Z}$ as that in **Game 5**.

– When server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ receives σ, let e_1 and e_2 be the first two messages, and $\mathsf{C}^{(i)}$ 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 $(\mathsf{S}^{(j)}, iid^{(j)})$ and a good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ are linked to each other.

- If it is the case, then $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||e_1||e_2$ and a random session key key must have been assigned to $(\mathsf{C}^{(i)}, iid^{(i)})$. Sim assigns the same sid and key to $(\mathsf{S}^{(j)}, iid^{(j)})$ via a query $(\mathsf{CopyKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{apake}}$. According to the definition of $\mathsf{CopyKey}$, this performs identically as that in **Game 5**.
- Otherwise, Sim checks whether there exists a record $(\mathsf{h}', \cdot, e_1, enc) \in \mathcal{L}_{\mathsf{IC}_1}$ and a record $(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, \mathsf{pw}', \mathsf{h}', v_1, v_2) \in \mathcal{L}_{\mathsf{H}_0}$. If such records exist, Sim issues a TestPW query $(\mathsf{TestPW}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{pw}')$ to ask $\mathcal{F}_{\mathsf{apake}}$ whether $\mathsf{pw}' = \mathsf{pw}$, where pw is the (correct) password used in $(\mathsf{S}^{(j)}, iid^{(j)})$.
    - If the record exists and $\mathcal{F}_{\mathsf{apake}}$ returns "correct guess" (i.e., $\mathsf{pw}' = \mathsf{pw}$), then Sim computes $X_1||X_2 \leftarrow \mathsf{D}_1(\mathsf{h}, e_1)$, $Y \leftarrow \mathsf{D}_2(\mathsf{h}, e_2)$, and $(\mathsf{key}, \sigma') \leftarrow \mathsf{H}(sid, \mathsf{2DH}(X_1, X_2, Y), \mathsf{2DH}(g^{v_1}, g^{v_2}, Y), \mathsf{h})$. If $\sigma = \sigma'$, Sim sends $(\mathsf{CorruptKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{apake}}$; otherwise, it sends $(\mathsf{Abort}, \mathsf{S}^{(j)}, iid^{(j)})$ to $\mathcal{F}_{\mathsf{apake}}$. According to the definitions of $\mathsf{CorruptKey}$ and Abort, the environment $\mathcal{Z}$ has the same view as that in **Game 5**.
    - If the record does not exist, or $\mathcal{F}_{\mathsf{apake}}$ returns "wrong guess" (i.e., $\mathsf{pw}' \neq \mathsf{pw}$), then Sim allocates $\perp$ for $(\mathsf{S}^{(j)}, iid^{(j)})$ by asking a query $(\mathsf{Abort}, \mathsf{S}^{(j)}, iid^{(j)})$ to $\mathcal{F}_{\mathsf{apake}}$. According to the definition of Abort interface, this results in the same view to the environment $\mathcal{Z}$ as that in **Game 5**.

The full description of Sim is shown in Fig. 7 and 8. From the analysis above we know **Game 5** and **Game 6** are conceptually identical. Furthermore, one can easily see that **Game 6** is just the experiment in the ideal world. Therefore, we have

$$\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}} = \mathbf{Game\ 6} = \mathbf{Game\ 5}.$$

Theorem 2 follows immediately from **Game 0** to **Game 6**, and Theorem 1.

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 [27]).

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). 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+ [47], KHAPE-HMQV [27], aEKE-HMQV and OKAPE-HMQV [45], 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.

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.

Proof. Following the strategy by Bader et al. [7] and Cohn-Gordon et al. [19], we construct a meta-reduction algorithm $\mathcal{M}$ that uses $\mathcal{R}$ as a subroutine to solve $\mathcal{P}$. We first describe a hypothetically inefficient (but on the other hand, perfect) adversary/environment $\mathcal{Z}$, and then show how $\mathcal{Z}$ can be efficiently simulated by $\mathcal{M}$.

Throughout the proof, we use $\mathsf{pw}^{(j)}$ to denote the password shared between the j-th client/server pair, and use $\mathsf{file}^{(j)}$ and $\mathsf{si}^{(j)}$ to denote the corresponding password file and secret input, respectively. For $1 \leq j \leq N$, we define $[N] := \{1, 2, ..., N\}$ and $[N \backslash j] := [N] - \{j\}$.

Hypothetical Adversary. The hypothetical adversary/environment $\mathcal{Z}$ works as follows.

1. $\mathcal{Z}$ steals password files for all N client-server pairs (C, S).
2. $\mathcal{Z}$ randomly samples $j^* \xleftarrow{\$} [N]$. For all $i \in [N \backslash j^*]$, it corrupts the client in the i-th pair to get the password $\mathsf{pw}^{(i)}$, and computes the secret input $\mathsf{si}^{(i)} := \mathsf{F}(\mathsf{pw}^{(i)}, \mathsf{file}^{(i)})$.

3. $\mathcal{Z}$ computes the uniquely corresponding secret input $\mathsf{si}^{(j^*)}$ from $\mathsf{file}^{(j^*)}$ via exhaustive search (recall that we consider hypothetically inefficient adversary here).
4. Let $(\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$ be the j^*-th client-server pair. $\mathcal{Z}$ impersonates $\mathsf{C}^{(*)}$ to initialize a protocol execution with $\mathsf{S}^{(*)}$, and obtains the session key key^* from $\mathsf{S}^{(*)}$. Meanwhile, $\mathcal{Z}$ uses $\mathsf{si}^{(j^*)}$ to compute the session key key it self.
5. If $\mathsf{key} = \mathsf{key}^*$, then $\mathcal{Z}$ makes a decision it is in the real world. And if $\mathsf{key} \neq \mathsf{key}^*$, then $\mathcal{Z}$ makes a decision it is in the ideal world.

We analyse the success probability of $\mathcal{Z}$. With the help of $\mathsf{si}^{(j^*)}$, $\mathcal{Z}$ is able to impersonate $\mathsf{C}^{(*)}$ to start a protocol execution with $\mathsf{S}^{(*)}$ and compute the session key key (the last property of Definition 3). Meanwhile, at the time $\mathsf{S}^{(*)}$ outputs the session key, the password file $\mathsf{file}^{(j^*)}$ has been compromised, but $\mathcal{Z}$ has not employed an offline search on the correct password $\mathsf{pw}^{(j^*)}$ (i.e., $\mathcal{Z}$ has not asked a hash query with correct input $\mathsf{pw}^{(j^*)}$). According to the functionality of $\mathcal{F}_{\mathsf{apake}}$, in the ideal world, $\mathsf{S}^{(*)}$ will output $\perp$, or an independent and random key[10], while in the real world, $\mathsf{S}^{(*)}$ will output the real session key. Via comparing key^* obtained from $\mathsf{S}^{(*)}$ and key computed by itself, $\mathcal{Z}$ can distinguish the real world execution from the ideal world execution with advantage $1 - 1/|\mathcal{K}|$. The loss $1/|\mathcal{K}|$ is due to the fact that with probability $1/|\mathcal{K}|$, a randomly sampled key collides to the real session key.

Meta-Reduction. The meta-reduction $\mathcal{M}$ interacts with reduction algorithm $\mathcal{R}$ by simulating the adversary $\mathcal{Z}$ (against $\mathcal{R}$) as follows.

1. $\mathcal{M}$ receives the problem instance $P \in \mathcal{P}$ from its own challenger, it also has access to the oracle $\mathcal{O}$. It passes the problem instance to $\mathcal{R}$, and provides $\mathcal{R}$ with the same oracle by forwarding queries and results honestly.
2. $\mathcal{M}$ steals password files for all N client-server pairs (C, S). Then it makes a snapshot of the current state $st_{\mathcal{R}}$ of $\mathcal{R}$.
3. For $j \in [N]$, $\mathcal{M}$ proceeds as follows.
   (a) It corrupts the client in the i-th pair for all $i \in [N \backslash j]$, to get the password $\mathsf{pw}^{(i)}$.
   (b) It computes the secret input $\mathsf{si}^{(i)} := \mathsf{F}(\mathsf{pw}^{(i)}, \mathsf{file}^{(i)})$. Note that F may depend on some ideal function simulated by $\mathcal{R}$, and $\mathcal{R}$ may abort for certain queries.
   (c) $\mathcal{M}$ resets $\mathcal{R}$ to state $st_{\mathcal{R}}$.
4. $\mathcal{M}$ simulates the adversary $\mathcal{Z}$ by processing the following steps.
   (a) It randomly samples $j^* \xleftarrow{\$} [N]$. For all $i \in [N \backslash j^*]$, it corrupts the client in the i-th pair to get the password $\mathsf{pw}^{(i)}$, and computes the secret input $\mathsf{si}^{(i)} := \mathsf{F}(\mathsf{pw}^{(i)}, \mathsf{file}^{(i)})$.

[10] If the Abort interface is provided by $\mathcal{F}_{\mathsf{apake}}$ (as defined in Fig. 5), then $\mathsf{S}^{(*)}$ may output $\perp$ in the ideal world; and if the Abort interface is not provided [45], then $\mathsf{S}^{(*)}$ will output an independent and random key.

(b) Let $(\mathsf{C}^{(*)}, \mathsf{S}^{(*)})$ be the j^*-th client-server pair. It impersonates $\mathsf{C}^{(*)}$ to initialize a protocol execution with $\mathsf{S}^{(*)}$, and obtains the session key key^* from $\mathsf{S}^{(*)}$.

(c) It uses $\mathsf{si}^{(j^*)}$, the secret input of $\mathsf{C}^{(*)}$ w.r.t. the intended partner $\mathsf{S}^{(*)}$, to compute the session key key, and compares it with key^*. Note that this works only if $\mathcal{M}$ was able to obtain $\mathsf{si}^{(j^*)}$ in Step 3.

(d) If $\mathsf{key} = \mathsf{key}^*$, then $\mathcal{M}$ makes a decision $\mathcal{R}$ is running the real world. And if $\mathsf{key} \neq \mathsf{key}^*$, then $\mathcal{M}$ makes a decision $\mathcal{R}$ is running the ideal world.

5. If $\mathcal{R}$ outputs s throughout the experiment, then $\mathcal{M}$ outputs the same value (to its challenger).

Analysis. $\mathcal{M}$ runs the reduction algorithm $\mathcal{R}$ at most N times. Apart from that, $\mathcal{M}$ only processes several simple operations like reading, writing, and protocol executions. Therefore, we have $t_{\mathcal{M}} \approx N \cdot t_{\mathcal{R}}$.

Then we analyse the success probability of $\mathcal{M}$. First, observe that if $\mathcal{M}$ obtains $\mathsf{si}^{(j^*)}$ in Step 3, then it perfectly simulates the hypothetical adversary $\mathcal{Z}$. Define bad as the event that j^* is the only index for which $\mathcal{R}$ did not abort in Step 3 of the meta-reduction. If bad happens, $\mathcal{M}$ learns all secret inputs except for $\mathsf{si}^{(j^*)}$, in which is the only case the simulation of $\mathcal{Z}$ in Step 4c fails. Since the reduction algorithm $\mathcal{R}$ works for at least one index $j \in [N]$, and j^* is sampled randomly, we have

$$\Pr[\mathsf{bad}] \leq 1/N.$$

Let $\mathsf{win}(\mathcal{R}, \mathcal{Z})$ denotes the event that $\mathcal{R}$ outputs the correct solution s to P when interacting with $\mathcal{Z}$, and $\mathsf{win}(\mathcal{R}, \mathcal{M})$ denotes the corresponding event with $\mathcal{M}$. Since $\mathcal{M}$ simulates $\mathcal{Z}$ perfectly unless bad happens, we have

$$|\Pr[\mathsf{win}(\mathcal{R}, \mathcal{Z})] - \Pr[\mathsf{win}(\mathcal{R}, \mathcal{M})]| \leq \Pr[\mathsf{bad}].$$

By definition, $\epsilon_{\mathcal{R}} = \Pr[\mathsf{win}(\mathcal{R}, \mathcal{Z})]$, and $\epsilon_{\mathcal{M}} = \Pr[\mathsf{win}(\mathcal{R}, \mathcal{M})]$ ($\mathcal{M}$ successes if and only if $\mathcal{R}$ successes). As a result, we have

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

which finishes the proof.

Further discussions. If we view the password file as a “public key” and the corresponding secret input as a “secret key”, such a bottleneck to tight reduction is similar to that in signature schemes with multi-user existential unforgeability agains adaptive corruptions (EUF-CMA$^{\mathsf{corr}}$ security) [6].

In the EUF-CMA$^{\mathsf{corr}}$ security of signatures.

- After seeing the verification key $vk^{(i)}$ (among all $\{vk^{(i)}\}$), the adversary $\mathcal{A}_{\mathsf{SIG}}$ can obtain the secret key $sk^{(i)}$ via adaptive corruptions.
- With $sk^{(i)}$, $\mathcal{A}_{\mathsf{SIG}}$ can trivially generates valid signatures for new messages w.r.t. $vk^{(i)}$.
- $\mathcal{A}_{\mathsf{SIG}}$ successes if it forges a valid signature for some new message under an uncorrupted verification key.

In the security of aPAKE.

- After stealing the password file $\mathsf{file}^{(i)}$, the adversary/environment $\mathcal{Z}$ can obtain the secret input $\mathsf{si}^{(i)}$ via successfully employing an offline search on the correct password and then computing F.
- With $\mathsf{si}^{(i)}$, the secret input w.r.t. client C and server S, $\mathcal{Z}$ can trivially impersonate C to S, and compute the session key.
- $\mathcal{Z}$ successes in distinguishing the real world from the ideal world, if it can impersonate a client to a server, after stealing the password file but before correctly searching the password.

The thorny problem to prove the tight EUF-CMA$^{\mathsf{corr}}$ security of signatures, is that in the reduction, the reduction algorithm does not know whether and when $\mathcal{A}_{\mathsf{SIG}}$ corrupts $vk^{(i)}$ and gets $sk^{(i)}$. Similarly, in the proof towards tight security of aPAKE, the reduction algorithm does not know whether and when $\mathcal{Z}$ successfully employs an offline search to get $\mathsf{si}^{(i)}$, after stealing the password file $\mathsf{file}^{(i)}$.

To achieve full-tight security of SIG, all existing schemes [6, 29, 25, 21] are in the form of non-unique secret keys (i.e., for each vk there exists multiple matching secret keys sk). Gjøsteen and Jager [25] pointed out that, these non-unique secret keys are inherently necessary to achieve fully-tight security. However, aPAKE works in a “unique secret key” pattern. Namely, for each password file there exists only one password (hence only one secret input) that can make the client match with the server. Otherwise a different password would also pass the authentication of server, which conflicts to the intention of aPAKE. As a result, the loss factor $L = N$ is inevitable in aPAKE.

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 [36], SPAKE2 [4], and TBPEKE [42], are secure under this relaxed model. We postpone the definition of lazy-extraction UC PAKE functionality $\mathcal{F}_{\mathsf{le\text{-}pake}}$ in Appendix B. Informally, $\mathcal{F}_{\mathsf{le\text{-}pake}}$ 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 out technique can be extended to get tightly secure and ideal cipher-free protocols in the relaxed UC framework. We take 2DH-SPAKE2 (Fig. 9) 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.

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. 9 securely emulates* $\mathcal{F}_{\mathsf{le\text{-}pake}}$. *More precisely, for any PPT environment* $\mathcal{Z}$ *and real world*

$$\begin{array}{ccc}
\multicolumn{3}{c}{\text{Public Parameter: } (\mathbb{G}, g, q), M_1, M_2, N_1, N_2, \mathsf{H}} \\
\text{Client } \mathsf{C}\ (\mathsf{pw}) & & \text{Server } \mathsf{S}\ (\mathsf{pw}) \\
x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q, X_1 := g^{x_1}, X_2 := g^{x_2} & \xrightarrow{X_1^*, X_2^*} & y_1, y_2 \xleftarrow{\$} \mathbb{Z}_q, Y_1 := g^{y_1}, Y_2 := g^{y_2} \\
X_1^* := X_1 \cdot M_1^{\mathsf{pw}}, X_2^* := X_2 \cdot M_2^{\mathsf{pw}} & \xleftarrow{Y_1^*, Y_2^*} & Y_1^* := Y_1 \cdot N_1^{\mathsf{pw}}, Y_2^* := Y_2 \cdot N_2^{\mathsf{pw}} \\
Z_{1,1} := (\frac{Y_1^*}{N_1^{\mathsf{pw}}})^{x_1}, Z_{1,2} := (\frac{Y_2^*}{N_2^{\mathsf{pw}}})^{x_1} & & Z_{1,1} := (\frac{X_1^*}{M_1^{\mathsf{pw}}})^{y_1}, Z_{1,2} := (\frac{X_1^*}{M_1^{\mathsf{pw}}})^{y_2} \\
Z_{2,1} := (\frac{Y_1^*}{N_1^{\mathsf{pw}}})^{x_2}, Z_{2,2} := (\frac{Y_2^*}{N_2^{\mathsf{pw}}})^{x_2} & & Z_{2,1} := (\frac{X_2^*}{M_2^{\mathsf{pw}}})^{y_1}, Z_{2,2} := (\frac{X_2^*}{M_2^{\mathsf{pw}}})^{y_2} \\
sid := \mathsf{C}||\mathsf{S}||X_1^*||X_2^*||Y_1^*||Y_2^* & & sid := \mathsf{C}||\mathsf{S}||X_1^*||X_2^*||Y_1^*||Y_2^* \\
\text{Output } \mathsf{key}_C \leftarrow \mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}) & & \text{Output } \mathsf{key}_S \leftarrow \mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})
\end{array}$$

Fig. 9. The 2DH-SPAKE2 Protocol.

adversary $\mathcal{A}$ which has access to random oracle H*, there exist a PPT simulator* Sim*, which has access to the ideal functionality* $\mathcal{F}_{\mathsf{le\text{-}pake}}$*, 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*

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

The proof is given in Appendix C.

Note that the technique can be further used to extend PAKE protocol PPK [41] (Fig. 17) to 2DH-PPK (Fig. 18 in Appendix D), that achieves tight security in the relaxed UC framework. We omit the detailed proof due to the similarity to the proof of Theorem 5.

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] Bader, C., Hofheinz, D., Jager, T., Kiltz, E., Li, Y.: Tightly-secure authenticated key exchange. In: TCC 2015. vol. 9014, pp. 629–658. Springer (2015)

[7] Bader, C., Jager, T., Li, Y., Schäge, S.: On the impossibility of tight cryptographic reductions. In: Fischlin, M., Coron, J. (eds.) EUROCRYPT 2016. vol. 9666, pp. 273–304. Springer (2016)

[8] 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)

[9] 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)

[10] 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)

[11] 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)

[12] 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)

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

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

[15] 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)

[16] 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)

[17] 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)

[18] 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)

[19] Cohn-Gordon, K., Cremers, C., Gjøsteen, K., Jacobsen, H., Jager, T.: Highly efficient key exchange protocols with optimal tightness. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019. vol. 11694, pp. 767–797. Springer (2019)

[20] Coron, J., Dodis, Y., Mandal, A., Seurin, Y.: A domain extender for the ideal cipher. In: Micciancio, D. (ed.) Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010. vol. 5978, pp. 273–289. Springer (2010)

[21] Diemert, D., Gellert, K., Jager, T., Lyu, L.: More efficient digital signatures with tight multi-user security. In: Public-Key Cryptography - PKC 2021. vol. 12711, pp. 1–31. Springer (2021)

[22] 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)

[23] 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)

[24] 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)

[25] Gjøsteen, K., Jager, T.: Practical and tightly-secure digital signatures and authenticated key exchange. In: Shacham, H., Boldyreva, A. (eds.) Advances in Cryptology - CRYPTO 2018. vol. 10992, pp. 95–125. Springer (2018)

[26] Groce, A., Katz, J.: A new framework for efficient password-based authenticated key exchange. In: CCS 2010. pp. 516–525. ACM (2010)

[27] 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)

[28] Günther, C.G.: An identity-based key-exchange protocol. In: Quisquater, J., Vandewalle, J. (eds.) EUROCRYPT 1989. vol. 434, pp. 29–37. Springer (1989)

[29] Han, S., Jager, T., Kiltz, E., Liu, S., Pan, J., Riepel, D., Schäge, S.: Authenticated key exchange and signatures with tight security in the standard model. In: Advances in Cryptology - CRYPTO 2021. vol. 12828, pp. 670–700. Springer (2021)

[30] Hao, F., Ryan, P.Y.A.: J-PAKE: authenticated key exchange without PKI. Trans. Comput. Sci. **11**, 192–206 (2010)

[31] Harkins, D.: Dragonfly key exchange. RFC **7664**, 1–18 (2015)

[32] Hesse, J.: Separating symmetric and asymmetric password-authenticated key exchange. In: SCN 2020. vol. 12238, pp. 579–599. Springer (2020)

[33] 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)

[34] 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

[35] Jablon, D.P.: Strong password-only authenticated key exchange. Comput. Commun. Rev. **26**(5), 5–26 (1996)

[36] 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)

[37] 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)

[38] 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)

[39] 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)

[40] 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)

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

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

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

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

[45] 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)

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

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

[48] 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)

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

[50] 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)

Supplementary Material

The organization of supplementary material is as follows.

- A: Functionalities of ideal ciphers and random oracles.
- B: Functionality $\mathcal{F}_{\mathsf{le\text{-}pake}}$ for lazy-extraction PAKE.
- C: Proof of Theorem 5 (security of 2DH-SPAKE2).
- D: More protocols (EKE [10, 9], SPAKE2 [4], PPK [41], and 2DH-PPK).

A Ideal Ciphers and Random Oracles

A.1 Ideal Cipher

The ideal cipher is an idealized computation model for block ciphers [20]. Fig. 10 shows the functionality of an ideal cipher (E, D), with key space $\mathcal{K}$, plaintext space $\mathcal{M}$ and ciphertext space $\mathcal{E}$.

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

Upon receiving a query $\mathsf{E}(k, m)$:

If there exists a record (k, m, e): return e.

Otherwise: sample $e \xleftarrow{\$} \mathcal{E}$, store (k, m, e) and return e.

Upon receiving a query $\mathsf{D}(k, e)$:

If there exists a record (k, m, e): return m.

Otherwise: sample $m \xleftarrow{\$} \mathcal{M}$, store (k, m, e) and return m.

Fig. 10. The ideal cipher functionality $\mathcal{F}_{\mathsf{ic}}$.

A.2 Random Oracle

The functionality of a random oracle $\mathsf{H} : \mathcal{X} \mapsto \mathcal{Y}$ is shown in Fig. 11.

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

Upon receiving a query $\mathsf{H}(x)$:

If there exists a record (x, y): return y.

Otherwise: sample $y \xleftarrow{\$} \mathcal{Y}$, store (x, y) and return y.

Fig. 11. The random oracle functionality $\mathcal{F}_{\mathsf{ro}}$.

B Relaxed UC Framework for PAKE

For some PAKE protocols like SPAKE2 [4], it is hard to prove their security in the (standard) UC framework, due to the "perfect hiding" property of transcript messages. To circumvent this problem, Abdalla et al. [1] relaxed the definition of PAKE functionality to a so-called lazy-extraction PAKE (lePAKE) $\mathcal{F}_{\mathsf{le\text{-}pake}}$. Under this relaxed model, Abdalla et al. proved that SPEKE [36], SPAKE2 [4] and TBPEKE [42], are UC-secure.

Late password guesses. In $\mathcal{F}_{\mathsf{le\text{-}pake}}$, the simulator Sim is allowed to make a password guess even *after* the session is completed and a key k is allocated. More precisely, in the ideal world, once Sim detects an active attack in one client/server instance, but it cannot extract the password guess "hidden" in it, Sim sends a $\mathsf{RegisterTest}$ query to $\mathcal{F}_{\mathsf{le\text{-}pake}}$ so that this instance will be marked with a special flag tested. After that $\mathcal{F}_{\mathsf{le\text{-}pake}}$ allocates a random key k to this instance and the session terminates. Recall that the session key k should be known to the adversary, only when it succeeds in guessing the password in this actively attacked session. Sim may be able to "know" the guess at some time after the client/server instance is completed. And if the guess is correct, Sim has to know the random key k allocated to this instance before, to make the view of the adversary be consistent (e.g., the view of random oracle H), and that is captured by the $\mathsf{LateTestPwd}$ interface in $\mathcal{F}_{\mathsf{le\text{-}pake}}$.

Meanwhile, the $\mathsf{FreshKey}$ interface in $\mathcal{F}_{\mathsf{pake}}$ is replaced by the NewKey interface in $\mathcal{F}_{\mathsf{le\text{-}pake}}$, which allocates a random key key to a client/server instance, but additionally allows concealing this key to Sim later when Sim succeeds in a postponed password guess.

We need to point out that, making a password guess after the key is established is somewhat contrast to the intuition what an ideal functionality of PAKE should be. Intuitively, the simulator (together with the ideal functionality) should decide immediately at the time a session key is generated, whether it is a fresh key, a copy of a fresh key, or a compromised key. Nevertheless, such relaxed notion still provides meaningful security in the password-only setting. See [1, 47] for further discussions.

Weak forward security. Abdalla et al. [1] proved that any protocol realizing $\mathcal{F}_{\mathsf{le\text{-}pake}}$ satisfies Weak Forward Security (wFS, a.k.a. weak forward secrecy). Here wFS means that the perfect forward security holds under passive/eavesdropping attacks.

C Proof of Theorem 5

We prove the security of 2DH-SPAKE2 in this section. First we extend the strong 2DH problem (cf. Definition 2) to the so-called strong extended 2DH assumption as follows, and show its equivalence to the CDH problem.

Functionality $\mathcal{F}_{\mathsf{le\text{-}pake}}$

The functionality $\mathcal{F}_{\mathsf{le\text{-}pake}}$ 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".

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

If there is a fresh record $(P, iid, Q, \cdot)$: mark it as $\mathsf{interrupted}$ and flag it tested.

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

If there is a record (P, iid, Q, pw) with flag tested and a record (P, Q, sid, k): remove tested, and

- If $\mathsf{pw}' = \mathsf{pw}$, return k to Sim.
- If $\mathsf{pw}' \neq \mathsf{pw}$, sample a random key k' and return it to Sim.

Key Generation

Upon receiving a query $(\mathsf{NewKey}, P, iid, sid)$ **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 (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. 12. The lazy-extraction PAKE functionality $\mathcal{F}_{\mathsf{le\text{-}pake}}$ [1]. The difference with $\mathcal{F}_{\mathsf{pake}}$ is highlighted in gray .

Definition 5. *For any adversary $\mathcal{A}$, its advantage in solving the strong extended twin DH (strong ex2DH) problem is defined as*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{st\text{-}ex2DH}}_{\mathbb{G},\mathcal{A}}(\lambda) := \Pr[\bar{x}_1, \bar{x}_2, \bar{y}_1, \bar{y}_2 \xleftarrow{\$} \mathbb{Z}_q : \mathcal{A}^{\textsc{ex2DH}_X(\cdot),\textsc{ex2DH}_Y(\cdot)}(g, g^{\bar{x}_1}, g^{\bar{x}_2}, g^{\bar{y}_1}, g^{\bar{y}_2}) \\ = (g^{\bar{x}_1\bar{y}_1}, g^{\bar{x}_1\bar{y}_2}, g^{\bar{x}_2\bar{y}_1}, g^{\bar{x}_2\bar{y}_2})],\end{aligned}$$

where the decision oracle $\textsc{ex2DH}_X(\cdot)$ *inputs* $(Y_1, Y_2, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2})$ *and outputs 1 if* $(Z_{1,1}, Z_{2,1}) = \mathsf{2DH}(\bar{X}_1, \bar{X}_2, Y_1) \wedge (Z_{1,2}, Z_{2,2}) = \mathsf{2DH}(\bar{X}_1, \bar{X}_2, Y_2)$ *and 0 otherwise, and* $\textsc{ex2DH}_Y(\cdot)$ *inputs* $(X_1, X_2, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2})$ *and outputs 1 if* $(Z_{1,1}, Z_{1,2}) = \mathsf{2DH}(X_1, \bar{Y}_1, \bar{Y}_2) \wedge (Z_{2,1}, Z_{2,2}) = \mathsf{2DH}(X_2, \bar{Y}_1, \bar{Y}_2)$ *and 0 otherwise.*

Theorem 6. *For any PPT adversary $\mathcal{A}$ against the strong ex2DH problem, there exists a PPT algorithm $\mathcal{B}$ against the CDH problem such that* $\mathsf{Adv}^{\mathsf{st\text{-}ex2DH}}_{\mathbb{G},\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{CDH}}_{\mathbb{G},\mathcal{B}}(\lambda) + 2Q/q$, *where Q is the maximum number of decision oracle queries* ($\textsc{ex2DH}_X$ *or* $\textsc{ex2DH}_Y$).

Proof. The proof borrows the idea of the trapdoor test in [18]. Consider a reduction algorithm $\mathcal{B}$ that gets a CDH problem instance (g^α, g^β). It provides the strong ex2DH problem adversary $\mathcal{A}$ with $(\bar{X}_1, \bar{X}_2, \bar{Y}_1, \bar{Y}_2)$ as well as oracles $\textsc{ex2DH}_X$ and $\textsc{ex2DH}_Y$, and tries to solve the CDH problem with $\mathcal{A}$'s ability. $\mathcal{B}$ randomly samples $s, t, u, v \xleftarrow{\$} \mathbb{Z}_q$, and sets $\bar{X}_1 := g^\alpha, \bar{X}_2 := g^s \cdot \bar{X}_1^t = g^{s+\alpha t}, \bar{Y}_1 := g^\beta, \bar{Y}_2 := g^u \cdot \bar{Y}_1^v = g^{u+\beta v}$. Besides, it simulates $\textsc{ex2DH}_X$ and $\textsc{ex2DH}_Y$ using trapdoors s, t, u, v as follows.

- $\textsc{ex2DH}_X(Y_1, Y_2, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2})$: if $Z_{2,1} = Y_1^s \cdot Z_{1,1}^t \;\wedge\; Z_{2,2} = Y_2^s \cdot Z_{1,2}^t$ return 1; otherwise return 0.
- $\textsc{ex2DH}_Y(X_1, X_2, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2})$: if $Z_{1,2} = X_1^u \cdot Z_{1,1}^v \wedge Z_{2,2} = X_2^u \cdot Z_{2,1}^t$ return 1; otherwise return 0.

The trapdoor test theorem in [18] tells us that for any $Y_1, Z_{1,1}, Z_{2,1} \in \mathbb{G}$, if

$$Z_{2,1} = Y_1^s \cdot Z_{1,1}^t \tag{1}$$

holds, then

$$Z_{1,1} = Y_1^{\bar{x}_1} \wedge Z_{2,1} = Y_1^{\bar{x}_2} \tag{2}$$

holds, with error probability at most $1/q$.

Theorem 6 holds immediately by the union bound.

Now we prove Theorem 5.

Proof. Similar to the proofs of Theorem 2 and 3, our main task is to construct a PPT simulator Sim, which has access to the ideal functionality $\mathcal{F}_{\mathsf{le\text{-}pake}}$ and interactions with the environment $\mathcal{Z}$, and simulates the real world 2DH-SPAKE2 protocol interactions among the adversary $\mathcal{A}$, parties, and the environment $\mathcal{Z}$. To this end, Sim needs to simulate common reference strings (CRS), simulate honestly generated messages from real parties, response adversarial messages approximately, and simulate random oracle H, as shown in Fig. 3. The functionality $\mathcal{F}_{\mathsf{le\text{-}pake}}$ provides information to Sim through interfaces including TestPW, $\mathsf{RegisterTest}$, $\mathsf{LateTestPwd}$, $\mathsf{NewClient}$, $\mathsf{NewServer}$, NewKey, $\mathsf{CopyKey}$, and $\mathsf{CorruptKey}$. Recall that Sim has no secret inputs (i.e., passwords).

The full description of the simulator Sim is given in Fig. 13 and 14. 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 two adjacent games are indistinguishable from $\mathcal{Z}$'s prospective of view.

We consider the scenario of multi-users and multi instances as before, and use $(\mathsf{C}^{(i)}, iid^{(i)})$ (resp., $(\mathsf{S}^{(j)}, iid^{(j)})$) to specify a client (resp., server) instance. Good/bad client instances are defined the same as those in Section 3, but the definition of linked instances is a bit different, since in 2DH-SPAKE2, both the client and the server can initialize a protocol execution.

To capture this property and for better presentation of Sim, we divide a client (resp., server) instance into two phases, of which the first phase generates $X_1^*||X_2^*$ (resp., $Y_1^*||Y_2^*$) and sends it out, and the second phase computes the session key when receiving a message from its intended partner. Meanwhile, we define linked instances as follows.

Linked instances. We say that a client (resp., server) instance (P, iid) is linked to a server (resp., client) instance (Q, iid'), if (Q, iid') generates $Z_1^*||Z_2^*$ which is also received by one instance (P, iid) of its intended partner P.

Further more, we define correct DH values as follows.

Correct DH values. Let $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ and pw be a password (not necessarily the correct password between $\mathsf{C}^{(i)}$ and $\mathsf{S}^{(j)}$). We say $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are the correct DH values w.r.t. sid and pw, if

$$Z_{1,1} = \mathsf{CDH}\left(\frac{X_1^*}{M_1^{\mathsf{pw}}}, \frac{Y_1^*}{N_1^{\mathsf{pw}}}\right),\ Z_{1,2} = \mathsf{CDH}\left(\frac{X_1^*}{M_1^{\mathsf{pw}}}, \frac{Y_2^*}{N_2^{\mathsf{pw}}}\right),$$
$$Z_{2,1} = \mathsf{CDH}\left(\frac{X_2^*}{M_2^{\mathsf{pw}}}, \frac{Y_1^*}{N_1^{\mathsf{pw}}}\right),\ Z_{2,2} = \mathsf{CDH}\left(\frac{X_2^*}{M_2^{\mathsf{pw}}}, \frac{Y_2^*}{N_2^{\mathsf{pw}}}\right).$$

Game 0. This is the real experiment $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$ in the proof of Theorem 5. 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, 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. 13 and 14. In **Game 1**, Sim still needs to take passwords as inputs. With the help of passwords, it perfectly simulates the executions in $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$. Meanwhile, it generates CRS, and emulates the random oracle H for parties and adversary $\mathcal{A}$ as follows.

Sim maintains lists $\mathcal{T}, \mathcal{D}, \mathcal{L}_{\mathsf{H}}, \mathcal{IF}, \mathcal{DL}$ in the simulation.

- $\mathcal{T}$: store messages sent by client/server instances.
- $\mathcal{D}$: store messages received by client/server instances.
- $\mathcal{L}_{\mathsf{H}}$: store records w.r.t. random oracle H.
- $\mathcal{IF}$: store information w.r.t. transcripts, (possible) password guesses, and session keys.
- $\mathcal{DL}$: store discrete logarithms.

PAKE Sessions

```
on $(\mathsf{NewClient}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, b)$ from $\mathcal{F}_{\mathsf{le\text{-}pake}}$:
    $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$, $X_1 := g^{x_1}$, $X_2 := g^{x_2}$, $\mathcal{DL} := \mathcal{DL} \cup \{(X_1||X_2, x_1||x_2)\}$, $\mathcal{T} := \mathcal{T} \cup \{(\mathsf{C}^{(i)}, iid^{(i)}, X_1||X_2)\}$,
    and send $X_1||X_2$ from $\mathsf{C}^{(i)}$ to $\mathcal{A}$.
    If $b = 1$: mark $(\mathsf{C}^{(i)}, iid^{(i)})$ as correct-pw. // client $\mathsf{C}^{(i)}$ correctly inputs the password
on $(\mathsf{NewServer}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)})$ from $\mathcal{F}_{\mathsf{le\text{-}pake}}$:
    $y_1, y_2 \xleftarrow{\$} \mathbb{Z}_q$, $Y_1 := g^{y_1}$, $Y_2 := g^{y_2}$, $\mathcal{DL} := \mathcal{DL} \cup \{(Y_1||Y_2, y_1||y_2)\}$, $\mathcal{T} := \mathcal{T} \cup \{(\mathsf{S}^{(j)}, iid^{(j)}, Y_1||Y_2)\}$.
    Send $Y_1||Y_2$ from $\mathsf{S}^{(j)}$ to $\mathcal{A}$.
on $X_1^*||X_2^*$ from $\mathcal{A}$ as a client message from $\mathsf{C}^{(i)}$ to $(\mathsf{S}^{(j)}, iid^{(j)})$:
    Retrieve $(\mathsf{S}^{(j)}, iid^{(j)}, Y_1||Y_2) \in \mathcal{T}$ and $(Y_1||Y_2, y_1||y_2) \in \mathcal{DL}$, $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1||Y_2$, $\mathcal{D} :=$
    $\mathcal{D} \cup \{(\mathsf{S}^{(j)}, iid^{(j)}, X_1^*||X_2^*)\}$.
    If $\exists(\mathsf{C}^{(i)}, iid^{(i)}, X_1^*||X_2^*) \in \mathcal{T}$ and $(\mathsf{C}^{(i)}, iid^{(i)})$ is correct-pw:
        If $\mathsf{Sim}$ has queried $(\mathsf{NewKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$: send $(\mathsf{CopyKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$;
        Otherwise, send $(\mathsf{NewKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$.
    Else if $\exists(sid, \mathsf{pw}', \mathsf{key}) \in \mathcal{IF}$: ask $(\mathsf{TestPW}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{pw}')$:
        If $\mathcal{F}_{\mathsf{le\text{-}pake}}$ returns "correct guess": send $(\mathsf{CorruptKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$;
        If $\mathcal{F}_{\mathsf{le\text{-}pake}}$ returns "wrong guess": send $(\mathsf{NewKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$.
    Else: Send $(\mathsf{RegisterTest}, \mathsf{S}^{(j)}, iid^{(j)})$ and $(\mathsf{NewKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$.
on $Y_1^*||Y_2^*$ from $\mathcal{A}$ as a server message from $\mathsf{S}^{(j)}$ to $(\mathsf{C}^{(i)}, iid^{(i)})$:
    Retrieve $(\mathsf{C}^{(i)}, iid^{(i)}, X_1||X_2) \in \mathcal{T}$ and $(X_1||X_2, x_1||x_2) \in \mathcal{DL}$, $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1||X_2||Y_1^*||Y_2^*$,
    $\mathcal{D} := \mathcal{D} \cup \{(\mathsf{C}^{(i)}, iid^{(i)}, Y_1^*||Y_2^*)\}$.
    If $\exists(\mathsf{S}^{(i)}, iid^{(j)}, Y_1^*||Y_2^*) \in \mathcal{T}$ and $(\mathsf{C}^{(i)}, iid^{(i)})$ is correct-pw:
        If $\mathsf{Sim}$ has queried $(\mathsf{NewKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$: send $(\mathsf{CopyKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$;
        Otherwise, send $(\mathsf{NewKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$.
    Else If $\exists(sid, \mathsf{pw}', \mathsf{key}) \in \mathcal{IF}$: ask $(\mathsf{TestPW}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{pw}')$:
        If $\mathcal{F}_{\mathsf{le\text{-}pake}}$ returns "correct guess": send $(\mathsf{CorruptKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid, \mathsf{key})$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$;
        If $\mathcal{F}_{\mathsf{le\text{-}pake}}$ returns "wrong guess": send $(\mathsf{NewKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$.
    Else: Send $(\mathsf{RegisterTest}, \mathsf{C}^{(i)}, iid^{(i)})$ and $(\mathsf{NewKey}, \mathsf{C}^{(i)}, iid^{(i)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$.
```

Fig. 13. Simulator Sim for 2DH-SPAKE2 in the proof of Theorem 5, part 1.

```
$\mathsf{Sim}$ maintains lists $\mathcal{T}, \mathcal{D}, \mathcal{L}_\mathsf{H}, \mathcal{IF}, \mathcal{DL}$ in the simulation.

On CRS and Random Oracles
Generating CRS:
    $m_1, m_2, n_1, n_2 \xleftarrow{\$} \mathbb{Z}_q$, $(M_1, M_2, N_1, N_2) := (g^{m_1}, g^{m_2}, g^{n_1}, g^{n_2})$ and return $(M_1, M_2, N_1, N_2)$.
on $\mathsf{H}(\mathsf{C}^{(i)}, \mathsf{S}^{(j)}, X_1^*, X_2^*, Y_1^*, Y_2^*, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$ from $\mathcal{A}$:
    $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$.
    If $\exists(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}, \mathsf{key}) \in \mathcal{L}_\mathsf{H}$: return $\mathsf{key}$.
    $\mathsf{key} \xleftarrow{\$} \mathcal{K}$.
    If $\exists(\mathsf{C}^{(i)}, iid^{(i)}, X_1^*||X_2^*) \in \mathcal{T}$, retrieve $(X_1^*||X_2^*, x_1||x_2) \in \mathcal{DL}$, and:
        If $Z_{1,1} := (Y_1^*/N_1^{\mathsf{pw}})^{(x_1 - m_1 \cdot \mathsf{pw})}$, $Z_{1,2} := (Y_2^*/N_2^{\mathsf{pw}})^{(x_1 - m_1 \cdot \mathsf{pw})}$, $Z_{2,1} := (Y_1^*/N_1^{\mathsf{pw}})^{(x_2 - m_2 \cdot \mathsf{pw})}$, $Z_{2,2} := (Y_2^*/N_2^{\mathsf{pw}})^{(x_2 - m_2 \cdot \mathsf{pw})}$:
            If $\exists(\mathsf{C}^{(i)}, iid^{(i)}, Y_1^*||Y_2^*) \in \mathcal{D}$: ask $(\mathsf{LateTestPwd}, \mathsf{C}^{(i)}, sid, \mathsf{pw})$ to get $\mathsf{reply}$, go to $\mathsf{FINAL}$.
            Otherwise: $\mathcal{IF} := \mathcal{IF} \cup \{(sid, \mathsf{pw}, \mathsf{key})\}$, go to $\mathsf{FINAL}$.
    If $\exists(\mathsf{S}^{(j)}, iid^{(j)}, Y_1^*||Y_2^*) \in \mathcal{T}$: retrieve $(Y_1^*||Y_2^*, y_1||y_2) \in \mathcal{DL}$, and:
        If $Z_{1,1} := (X_1^*/M_1^{\mathsf{pw}})^{(y_1 - n_1 \cdot \mathsf{pw})}$, $Z_{1,2} := (X_1^*/M_1^{\mathsf{pw}})^{(y_2 - n_2 \cdot \mathsf{pw})}$, $Z_{2,1} := (X_2^*/M_2^{\mathsf{pw}})^{(y_1 - n_1 \cdot \mathsf{pw})}$, $Z_{2,2} := (X_2^*/M_2^{\mathsf{pw}})^{(y_2 - n_2 \cdot \mathsf{pw})}$:
            If $\exists(\mathsf{S}^{(j)}, iid^{(j)}, X_1^*||X_2^*) \in \mathcal{D}$: ask $(\mathsf{LateTestPwd}, \mathsf{S}^{(j)}, sid, \mathsf{pw})$ to get $\mathsf{reply}$, go to $\mathsf{FINAL}$.
            Otherwise: $\mathcal{IF} := \mathcal{IF} \cup \{(sid, \mathsf{pw}, \mathsf{key})\}$, go to $\mathsf{FINAL}$.
        Otherwise: go to $\mathsf{FINAL}$.
    In any other case: go to $\mathsf{FINAL}$.

    $\mathsf{FINAL}$: Set $\mathsf{key} := \mathsf{reply}$ if $\mathsf{reply}$ exists. Record $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}, \mathsf{key})$ in $\mathcal{L}_\mathsf{H}$ and return $\mathsf{key}$.
```

Fig. 14. Simulator Sim for 2DH-SPAKE2 in the proof of Theorem 5, part 2.

- Generation of CRS: Sample $m_1, m_2, n_1, n_2 \xleftarrow{\$} \mathbb{Z}_q$, set $(M_1, M_2, N_1,\ N_2) := (g^{m_1}, g^{m_2}, g^{n_1}, g^{n_2})$, and return (M_1, M_2, N_1, N_2).
- On $\mathsf{H}(\mathsf{C}, \mathsf{S}, X_1^*, X_2^*, Y_1^*, Y_2^*, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$: Let $sid := \mathsf{C}||\mathsf{S}||X_1^*||X_2^*||Y_1^*||Y_2^*$. If there exists $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}, \mathsf{key}) \in \mathcal{L}_\mathsf{H}$, return key; otherwise, $\mathsf{key} \xleftarrow{\$} \mathcal{K}$, add $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}, \mathsf{key})$ in $\mathcal{L}_\mathsf{H}$ and return key.

Obviously we have

$$\Pr[\textbf{Game 1} \Rightarrow 1] = \Pr[\textbf{Game 0} \Rightarrow 1].$$

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, if a good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ and a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ are linked to each other, then Sim generates a random session key for one of them, and copies the key as the session key for the other.

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 $\mathsf{H}(sid, \hat{Z}_{1,1}, \hat{Z}_{1,2}, \hat{Z}_{2,1}, \hat{Z}_{2,2}, \hat{\mathsf{pw}})$ such that $\hat{Z}_{1,1}, \hat{Z}_{1,2}, \hat{Z}_{2,1}, \hat{Z}_{2,2}$ are correct DH values w.r.t. sid and $\hat{\mathsf{pw}}$, where $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ with $X_1^*, X_2^*, Y_1^*, Y_2^*$ 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 ex2DH problem.

The reduction works as follows. $\mathcal{B}_1$ receives the strong ex2DH challenge $(\bar{X}_1, \bar{X}_2, \bar{Y}_1, \bar{Y}_2)$, as well as oracles $\textsc{ex2DH}_X$ and $\textsc{ex2DH}_Y$. Then it simulates **Game 2** as below.

- The simulation of H is the same as that in **Game 2**.
- For the simulation of good client instance $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ generating $X_1^*||X_2^*$: $\mathcal{B}_1$ samples $a_{s,1}^{(i)}, a_{s,2}^{(i)} \xleftarrow{\$} \mathbb{Z}_q$, sets $X_1^* := \bar{X}_1 g^{a_{s,1}^{(i)}} M_1^{\hat{\mathsf{pw}}} = g^{\bar{x}_1 + a_{s,1}^{(i)}} M_1^{\hat{\mathsf{pw}}}$, $X_2^* := \bar{X}_2 g^{a_{s,2}^{(i)}} M_2^{\hat{\mathsf{pw}}} = g^{\bar{x}_2 + a_{s,2}^{(i)}} M_2^{\hat{\mathsf{pw}}}$, and sends $X_1^*||X_2^*$ out.
- For the simulation of server instance $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ generating $Y_1^*||Y_2^*$: $\mathcal{B}_1$ samples $b_{t,1}^{(j)}, b_{t,2}^{(j)} \xleftarrow{\$} \mathbb{Z}_q$, sets $Y_1^* := \bar{Y}_1 g^{b_{t,1}^{(j)}} N_1^{\hat{\mathsf{pw}}} = g^{\bar{y}_1 + b_{t,1}^{(j)}} N_1^{\hat{\mathsf{pw}}}$, $Y_2^* := \bar{Y}_2 g^{b_{t,2}^{(j)}} N_2^{\hat{\mathsf{pw}}} = g^{\bar{y}_2 + b_{t,2}^{(j)}} N_2^{\hat{\mathsf{pw}}}$, and sends $Y_1^*||Y_2^*$ out.
- For the simulation of good client instance $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ generating the session key: Let $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ with $X_1^*, X_2^*, Y_1^*, Y_2^*$ the transcripts. There are two cases.
  - If $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ and server instance $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ are linked to each other, then $\mathcal{B}_1$ generates a random key for it if $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ has not computed the session key yet, or assigns the same key of $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ to it.
  - If $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ receives an adversarially generated $Y_1^*||Y_2^*$, then $\mathcal{B}_1$ computes the session key with the help of decisional oracle $\textsc{ex2DH}_X$. Concretely, $\mathcal{B}_1$ first computes $X_1 := X_1^*/M_1^{\hat{\mathsf{pw}}}, X_2 := X_2^*/M_2^{\hat{\mathsf{pw}}}, Y_1 := Y_1^*/N_1^{\hat{\mathsf{pw}}}, Y_2 := Y_2^*/N_2^{\hat{\mathsf{pw}}}$. Then, it uses $\textsc{ex2DH}_X$ to check whether there exists $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \hat{\mathsf{pw}}, \mathsf{key}) \in \mathcal{L}_\mathsf{H}$, such that $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and $\hat{\mathsf{pw}}$. If so, $\mathcal{B}_1$ assigns key as the session key to $(\mathsf{C}^{(i)}, iid^{(i)} = s)$. Otherwise, $\mathcal{B}_1$ randomly samples a key and "views" it as the hash output for the correct input $\mathsf{H}(sid, \mathsf{CDH}(X_1, Y_1) =?, \mathsf{CDH}(X_1, Y_2) =?, \mathsf{CDH}(X_2, Y_1) =?, \mathsf{CDH}(X_2, Y_2) =?, \hat{\mathsf{pw}})$, where ? means that the value is to be determined. If $\mathcal{A}$ later asks $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \hat{\mathsf{pw}})$, then $\mathcal{B}_1$ checks whether $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and $\hat{\mathsf{pw}}$ via the decisional oracle, i.e., it checks whether

$$\textsc{ex2DH}_X(Y_1, Y_2, Z_{1,1}/(Y_1)^{a_{s,1}^{(i)}}, Z_{1,2}/(Y_2)^{a_{s,1}^{(i)}}, Z_{2,1}/(Y_1)^{a_{s,2}^{(i)}}, Z_{2,2}/(Y_2)^{a_{s,2}^{(i)}}) = 1.$$

    If yes, $\mathcal{B}_1$ reprograms the random oracle by replacing $(?, ?, ?, ?)$ with $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$. In this way, the view of $\mathcal{A}$ is consistent.

- For the simulation of server instance $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ generating the session key: Let $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ with $X_1^*, X_2^*, Y_1^*, Y_2^*$ the transcripts. There are two cases.
  - If $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ and good client instance $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ are linked to each other, then $\mathcal{B}_1$ generates a random key for it if $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ has not computed the session key yet, or assigns the same key of $(\mathsf{C}^{(i)}, iid^{(i)} = s)$ to it.
  - If $(\mathsf{S}^{(j)}, iid^{(j)} = t)$ receives an adversarially generated $X_1^*||X_2^*$, then $\mathcal{B}_1$ computes the session key with the help of decisional oracle $\textsc{ex2DH}_Y$ as above, except that the verification equivalence is changed to

$$\textsc{ex2DH}_Y(X_1, X_2, Z_{1,1}/(X_1)^{b_{t,1}^{(j)}}, Z_{1,2}/(X_1)^{b_{t,2}^{(j)}}, Z_{2,1}/(X_2)^{b_{t,1}^{(j)}}, Z_{2,2}/(X_2)^{b_{t,2}^{(j)}}) = 1.$$

- The simulation of bad client instances is the same as that in **Game 2**.

Suppose that bad_1 happens w.r.t. instances $(\mathsf{C}^{(i)}, s)$ and $(\mathsf{S}^{(j)}, t)$ with transcripts $X_1^*||X_2^*||Y_1^*||Y_2^*$, then $\mathcal{A}$ must have asked $\mathsf{H}(sid, \hat{Z}_{1,1}, \hat{Z}_{1,2}, \hat{Z}_{2,1}, \hat{Z}_{2,2}, \hat{\mathsf{pw}})$ s.t. $\hat{Z}_{1,1}, \hat{Z}_{1,2}, \hat{Z}_{2,1}, \hat{Z}_{2,2}$ are correct DH values w.r.t sid and $\hat{\mathsf{pw}}$. Note that $\mathcal{B}_1$ can detect bad_1 with the help of oracles $\textsc{ex2DH}_X, \textsc{ex2DH}_Y$, and trapdoors $a_{s,1}^{(i)}, a_{s,2}^{(i)}, b_{t,1}^{(j)}, b_{t,2}^{(j)}$. Then $\mathcal{B}_1$ can extract the solution

$$\hat{Z}_{1,1}/g^{\bar{x}_1 b_{t,1}^{(j)} + \bar{y}_1 a_{s,1}^{(i)} + a_{s,1}^{(i)} b_{t,1}^{(j)}},\ \hat{Z}_{1,2}/g^{\bar{x}_1 b_{t,2}^{(j)} + \bar{y}_2 a_{s,1}^{(i)} + a_{s,1}^{(i)} b_{t,2}^{(j)}},$$
$$\hat{Z}_{2,1}/g^{\bar{x}_2 b_{t,1}^{(j)} + \bar{y}_1 a_{s,2}^{(i)} + a_{s,2}^{(i)} b_{t,1}^{(j)}},\ \hat{Z}_{2,2}/g^{\bar{x}_2 b_{t,2}^{(j)} + \bar{y}_2 a_{s,2}^{(i)} + a_{s,2}^{(i)} b_{t,2}^{(j)}},$$

to the strong ex2DH problem.

Therefore, we have

$$|\Pr[\textbf{Game 2} \Rightarrow 1] - \Pr[\textbf{Game 1} \Rightarrow 1]| \le \mathsf{Adv}_{\mathbb{G},\mathcal{B}_1}^{\mathsf{st\text{-}ex2DH}}(\lambda).$$

Game 3. (Randomize simulated messages.) In this game, Sim directly samples random messages to simulate the message outputs of client and server instances, and uses the trapdoors m_1, m_2, n_1, n_2 to compute 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 transcript, Sim samples $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$, and computes $X_1^*||X_2^* = g^{x_1}||g^{x_2}$ as the output message.
- For the simulation of a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ generating the transcript, Sim samples $y_1, y_2 \xleftarrow{\$} \mathbb{Z}_q$, and computes $Y_1^*||Y_2^* = g^{y_1}||g^{y_2}$ as the output message.
- For the simulation of a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ generating the session key, let $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ and pw be the (possibly incorrect) password used in this instance. Sim computes the correct DH values $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ w.r.t. sid and pw via

$$Z_{1,1} = (Y_1^*/N_1^{\mathsf{pw}})^{x_1 - m_1 \cdot \mathsf{pw}},\ Z_{1,2} = (Y_2^*/N_2^{\mathsf{pw}})^{x_1 - m_1 \cdot \mathsf{pw}},$$
$$Z_{2,1} = (Y_1^*/N_1^{\mathsf{pw}})^{x_2 - m_2 \cdot \mathsf{pw}},\ Z_{2,2} = (Y_2^*/N_2^{\mathsf{pw}})^{x_2 - m_2 \cdot \mathsf{pw}}.$$

Then Sim generates the session key as $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$.

- For the simulation of a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ generating the session key, let $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ and pw be the (correct) password used in this instance. Sim computes the correct DH values $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ w.r.t. sid and pw via

$$Z_{1,1} = (X_1^*/M_1^{\mathsf{pw}})^{y_1 - n_1 \cdot \mathsf{pw}},\ Z_{1,2} = (X_1^*/M_1^{\mathsf{pw}})^{y_2 - n_2 \cdot \mathsf{pw}},$$
$$Z_{2,1} = (X_2^*/M_2^{\mathsf{pw}})^{y_1 - n_1 \cdot \mathsf{pw}},\ Z_{2,2} = (X_2^*/M_2^{\mathsf{pw}})^{y_2 - n_2 \cdot \mathsf{pw}}.$$

Then Sim generates the session key as $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$.

Obviously **Game 3** and **Game 2** are identical, and we have

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

Game 4. (Randomize keys for client instances in case of incorrect password guesses.) In **Game 4**, Sim changes the generation of session keys for client instances when detecting an active attack.

Concretely, for a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ with transcript $X_1^*||X_2^*||Y_1^*||Y_2^*$, let $\mathsf{S}^{(j)}$ be the intended partner and pw be the (possibly incorrect) password used in this instance. Sim sets $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$, and generates the session key 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 key for both $(\mathsf{C}^{(i)}, iid^{(i)})$ and $(\mathsf{S}^{(j)}, iid^{(j)})$, just as that in **Game 3**.

Case (C.2). If $(\mathsf{C}^{(i)}, iid^{(i)})$ is not linked to any instance of $\mathsf{S}^{(j)}$, or $(\mathsf{C}^{(i)}, iid^{(i)})$ is bad[11]. We further divide it into the following two subcases.

Case (C.2.1). Sim retrieves the *first* record $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}', \mathsf{key}) \in \mathcal{L}_{\mathsf{H}}$ s.t. $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw'. If such a record exists and $\mathsf{pw}' = \mathsf{pw}$, then Sim sets the session key to be key. And if $\mathsf{pw}' \neq \mathsf{pw}$, then Sim sets the session key to be random.

Case (C.2.2). If the record does not exist, Sim samples a random key as the session key of $(\mathsf{C}^{(i)}, iid^{(i)})$. Meanwhile, it keeps looking on the *first* hash query $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}')$ s.t. $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw'. If $\mathsf{pw}' = \mathsf{pw}$, then Sim reprograms the random oracle by setting $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}) = \mathsf{key}$. And if $\mathsf{pw}' \neq \mathsf{pw}$, then Sim returns a random key and records it in the hash list $\mathcal{L}_{\mathsf{H}}$.

According to the protocol specification, the session key of a client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ is computed as $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$ s.t. $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$

[11] As described in Subsec. 3.1, when considering security, a bad client instance (the case when the client mistypes the password) can be simply viewed as an "online" attack.

are correct DH values w.r.t. sid and pw, where pw is the password used in $(\mathsf{C}^{(i)}, \mathit{iid}^{(i)})$. That is, in either Case (C.2.1) or Case (C.2.2), if the first query satisfies $\mathsf{pw}' = \mathsf{pw}$ or the query does not exists, then the key generation performs the same as in **Game 3**.

Define bad_2 as the event that $\mathcal{A}$ has asked more than one hash query of the form $\mathsf{H}(\mathit{sid}, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}')$ s.t. $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw', and $\mathsf{pw}' \neq \mathsf{pw}$ in the first such query, but $\mathsf{pw}' = \mathsf{pw}$ in some subsequent query.

Obviously **Game 4** and **Game 3** are the same unless bad_2 happens. Next we show that if bad_2 happens, then we can construct a reduction algorithm $\mathcal{B}_2$ to solve the strong ex2DH problem.

The reduction works as follows. $\mathcal{B}_2$ receives the strong ex2DH challenge $(\bar{X}_1, \bar{X}_2, \bar{Y}_1, \bar{Y}_2)$, as well as oracles $\textsc{ex2DH}_X$ and $\textsc{ex2DH}_Y$. Then it simulates **Game 4** as below.

- For the generation of CRS, $\mathcal{B}_2$ randomly samples m_1, m_2, computes $M_1 := g^{m_1}, M_2 := g^{m_2}$, sets $N_1 := \bar{Y}_1, N_2 := \bar{Y}_2$, and outputs (M_1, M_2, N_1, N_2).
- For the simulation of client instance $(\mathsf{C}^{(i)}, \mathit{iid}^{(i)})$ generating $X_1^*||X_2^*$: $\mathcal{B}_2$ samples $a_1, a_2 \xleftarrow{\$} \mathbb{Z}_q$, sets $X_1^* := \bar{X}_1 g^{a_1} = g^{\bar{x}_1 + a_1}$, $X_2^* := \bar{X}_2 g^{a_2} = g^{\bar{x}_2 + a_2}$, and sends $X_1^*||X_2^*$ out.
- For the simulation of server instance $(\mathsf{S}^{(j)}, \mathit{iid}^{(j)})$ generating $Y_1^*||Y_2^*$: $\mathcal{B}_2$ samples $y_1, y_2 \xleftarrow{\$} \mathbb{Z}_q$, sets $Y_1^* := g^{y_1}$, $Y_2^* := g^{y_2}$, adds $(Y_1^*||Y_2^*, y_1||y_2)$ into $\mathcal{DL}$, and sends $Y_1^*||Y_2^*$ out.
- For the simulation of client instance $(\mathsf{C}^{(i)}, \mathit{iid}^{(i)})$ generating the session key: Let $\mathit{sid} = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ with $X_1^*, X_2^*, Y_1^*, Y_2^*$ the transcripts. There are two cases.
    - If $(\mathsf{C}^{(i)}, \mathit{iid}^{(i)})$ and server instance $(\mathsf{S}^{(j)}, \mathit{iid}^{(j)})$ are linked to each other, and $(\mathsf{C}^{(i)}, \mathit{iid}^{(i)})$ is good, then $\mathcal{B}_2$ generates a random key for it if $(\mathsf{S}^{(j)}, \mathit{iid}^{(j)})$ has not computed the session key yet, or assigns the same key of $(\mathsf{S}^{(j)}, \mathit{iid}^{(j)})$ to it, just as **Game 4**.
    - Otherwise, $\mathcal{B}_2$ computes the session key with the help of decisional oracle $\textsc{ex2DH}_X$. $\mathcal{B}_2$ first computes $X_1 := X_1^*/M_1^{\mathsf{pw}}, X_2 := X_2^*/M_2^{\mathsf{pw}}, Y_1 := Y_1^*/N_1^{\mathsf{pw}}, Y_2 := Y_2^*/N_2^{\mathsf{pw}}$. Then, it uses $\textsc{ex2DH}_X$ to check whether there exists $(\mathit{sid}, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}, \mathsf{key}) \in \mathcal{L}_{\mathsf{H}}$, such that $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw. If so, $\mathcal{B}_2$ assigns key as the session key to $(\mathsf{C}^{(i)}, \mathit{iid}^{(i)})$. Otherwise, $\mathcal{B}_2$ randomly samples a key and "views" it as the hash output for the correct input $\mathsf{H}(\mathit{sid}, \mathsf{CDH}(X_1, Y_1) = ?, \mathsf{CDH}(X_1, Y_2) = ?, \mathsf{CDH}(X_2, Y_1) = ?, \mathsf{CDH}(X_2, Y_2) = ?, \mathsf{pw})$, where ? means that the value is to be determined. If $\mathcal{A}$ later asks $\mathsf{H}(\mathit{sid}, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$, then $\mathcal{B}_2$ checks whether $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw via the decisional oracle, i.e., it checks whether

$$\textsc{ex2DH}_X(Y_1, Y_2, Z_{1,1} \cdot Y_1^{m_1 \cdot \mathsf{pw} - a_1}, Z_{1,2} \cdot Y_2^{m_1 \cdot \mathsf{pw} - a_1}, Z_{2,1} \cdot Y_1^{m_2 \cdot \mathsf{pw} - a_2}, Z_{2,2} \cdot Y_2^{m_2 \cdot \mathsf{pw} - a_2}) = 1.$$

      If yes, $\mathcal{B}_1$ reprograms the random oracle by replacing $(?, ?, ?, ?)$ with $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$. In this way, the view of $\mathcal{A}$ is consistent.

- For the simulation of server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ generating the session key: Let $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ with $X_1^*, X_2^*, Y_1^*, Y_2^*$ the transcripts. There are two cases.
  - If $(\mathsf{S}^{(j)}, iid^{(j)})$ and good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ are linked to each other, then $\mathcal{B}_2$ generates a random key for it if $(\mathsf{C}^{(i)}, iid^{(i)})$ has not computed the session key yet, or assigns the same key of $(\mathsf{C}^{(i)}, iid^{(i)})$ to it, just as **Game 4**.
  - Otherwise, $\mathcal{B}_2$ computes $X_1 := X_1^*/M_1^{\mathsf{pw}}, X_2 := X_2^*/M_2^{\mathsf{pw}}, Y_1 := Y_1^*/N_1^{\mathsf{pw}}$, $Y_2 := Y_2^*/N_2^{\mathsf{pw}}$. Then it generates the session key with the help of decisional oracle $\textsc{ex2DH}_Y$ as above, except that the verification formula is changed to

$$\textsc{ex2DH}_Y(X_1, X_2, (X_1^{y_1}/Z_{1,1})^{\mathsf{pw}^{-1}}, (X_1^{y_2}/Z_{1,2})^{\mathsf{pw}^{-1}}, (X_2^{y_1}/Z_{2,1})^{\mathsf{pw}^{-1}}, (X_2^{y_2}/Z_{2,2})^{\mathsf{pw}^{-1}}) = 1.$$

- The simulation of H is the same as that in **Game 4**.

If bad_2 happens, then there exists two queries $(sid, Z'_{1,1}, Z'_{1,2}, Z'_{2,1}, Z'_{2,2}, \mathsf{pw}')$ and $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$, such that $Z'_{1,1}, Z'_{1,2}, Z'_{2,1}, Z'_{2,2}$ are correct DH values w.r.t. sid and pw', and $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw.

According to $\mathcal{B}_2$'s simulation above, we get

$$\begin{aligned} Z_{1,1}/Z'_{1,1} &= \frac{\mathsf{CDH}(X_1^*/M_1^{\mathsf{pw}}, Y_1^*/N_1^{\mathsf{pw}})}{\mathsf{CDH}(X_1^*/M_1^{\mathsf{pw}'}, Y_1^*/N_2^{\mathsf{pw}'})} \\ &= \frac{g^{(\bar{x}_1 + a_1 - m_1 \cdot \mathsf{pw})(y_1 - \bar{y}_1 \cdot \mathsf{pw})}}{g^{(\bar{x}_1 + a_1 - m_1 \cdot \mathsf{pw}')(y_1 - \bar{y}_1 \cdot \mathsf{pw}')}} \\ &= g^{(m_1 y_1 + a_1 \bar{y}_1) \cdot (\mathsf{pw}' - \mathsf{pw})} \cdot g^{m_1 \bar{y}_1 \cdot (\mathsf{pw}^2 - \mathsf{pw}'^2)} \cdot g^{\bar{x}_1 \bar{y}_1 \cdot (\mathsf{pw}' - \mathsf{pw})}. \end{aligned}$$

Similarly,

$$\begin{aligned} Z_{1,2}/Z'_{1,2} &= g^{(m_1 y_2 + a_1 \bar{y}_2) \cdot (\mathsf{pw}' - \mathsf{pw})} \cdot g^{m_1 \bar{y}_2 \cdot (\mathsf{pw}^2 - \mathsf{pw}'^2)} \cdot g^{\bar{x}_1 \bar{y}_2 \cdot (\mathsf{pw}' - \mathsf{pw})}, \\ Z_{2,1}/Z'_{2,1} &= g^{(m_2 y_1 + a_2 \bar{y}_1) \cdot (\mathsf{pw}' - \mathsf{pw})} \cdot g^{m_2 \bar{y}_1 \cdot (\mathsf{pw}^2 - \mathsf{pw}'^2)} \cdot g^{\bar{x}_2 \bar{y}_1 \cdot (\mathsf{pw}' - \mathsf{pw})}, \\ Z_{2,2}/Z'_{2,2} &= g^{(m_2 y_2 + a_2 \bar{y}_2) \cdot (\mathsf{pw}' - \mathsf{pw})} \cdot g^{m_2 \bar{y}_2 \cdot (\mathsf{pw}^2 - \mathsf{pw}'^2)} \cdot g^{\bar{x}_2 \bar{y}_2 \cdot (\mathsf{pw}' - \mathsf{pw})}. \end{aligned}$$

Note that $\mathcal{B}_2$ can detect bad_2 with the help of oracles $\textsc{ex2DH}_X$ and $\textsc{ex2DH}_Y$ and trapdoors $m_1, m_2, a_1, a_2, y_1, y_2$. Subsequently, it can extract the solution to the strong ex2DH problem, from tuples $(sid, Z'_{1,1}, Z'_{1,2}, Z'_{2,1}, Z'_{2,2}, \mathsf{pw}')$ and $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$.

Thus we have

$$|\Pr[\textbf{Game 4} \Rightarrow 1] - \Pr[\textbf{Game 3} \Rightarrow 1]| \le \mathsf{Adv}^{\mathsf{st\text{-}ex2DH}}_{\mathbb{G}, \mathcal{B}_2}(\lambda).$$

Game 5. (Randomize keys for server instances in case of incorrect password guesses.) In **Game 5**, Sim further changes the generation of session keys for server instances when detecting an active attack.

Concretely, for a server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ with transcript $X_1^*||X_2^*||Y_1^*||Y_2^*$, let $\mathsf{C}^{(i)}$ be the intended partner and pw be the (correct) password used in this instance. Sim sets $sid := \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$, and generates the session key in the following way.

Case (S.1). If $(\mathsf{S}^{(j)}, iid^{(j)})$ and some good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ are linked to each other, then Sim assigns the same random key for both $(\mathsf{S}^{(j)}, iid^{(j)})$ and $(\mathsf{C}^{(i)}, iid^{(i)})$, just as that in **Game 4**.

Case (S.2). If $(\mathsf{S}^{(j)}, iid^{(j)})$ 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). Sim retrieves the *first* record $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}', \mathsf{key}) \in \mathcal{L}_\mathsf{H}$ s.t. $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw'. If such a record exists and $\mathsf{pw}' = \mathsf{pw}$, then Sim sets the session key to be key. And if $\mathsf{pw}' \neq \mathsf{pw}$, then Sim sets the session key to be random.

Case (S.2.2). If the record does not exist, Sim samples a random key as the session key of $(\mathsf{S}^{(j)}, iid^{(j)})$. Meanwhile, it keeps looking on the *first* hash query $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}')$ s.t. $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw'. If $\mathsf{pw}' = \mathsf{pw}$, then Sim reprograms the random oracle by setting $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}) = \mathsf{key}$. And if $\mathsf{pw}' \neq \mathsf{pw}$, then Sim returns a random key and records it in the hash list $\mathcal{L}_\mathsf{H}$.

Define bad_3 as the event that $\mathcal{A}$ has asked more than one hash query of the form $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}')$ s.t. $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw', and $\mathsf{pw}' \neq \mathsf{pw}$ in the first such query, but $\mathsf{pw}' = \mathsf{pw}$ in some subsequent query.

Obviously **Game 5** and **Game 4** are the same unless bad_3 happens. Next we show that if bad_3 happens, then we can construct a reduction algorithm $\mathcal{B}_3$ to solve the strong ex2DH problem. The reduction is very similar to $\mathcal{B}_2$'s (from **Game 3** to **Game 4**), except that $\mathcal{B}_3$ embeds the problem instance into M_1, M_2, and server's message $Y_1^*||Y_2^*$.

In more detail, $\mathcal{B}_3$ receives the strong ex2DH challenge $(\bar{X}_1, \bar{X}_2, \bar{Y}_1, \bar{Y}_2)$, as well as oracles $\textsc{ex2DH}_X$ and $\textsc{ex2DH}_Y$. Then it simulates **Game 5** as below.

- For the generation of CRS, $\mathcal{B}_3$ randomly samples n_1, n_2, computes $N_1 := g^{n_1}, N_2 := g^{n_2}$, sets $M_1 := \bar{X}_1, M_2 := \bar{X}_2$, and outputs (M_1, M_2, N_1, N_2).
- For the simulation of client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ generating $X_1^*||X_2^*$: $\mathcal{B}_3$ randomly samples $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$, sets $X_1^* := g^{x_1}, X_2^* := g^{x_2}$, adds $(X_1^*||X_2^*, x_1||x_2)$ into $\mathcal{DL}$, and sends $X_1^*||X_2^*$ out.
- For the simulation of server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ generating $Y_1^*||Y_2^*$: $\mathcal{B}_3$ samples $b_1, b_2 \xleftarrow{\$} \mathbb{Z}_q$, sets $Y_1^* := \bar{Y}_1 \cdot g^{b_1} = g^{\bar{y}_1 + b_1}$, $Y_2^* := \bar{Y}_2 g^{b_2} = g^{\bar{y}_2 + b_2}$, and sends $Y_1^*||Y_2^*$ out.
- For the simulation of client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ generating the session key: Let $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ with $X_1^*, X_2^*, Y_1^*, Y_2^*$ the transcripts. There are two cases.

- If $(\mathsf{C}^{(i)}, iid^{(i)})$ and server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ are linked to each other, and $(\mathsf{C}^{(i)}, iid^{(i)})$ is good, then $\mathcal{B}_3$ generates a random key for it if $(\mathsf{S}^{(j)}, iid^{(j)})$ has not computed a session key yet, or assigns the same key of $(\mathsf{S}^{(j)}, iid^{(j)})$ to it, just as **Game 5**.
- Otherwise, $\mathcal{B}_3$ computes the session key with the help of decisional oracle $\textsc{ex2DH}_X$. Concretely, $\mathcal{B}_3$ first computes $X_1 := X_1^*/M_1^{\mathsf{pw}}, X_2 := X_2^*/M_2^{\mathsf{pw}}, Y_1 := Y_1^*/N_1^{\mathsf{pw}}, Y_2 := Y_2^*/N_2^{\mathsf{pw}}$. Then, it uses $\textsc{ex2DH}_X$ to check whether there exists $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}, \mathsf{key}) \in \mathcal{L}_{\mathsf{H}}$, such that $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw. If so, $\mathcal{B}_3$ samples a key and "views" it as the hash output for the correct input $\mathsf{H}(sid, \mathsf{CDH}(X_1, Y_1) =?, \mathsf{CDH}(X_1, Y_2) =?, \mathsf{CDH}(X_2, Y_1) =?, \mathsf{CDH}(X_2, Y_2) =?, \hat{\mathsf{pw}})$, where ? means that the value is to be determined. If $\mathcal{A}$ later asks $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$, then $\mathcal{B}_3$ checks whether $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw via the decisional oracle, i.e., it checks whether

$$\textsc{ex2DH}_X(Y_1, Y_2, (Y_1^{x_1}/Z_{1,1})^{\mathsf{pw}^{-1}}, (Y_2^{x_1}/Z_{1,2})^{\mathsf{pw}^{-1}}, (Y_1^{x_2}/Z_{2,1})^{\mathsf{pw}^{-1}}, (Y_2^{x_2}/Z_{2,2})^{\mathsf{pw}^{-1}}) = 1.$$

  If yes, $\mathcal{B}_3$ reprograms the random oracle by replacing $(?, ?, ?, ?)$ with $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$. In this way, the view of $\mathcal{A}$ is consistent.

– For the simulation of server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ generating the session key: Let $sid = \mathsf{C}^{(i)}||\mathsf{S}^{(j)}||X_1^*||X_2^*||Y_1^*||Y_2^*$ with $X_1^*, X_2^*, Y_1^*, Y_2^*$ the transcripts. There are two cases.

- If $(\mathsf{S}^{(j)}, iid^{(j)})$ and good client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ are linked to each other, then $\mathcal{B}_3$ generates a random key for it if $(\mathsf{C}^{(i)}, iid^{(i)})$ has not computed a session key yet, or assigns the same key of $(\mathsf{C}^{(i)}, iid^{(i)})$ to it, just as **Game 5**.
- Otherwise, $\mathcal{B}_2$ computes $X_1 := X_1^*/M_1^{\mathsf{pw}}, X_2 := X_2^*/M_2^{\mathsf{pw}}, Y_1 := Y_1^*/N_1^{\mathsf{pw}}$, $Y_2 := Y_2^*/N_2^{\mathsf{pw}}$. Then it generates the session key with the help of decisional oracle $\textsc{ex2DH}_Y$ as above, except that the verification formula is changed to

$$\textsc{ex2DH}_Y(X_1, X_2, Z_{1,1}{\cdot}X_1^{n_1 \cdot \mathsf{pw} - b_1}, Z_{1,2}{\cdot}X_1^{n_2 \cdot \mathsf{pw} - b_2}, Z_{2,1}{\cdot}X_2^{n_1 \cdot \mathsf{pw} - b_1}, Z_{2,2}{\cdot}X_2^{n_2 \cdot \mathsf{pw} - b_2}) = 1.$$

– The simulation of H is the same as that in **Game 5**.

If bad_3 happens, then there exists two queries $(sid, Z'_{1,1}, Z'_{1,2}, Z'_{2,1}, Z'_{2,2}, \mathsf{pw}')$ and $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$, such that $Z'_{1,1}, Z'_{1,2}, Z'_{2,1}, Z'_{2,2}$ are correct DH values w.r.t. sid and pw', and $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw.

According to $\mathcal{B}_3$'s simulation above, we get

$$\begin{aligned} Z_{1,1}/Z'_{1,1} &= \frac{\mathsf{CDH}(X_1^*/M_1^{\mathsf{pw}}, Y_1^*/N_1^{\mathsf{pw}})}{\mathsf{CDH}(X_1^*/M_1^{\mathsf{pw}'}, Y_1^*/N_2^{\mathsf{pw}'})} \\ &= \frac{g^{(x_1 - \bar{x}_1 \cdot \mathsf{pw})(\bar{y}_1 + b_1 - n_1 \cdot \mathsf{pw})}}{g^{(x_1 - \bar{x}_1 \cdot \mathsf{pw}')(\bar{y}_1 + b_1 - n_1 \cdot \mathsf{pw}')}} \\ &= g^{(x_1 n_1 + \bar{x}_1 b_1)(\mathsf{pw}' - \mathsf{pw})} \cdot g^{\bar{x}_1 n_1 \cdot (\mathsf{pw}^2 - \mathsf{pw}'^2)} \cdot g^{\bar{x}_1 \bar{y}_1 \cdot (\mathsf{pw}' - \mathsf{pw})}. \end{aligned}$$

Similarly,

$$\begin{aligned}
Z_{1,2}/Z'_{1,2} &= g^{(x_1 n_2 + \bar{x}_1 b_2)(\mathsf{pw}' - \mathsf{pw})} \cdot g^{\bar{x}_1 n_2 \cdot (\mathsf{pw}^2 - \mathsf{pw}'^2)} \cdot g^{\bar{x}_1 \bar{y}_2 \cdot (\mathsf{pw}' - \mathsf{pw})}, \\
Z_{2,1}/Z'_{2,1} &= g^{(x_2 n_1 + \bar{x}_2 b_1)(\mathsf{pw}' - \mathsf{pw})} \cdot g^{\bar{x}_2 n_1 \cdot (\mathsf{pw}^2 - \mathsf{pw}'^2)} \cdot g^{\bar{x}_2 \bar{y}_1 \cdot (\mathsf{pw}' - \mathsf{pw})}, \\
Z_{2,2}/Z'_{2,2} &= g^{(x_2 n_2 + \bar{x}_2 b_2)(\mathsf{pw}' - \mathsf{pw})} \cdot g^{\bar{x}_2 n_2 \cdot (\mathsf{pw}^2 - \mathsf{pw}'^2)} \cdot g^{\bar{x}_2 \bar{y}_2 \cdot (\mathsf{pw}' - \mathsf{pw})}.
\end{aligned}$$

Note that $\mathcal{B}_3$ can detect bad_3 with the help of oracles EX2DH_X and EX2DH_Y and trapdoors $n_1, n_2, b_1, b_2, x_1, x_2$. Subsequently, it can extract the solution to the strong ex2DH problem, from tuples $(sid, Z'_{1,1}, Z'_{1,2}, Z'_{2,1}, Z'_{2,2}, \mathsf{pw}')$ and $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$.

Thus we have

$$|\Pr[\textbf{Game 5} \Rightarrow 1] - \Pr[\textbf{Game 4} \Rightarrow 1]| \leq \mathsf{Adv}^{\mathsf{st\text{-}ex2DH}}_{\mathbb{G},\mathcal{B}_3}(\lambda).$$

Now in **Game 5**, Sim does not use pw any more, except the case Sim checks whether $\mathsf{pw}' = \mathsf{pw}$, when the first hash query happens on $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}')$ s.t. $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t. sid and pw'. Now we are ready to introduce the complete simulator in Fig. 13 and 14, which helps us stepping to the ideal experiment $\textbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$.

Game 6. (Use $\mathcal{F}_{\mathsf{le\text{-}pake}}$ interfaces.) In the final game we introduce the ideal functionality $\mathcal{F}_{\mathsf{le\text{-}pake}}$. By using interfaces to interact with $\mathcal{F}_{\mathsf{le\text{-}pake}}$, the simulator Sim can perfectly simulates **Game 5** as follows.

- The simulation of CRS generation is the same as that in **Game 5**.
- When Sim receives $(\mathsf{NewClient}, \mathsf{C}^{(i)}, iid^{(i)}, \mathsf{S}^{(j)}, b)$ from $\mathcal{F}_{\mathsf{le\text{-}pake}}$, 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 samples $x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q$, sets $X_1^* || X_2^* := g^{x_1} || g^{x_2}$ as the output message, and adds $(X_1^* || X_2^*, x_1 || x_2)$ in $\mathcal{DL}$.
- When Sim receives $(\mathsf{NewServer}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{C}^{(i)})$ from $\mathcal{F}_{\mathsf{le\text{-}pake}}$, it samples $y_1, y_2 \xleftarrow{\$} \mathbb{Z}_q$, sets $Y_1^* || Y_2^* := g^{y_1} || g^{y_2}$ as the output message, and adds $(Y_1^* || Y_2^*, y_1 || y_2)$ in $\mathcal{DL}$.
- When server instance $(\mathsf{S}^{(j)}, iid^{(j)})$ receives $X_1^* || X_2^*$, let $\mathsf{C}^{(i)}$ be its intended partner, $Y_1^* || Y_2^*$ be the message sent out, and pw be the password used in it. Sim sets the session identity to be $sid := \mathsf{C}^{(i)} || \mathsf{S}^{(j)} || X_1^* || X_2^* || Y_1^* || Y_2^*$, 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 $(\mathsf{CopyKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ or a query $(\mathsf{NewKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$, and the choice depends on whether the linked client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ has already been assigned with a random key. According to the definition of NewKey and $\mathsf{CopyKey}$, this performs identically as that in **Game 5**.
    - Otherwise, Sim checks whether there exists a record $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}', \mathsf{key}) \in \mathcal{L}_{\mathsf{H}}$ such that $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values

w.r.t sid and pw'. For ease of retrieval, Sim holds a list $\mathcal{IF}$ to record $(sid, \mathsf{pw}', \mathsf{key})$ and continues detecting such queries in the simulation of H.

  * If the record $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}', \mathsf{key})$ exists, Sim issues a query $(\mathsf{TestPW}, \mathsf{S}^{(j)}, iid^{(j)}, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$. If $\mathcal{F}_{\mathsf{le\text{-}pake}}$ returns "correct guess", Sim sets the session key to be key via a $(\mathsf{CorruptKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid, \mathsf{key})$ query to $\mathcal{F}_{\mathsf{le\text{-}pake}}$. And if $\mathcal{F}_{\mathsf{le\text{-}pake}}$ returns "wrong guess", then Sim allocates sid and a random key by asking a query $(\mathsf{NewKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$. According to the definition of $\mathsf{CorruptKey}$ and NewKey, the environment $\mathcal{Z}$ has the same view as that in **Game 5**.
  * If the record does not exist, Sim marks $(\mathsf{S}^{(j)}, iid^{(j)})$ as "actively attacked" via a query $(\mathsf{RegisterTest}, \mathsf{S}^{(j)}, iid^{(j)})$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$, and allocates a random key to it via query $(\mathsf{NewKey}, \mathsf{S}^{(j)}, iid^{(j)}, sid)$. Meanwhile, if Sim later detects a hash query $(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}')$ such that $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t sid and pw', Sim checks whether $\mathsf{pw}' = \mathsf{pw}$ via a query $(\mathsf{LateTestPwd}, \mathsf{S}^{(j)}, sid, \mathsf{pw}')$ to $\mathcal{F}_{\mathsf{le\text{-}pake}}$. If $\mathsf{pw}' = \mathsf{pw}$ then $\mathcal{F}_{\mathsf{le\text{-}pake}}$ will return the assigned random session key, so that Sim can reprogram H by setting $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}') = \mathsf{key}$. And if $\mathsf{pw}' = \mathsf{pw}$ then $\mathcal{F}_{\mathsf{le\text{-}pake}}$ will return a random key, and Sim just takes it as the result of H, which performs the same as before since H is modelled as a random oracle. Therefore, the view is identical as that in **Game 5**.

- The simulation of key generation for client instance $(\mathsf{C}^{(i)}, iid^{(i)})$ is similar as above due to the symmetry of 2DH-SPAKE2 protocol. We safely omit the description here and refer Fig. 13 and 14 for details.
- The simulation of H is the same as that in **Game 5**. Specifically, Sim maintains a list $\mathcal{IF}$ to record $(sid, \mathsf{pw}', \mathsf{key})$, indicating possible password guesses and corresponding hash values in actively attacked instances. Meanwhile, it continues detecting queries of the form $\mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw}')$ such that $Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}$ are correct DH values w.r.t sid and pw', with the help of trapdoors m_1, m_2, n_1, n_2.

The full description of Sim is shown in Fig. 13 and 14. From the analysis above we know **Game 5** and **Game 6** are conceptually identical. Furthermore, one can easily see that **Game 6** is just the experiment in the ideal world. Therefore, we have

$$\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}} = \textbf{Game 6} = \textbf{Game 5}.$$

Theorem 5 follows immediately from **Game 0** to **Game 6**, and Theorem 6.

D Some Other Protocols

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

Fig. 15. The EKE Protocol [10, 9].

Public Parameter: $(\mathbb{G}, g, q), M, N, \mathsf{H}$		
Client C (pw)		Server S (pw)
$x \overset{\$}{\leftarrow} \mathbb{Z}_q, X := g^x$ $X^* := X \cdot M^{\mathsf{pw}}$	$\xrightarrow{X^*}$ $\xleftarrow{Y^*}$	$y \overset{\$}{\leftarrow} \mathbb{Z}_q, Y := g^y$ $Y^* := Y \cdot N^{\mathsf{pw}}$
$sid := \mathsf{C}\|\|\mathsf{S}\|\|X^*\|\|Y^*$ Output $\mathsf{key}_C \leftarrow \mathsf{H}(sid, (Y^*/N^{\mathsf{pw}})^x, \mathsf{pw})$		$sid := \mathsf{C}\|\|\mathsf{S}\|\|X^*\|\|Y^*$ Output $\mathsf{key}_S \leftarrow \mathsf{H}(sid, (X^*/M^{\mathsf{pw}})^y, \mathsf{pw})$

Fig. 16. The SPAKE2 Protocol [4].

Public Parameter: $(\mathbb{G}, g, q), \mathsf{H}_X, \mathsf{H}_Y$		
Client C (pw)		Server S (pw)
$x \overset{\$}{\leftarrow} \mathbb{Z}_q, X := g^x$ $H_X \leftarrow \mathsf{H}_X(\mathsf{C}, \mathsf{S}, \mathsf{pw})$ $X^* := X \cdot H_X$	$\xrightarrow{X^*}$ $\xleftarrow{Y^*}$	$y \overset{\$}{\leftarrow} \mathbb{Z}_q, Y := g^y$ $H_Y \leftarrow \mathsf{H}_Y(\mathsf{C}, \mathsf{S}, \mathsf{pw})$ $Y^* := Y \cdot H_Y$
$H_Y \leftarrow \mathsf{H}_Y(\mathsf{C}, \mathsf{S}, \mathsf{pw})$ $sid := \mathsf{C}\|\|\mathsf{S}\|\|X^*\|\|Y^*$ Output $\mathsf{key}_C \leftarrow \mathsf{H}(sid, (Y^*/H_Y)^x, \mathsf{pw})$		$H_X \leftarrow \mathsf{H}_X(\mathsf{C}, \mathsf{S}, \mathsf{pw})$ $sid := \mathsf{C}\|\|\mathsf{S}\|\|X^*\|\|Y^*$ Output $\mathsf{key}_S \leftarrow \mathsf{H}(sid, (X^*/H_X)^y, \mathsf{pw})$

Fig. 17. The PPK Protocol [41].

Public Parameter: $(\mathbb{G}, g, q), \mathsf{H}_X, \mathsf{H}_Y$

Client C (pw)		Server S (pw)
$x_1, x_2 \xleftarrow{\$} \mathbb{Z}_q, X_1 := g^{x_1}, X_2 := g^{x_2}$	$\xrightarrow{X_1^*, X_2^*}$	$y \xleftarrow{\$} \mathbb{Z}_q, Y := g^y$
$H_{X,1} \| H_{X,2} \leftarrow \mathsf{H}_X(\mathsf{C}, \mathsf{S}, \mathsf{pw})$		$H_{Y,1} \| H_{Y,2} \leftarrow \mathsf{H}_Y(\mathsf{C}, \mathsf{S}, \mathsf{pw})$
$X_1^* := X_1 \cdot H_{X,1}, X_2^* := X_2 \cdot H_{X,2}$	$\xleftarrow{Y_1^*, Y_2^*}$	$Y_1^* := Y_1 \cdot H_{Y,1}, Y_2^* := Y_2 \cdot H_{Y,2}$
$H_{Y,1} \| H_{Y,2} \leftarrow \mathsf{H}_Y(\mathsf{C}, \mathsf{S}, \mathsf{pw})$		$H_{X,1} \| H_{X,2} \leftarrow \mathsf{H}_X(\mathsf{C}, \mathsf{S}, \mathsf{pw})$
$Z_{1,1} := (\frac{Y_1^*}{H_{Y,1}})^{x_1}, Z_{1,2} := (\frac{Y_2^*}{H_{Y,2}})^{x_1}$		$Z_{1,1} := (\frac{X_1^*}{H_{X,1}})^{y_1}, Z_{1,2} := (\frac{X_1^*}{H_{X,1}})^{y_2}$
$Z_{2,1} := (\frac{Y_1^*}{H_{Y,1}})^{x_2}, Z_{2,2} := (\frac{Y_2^*}{H_{Y,2}})^{x_2}$		$Z_{2,1} := (\frac{X_2^*}{H_{X,2}})^{y_1}, Z_{2,2} := (\frac{X_2^*}{H_{X,2}})^{y_2}$
$sid := \mathsf{C} \| \mathsf{S} \| X_1^* \| X_2^* \| Y_1^* \| Y_2^*$		$sid := \mathsf{C} \| \mathsf{S} \| X_1^* \| X_2^* \| Y_1^* \| Y_2^*$
Output $\mathsf{key}_C \leftarrow \mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$		Output $\mathsf{key}_S \leftarrow \mathsf{H}(sid, Z_{1,1}, Z_{1,2}, Z_{2,1}, Z_{2,2}, \mathsf{pw})$

Fig. 18. The 2DH-PPK Protocol.

Table of Contents

EKE Meets Tight Security in the Universally Composable Framework . . . 1
Xiangyu Liu, Shengli Liu, Shuai Han, and Dawu Gu
1 Introduction . . . 1
1.1 Technical Overview . . . 4
1.2 Roadmap . . . 7
2 Preliminaries . . . 8
2.1 Hardness Assumptions . . . 8
3 PAKE with Tight Security in the UC Framework . . . 8
3.1 UC Framework for PAKE . . . 8
3.2 The 2DH-EKE Protocol . . . 12
3.3 Security Analysis . . . 13
4 Asymmetric PAKE with Optimal Tightness in the UC Framework . . . 23
4.1 UC Framework for aPAKE . . . 23
4.2 The 2DH-aEKE Protocol . . . 25
4.3 Security Analysis . . . 25
5 Optimal Reduction Loss in aPAKE . . . 42
6 Tight Security for 2DH-SPAKE2 in the Relaxed UC Framework . . . 46
A Ideal Ciphers and Random Oracles . . . 51
A.1 Ideal Cipher . . . 51
A.2 Random Oracle . . . 51
B Relaxed UC Framework for PAKE . . . 52
C Proof of Theorem 5 . . . 52
D Some Other Protocols . . . 66