Hybrid Password Authentication Key Exchange in the UC Framework

You Lyu[1,2] and Shengli Liu[1,2](✉)

[1] Department of Computer Science and Engineering
Shanghai Jiao Tong University, Shanghai 200240, China
[2] State Key Laboratory of Cryptology, P.O. Box 5159, Beijing 100878, China
{vergil,slliu}@sjtu.edu.cn

Abstract. A hybrid cryptosystem combines two systems that fulfill the same cryptographic functionality, and its security enjoys the security of the harder one. There are many proposals for hybrid public-key encryption (hybrid PKE), hybrid signature (hybrid SIG) and hybrid authenticated key exchange (hybrid AKE). In this paper, we fill the blank of Hybrid Password Authentication Key Exchange (hybrid PAKE).

For constructing hybrid PAKE, we first define an important class of PAKE – *full* DH-type PAKE, from which we abstract sufficient properties to achieve UC security. Our *full* DH-type PAKE framework unifies lots of PAKE schemes like SPAKE2, TBPEKE, (Crs)X-GA-PAKE, and summarizes their common features for UC security.

Stepping from full DH-type PAKE, we propose two generic approaches to hybrid PAKE, *parallel composition* and *serial composition*.

- We propose a generic construction of hybrid PAKE via *parallel composition* and prove that the hybrid PAKE by composing DH-type PAKEs *in parallel* is a full DH-type PAKE and hence achieves UC security, as long as one underlying DH-type PAKE is a full DH-type.
- We propose a generic construction of hybrid PAKE via *serial composition*, and prove that the hybrid PAKE by composing a DH-type PAKE and another PAKE *in serial* achieves UC security, if *either* the DH-type PAKE is a full DH-type *or* the other PAKE has UC security and the DH-type PAKE only has some statistical properties.

Our generic constructions of hybrid PAKE result in a variety of hybrid PAKE schemes enjoying different nice features, like round-optimal, high efficiency, or UC security in quantum random oracle model (QROM).

1 Introduction

Password-based cryptosystems are widely deployed in our life, providing authentication and session keys to facilitate secure communications. The task of establishing session keys between two parties in the password only setting is accomplished by Password Authenticated Key Exchange (PAKE) protocols. Since its introduction by Bellovin and Meritt [14], many efforts have been devoted to the design and security of PAKE [13,18,7,28,24,34,3,11].

UC Security of PAKE. There are two primary security notions for PAKE, the game-based security in the Indistinguishability model (IND security) [13] and the simulation-based security under the Universally Composable framework (UC security) [18]. In contrast to the IND model which assumes uniformly distributed passwords, the UC framework permits arbitrary correlations and distributions for passwords and guarantees security amidst composition with arbitrary protocols. Moreover, UC security, even in a relaxed form, can imply IND security as shown in [3]. In practice, PAKE usually serves as a sub-protocol to provide session keys for upper-level protocols, so it always works with other sub-protocols to support flexible cryptographic functionalities. Thanks to UC security of sub-protocols, the whole protocol composed by these sub-protocols still enjoys security in the UC framework. Therefore, UC security is a preferred security notion for PAKE and received lots of concerns.

UC-Secure PAKE from Traditional Assumptions. Numerous efficient UC-secure PAKE protocols have been proposed from traditional number-theoretic assumptions, especially over cyclic groups. Examples include SPEKE [26], SPAKE2 [7], KC-SPAKE2 [34], TBPEKE [32], and CPace [24,5], etc. These protocols follow Diffie-Hellman style and can be viewed as password-aided Diffie-Hellman key exchange protocol. Notably, most of these protocols are round-optimal [26,7,32,5], i.e., they can be executed in a single simultaneous round, mirroring the round efficiency of Diffie-Hellman protocols. Due to its efficiency and UC security, CPace has been chosen as one of the recommended PAKE protocols by IETF [6].

UC-Secure PAKE from Post-Quantum Assumptions. Up to now, there are two approaches to UC-secure PAKE from post-quantum assumptions. One approach resorts to Encrypted Key Exchange (EKE)[14], which takes passwords as the symmetric key and encrypts the transcripts generated by KEM algorithms or None Interactive Key Exchange (NIKE). The UC security is proved by modeling the symmetric encryption/decryption as Ideal Cipher, and hence based on the Ideal Cipher Model (ICM). In fact, the two-round PAKE schemes [33,12,9] and the round-optimal PAKE scheme [11] from lattices all follow the EKE approach. However, the EKE approach does not apply to isogenies since there are no instantiations of ideal cipher applying to supersingular elliptic curves yet.

Another approach [30] combines lossy public key encryption (LPKE) and key encapsulation mechanism (KEM), and the resulting PAKE achieves UC security in the Random Oracle model (ROM). If the underlying LPKE is upgraded to extractable LPKE, then the UC security of the PAKE scheme can be proved in Quantum Random Oracle model (QROM). This approach yields three-round PAKE schemes from lattices and three-round PAKE schemes from isogenies.

The EKE approach is more efficient, but the LPKE approach is more general and covers both lattice and isogeny-based instantiations.

We note that the PAKE scheme from lattices in [11] is round-optimal. There do exist round-optimal PAKE schemes [4,25] from isogenies, but their securities were only proved in IND security model, rather than in UC framework. Hence round-optimal PAKE with UC security from isogenies is still unknown, which is highlighted in [27]. Naturally, we have the following question:

Q1: Can we find a round-optimal PAKE achieving UC security from isogenies?

Hybrid PAKE from Traditional and Post-Quantum Ones. The standardization efforts by NIST are pushing industries to migrate traditional cryptosystems to post-quantum ones. However, the current quantum-resistant hard problems are young and not as well-understood as the traditional number-theoretic hard problems, like factoring, RSA, CDH, DDH etc. Moreover, an imperfect deployment of post-quantum algorithms may compromise the security of the system. So we need more choices for quantum-resistant hard problems and corresponding algorithms to improve robustness. On the other hand, the practical realization of quantum computers remains distant due to challenges like qubit decoherence resulting from the current building materials. So the traditional cryptosystems may retain their vitality for a considerable time. Based on the above observations, many national agencies, like the German BSI [2] and the French ANSSI [1], called for hybrid cryptosystems which combine both traditional and post-quantum ones: the security of hybrid cryptosystems relies on *either* the traditional one *or* the post-quantum one. Meanwhile, hybrid cryptosystems can also work in other ways: it combines two or several traditional ones, or it combines two or several post-quantum ones. The resulting hybrid scheme enjoys the security of the harder of the two or more, improving security robustness.

There have already been lots of works on this topic. Hybrid KEM schemes are proposed in [23,10], hybrid digital signature schemes in [17,15], and hybrid authenticated key exchange (AKE) in [21]. However, up to now hybrid PAKE is still missing. In fact, Katz and Rosenberg [27] pointed out: "there is no known hybrid PAKE (in UC framework), let alone a generic method for creating one" and listed the following question as future work:

Q2: Can we find a generic approach to Hybrid PAKE to achieve UC security?

It is not easy to solve *Q2*. For example, we cannot use EKE-based PAKE schemes [33,12,9] from lattices directly in the hybrid PAKE. The reason is as follows. Suppose that the LWE assumption does not hold anymore. Then the encrypted transcript $c = \mathsf{IC.Enc}(pw, \mathbf{As} + \mathbf{e})$ in the EKE-based PAKE can help the adversary implement offline dictionary attacks: $\mathcal{A}$ tries every possible choice of password to recover $\mathbf{v} = \mathsf{IC.Dec}(pw, c)$, and $\mathcal{A}$ checks whether $(\mathbf{A}, \mathbf{v})$ is an LWE tuple. If yes, then $\mathcal{A}$ obtains the correct password pw. If this EKE-PAKE is employed by a hybrid PAKE, the hybrid PAKE has already lost its security even if its other PAKE components are UC-secure. This example suggests that hybrid PAKE needs careful and delicate designs.

Our Contribution. We answer the above questions with a full DH-type PAKE Framework and two generic constructions of hybrid PAKE (HPAKE), namely parallel composition and serial composition. See Fig. 1.

(1) Full DH-Type PAKE Framework. We define *full DH-type PAKE* with sufficient properties, which help prove that any full DH-type PAKE has UC security. We unify many existing PAKE schemes in our full DH-type PAKE framework, including SPAKE2 [7], TBPEKE [32], and (Crs)X-GA-PAKE

[4,25]. Our framework captures the underlying principles of these schemes and explains why they can achieve UC security.

(2) Hybrid PAKE via Parallel Composition. We show how to compose two (or more) PAKEs in parallel to obtain HPAKE. We prove that parallel composition of two (or many) DH-type PAKE yields a full DH-type HPAKE as long as one of the underlying PAKE components is full DH-type. According to (1), the resulting full DH-type HPAKE also achieves UC security which can rely on the security of a single underlying PAKE component.

(3) Hybrid PAKE via Serial Composition. We show how to compose two PAKEs in serial to obtain Hybrid PAKE. We prove that the Hybid PAKE achieves UC security, if either the first PAKE is a full DH-type one or the second PAKE has UC security (and the DH-type only has some statistical properties). Moreover, if the second PAKE has UC security in QROM, then the resulting Hybid PAKE also enjoys UC security in QROM.

Our result (1) suggests that the (Crs)X-GA-PAKE schemes [4,25] are round-optimal UC-secure PAKEs from isogenies, and hence solves the aforementioned question *Q1*. Our results (2) and (3) solve the aforementioned question *Q2*.

According to Fig. 1, our result implies abundant hybrid PAKE schemes. Below we list a few.

- **Round-optimal Hybrid PAKE.** Parallel composition of SPAKE2[7] and (Crs)X-GA-PAKE[4,25] or parallel composition of SPAKE2, (Crs)X-GA-PAKE and TBPEKE[32] yields a round-optimal hybrid PAKE scheme relying on either the traditional assumptions or post-quantum assumption from isogenies in ROM.
- **Efficient Hybrid PAKE.** Serial composition of SPAKE2 and the recent EKE-based PAKE CHIC[9] yields an efficient hybrid PAKE scheme relying on either the traditional assumptions or post-quantum assumption from lattices in ROM. Recall that EKE-based PAKEs are the most efficient ones among the existing UC-secure PAKEs from post-quantum assumptions. This hybrid PAKE is efficient since it inherits efficiency from SPAKE2 and CHIC.
- **Hybrid PAKE in QROM.** Serial composition of SPAKE2 and $\mathsf{PAKE}^{\mathsf{QRO}}$[30] yields a hybrid PAKE scheme whose UC-security is achieved in QROM as long as $\mathsf{PAKE}^{\mathsf{QRO}}$ has UC security in QROM.

Technique Overview. UC security considers the indistinguishability between the real world and the ideal world for an environment $\mathcal{Z}$. In the real world, $\mathcal{Z}$ initiates passwords for the parties, sees the interactions between parties and the adversary $\mathcal{A}$, and obtains the session key output by parties. In the ideal world, the simulator Sim simulates the view of $\mathcal{Z}$ without pw by accessing an ideal functionality $\mathcal{F}$ for PAKE. For UC security, we have to design PAKE and a corresponding simulator Sim to accomplish the following tasks:

Task I. Sim can simulate all the interaction transcripts for both passive attacks and active attacks without passwords.

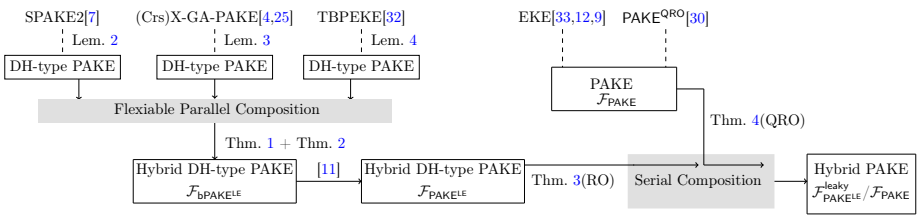

Fig. 1: Schematic overview of constructing hybrid PAKEs via parallel composition and serial composition. Here "*flexible* parallel composition" means that the DH-type PAKE components can be flexibly chosen: one, two or many. "$\_\_\_\_$" denotes instantiations. $\boxed{\text{A}} \rightarrow \text{B} \rightarrow \boxed{\text{C}}$ denotes that A is used for C via B.

Task II. PAKE has pseudo-random session keys in case of passive attacks and unsuccessful active attacks (i.e., attacks with wrong passwords), so Sim can sample random keys for simulation, which is done by invoking $\mathcal{F}$.

Task III. Sim can extract password in case of active attacks, and keep the consistency of session keys between $\mathcal{Z}$ and $\mathcal{A}$ for successful active attacks (i.e., attacks with correct passwords) with the help of access to Testpw (one access per protocol instance) provided by $\mathcal{F}$.

Next we give a high-level description of our full DH-type PAKE framework and explain why it can achieve UC security. Then we show how to combine DH-type PAKEs via parallel composition and how to combine a DH-type PAKE with another PAKE via serial composition to construct hybrid PAKEs.

FULL DH-TYPE PAKE FRAMEWORK. The starting point is a DH-type PAKE (see Fig. 6). The two parties can invoke algorithms $\mathsf{DHInit}(pw)$ and $\mathsf{DHResp}(pw)$ to generate round messages m_1 and m_2 (and their state st_1 and st_2), and compute their DH-key w by $\mathsf{Comp_i}(pw, st_1, m_1, m_2)$ and $\mathsf{Comp_r}(pw, st_2, m_1, m_2)$, respectively. The session key is a hash value $\mathsf{Key} = H(pw, m_1, m_2, w)$. To support UC security, we equip DH-type PAKE with simulation algorithms $\mathsf{SimInit}$, $\mathsf{SimResp}$, $\mathsf{SimComp_i}$, and $\mathsf{SimComp_r}$ satisfying the following properties to help construct a simulator Sim. Here we require $\mathsf{SimComp_i}$ and $\mathsf{SimComp_r}$ are deterministic algorithms.

Perfect simulation of round messages. $\mathsf{SimInit}$ and $\mathsf{SimResp}$ should simulate messages m_1 and m_2 perfectly without any password, and hence Sim can invoke them to accomplish task I.

One-wayness of the DH-Key in case of passive attacks. It is hard for $\mathcal{A}$ to compute w given (m_1, m_2), so session key $\mathsf{Key} = H(pw, m_1, m_2, w)$ is uniform in the real world due to the one-wayness of w and random oracle H. So Sim can sample a random key to simulate Key. This accomplishes task II.

Unique password extraction in case of active attacks. To support this, we have the following two requirements.

- **Perfect simulation of DH-key.** For $\mathcal{A}$'s active attack with $\tilde{m}_1$, the DH-key $w := \mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$, has identical distribution as the output of $\mathsf{Comp_i}$. A similar requirement applies to $\mathsf{SimComp_r}$ as well.

– **Unique password for initiator/responder.** It is hard for $\mathcal{A}$ to present two pairs (pw, w) and (pw', w') with $pw \neq pw'$ such that $w = \mathsf{SimComp_i}(td, pw, st_1, \tilde{m}_1, m_2)$ and $w' = \mathsf{SimComp_i}(td, pw', st_1, \tilde{m}_1, m_2)$. A similar requirement applies to $\mathsf{SimComp_r}$ as well.

For an active attack, Sim can extract at most one pair (pw, w) s.t. $w = \mathsf{SimComp_i}(td, pw, st_1, \tilde{m}_1, m_2)$ (or $w = \mathsf{SimComp_r}(td, pw, st_2, m_1, \tilde{m}_2)$) from $\mathcal{A}$'s hash queries $H(pw, m_1, m_2, w)$. $\mathsf{Testpw}(pw)$ helps Sim to determine whether pw is correct or not. Therefore, in case of unsuccessful active attacks (pw is not correct), $\mathcal{A}$ did not query $H(pw^*, m_1, m_2, w)$ with the correct password pw^* due to the unique password property, so the session key $\mathsf{sKey} := H(pw^*, m_1, m_2, w)$ is uniform in the real world, which accomplishes task II. For successful active attacks, Sim can extract the correct password and honestly set the session key as $\mathsf{sKey} := H(pw^*, m_1, m_2, w)$ to keep consistency, which accomplishes task III.

We call the DH-type PAKE satisfying the above properties a *full* DH-type PAKE. The above analysis shows that full DH-type PAKE has UC security[3] and enjoys the round-optimal property[4]. Actually, SPAKE2 [7], TBPEKE [32], and (Crs)X-GA-PAKE [4,25] all fall into our full DH-type PAKE framework.

The full DH-type PAKE does not end with UC security. In fact, the good property of "*perfect simulation of round messages and DH-keys*" implies that the round messages contain no information about the password at all, and the DH-key in the real world can be simulated perfectly in case of successful active attacks. Therefore, DH-type PAKEs with this property can serve as vital components in constructing hybrid PAKE no matter via parallel or serial compositions.

Hybrid PAKE via Parallel Composition. Given two PAKE schemes PAKE_1 and PAKE_2, hybrid PAKE via parallel composition means that two parties $\mathcal{P}_i$ and $\mathcal{P}_j$ run PAKE_1 and PAKE_2 with password pw in parallel to obtain their session keys k_1 and k_2 respectively, and compute the session key $\mathsf{sKey} := \mathrm{Hash}(pw, k_1, k_2, trans)$. See the left part of Fig. 2.

If PAKE_1 and PAKE_2 are both DH-type PAKE, then the hybrid PAKE sets its DH-key as $w := (w^{(1)}, w^{(2)})$ and its session key as $\mathsf{sKey} := H(pw, w^{(1)}, w^{(2)}, trans)$, where $w^{(1)}$(resp. $w^{(2)}$) is the DH-key of PAKE_1(resp. PAKE_2).

If one PAKE scheme, say PAKE_1, is a full DH-type PAKE, and the other PAKE scheme, say PAKE_2, is DH-type PAKE with the aforementioned good property, then we show that the hybrid PAKE via parallel composition is a full DH-type PAKE. The simulation algorithms of hybrid PAKE are parallel invocations of the underlying simulation algorithms of PAKE_1 and PAKE_2. For example, $\mathsf{SimComp_i} := (\mathsf{SimComp_i}^{(1)}, \mathsf{SimComp_i}^{(2)})$, where $\mathsf{SimComp_i}^{(1)}$ belongs to PAKE_1 and $\mathsf{SimComp_i}^{(2)}$ belongs to PAKE_2.

[3] Note that the adversary can make the hash query after session key generation. Accordingly, Sim extracts the password after session key generation. This UC security is reflected by lazy-extraction of the ideal functionality $\mathcal{F}_{\mathsf{PAKE^{LE}}}$ (see Fig. 4).

[4] Due to the independence of the two round messages, full DH-type PAKE is actually a bare PAKE [11], which can emulate $\mathcal{F}_{\mathsf{bPAKE^{LE}}}$, better than $\mathcal{F}_{\mathsf{PAKE^{LE}}}$. See section 3.2 for details.

- The good property of perfect simulation of round messages and DH-key for hybrid PAKE inherits from PAKE_1 and PAKE_2.
- The one-wayness of $w = (w^{(1)}, w^{(2)})$ follows from that of $w^{(1)}$.
- PAKE_1's property of unique password for initiator implies there exists at most one $(pw, w^{(1)})$ s.t. $w^{(1)} = \mathsf{SimComp_i}^{(1)}(td^{(1)}, pw, st_1^{(1)}, m_1^{(1)}, m_2^{(1)})$. Meanwhile, the unique pw determines the unique $w^{(2)} := \mathsf{SimComp_i}^{(2)}(td^{(2)}, pw, st_1^{(2)}, m_1^{(2)}, m_2^{(2)})$. Consequently, there exists at most one $(pw, w = (w^{(1)}, w^{(2)}))$ satisfying the above two equations, and hybrid PAKE also has the property of unique password for initiator. A similar argument applies to the responder.

Therefore, the hybrid PAKE via parallel composition falls into the full DH-type PAKE framework and hence achieves UC security as long as one component is a full DH-type one and the other has the good property of "perfect simulation of round messages and DH-keys". This result can be extended to parallel composition of *multiple* DH-type PAKEs.

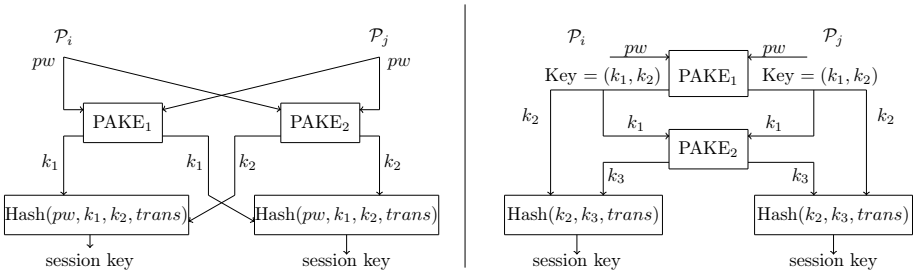

Fig. 2: Parallel composition (left) & serial composition (right) for hybrid PAKE.

Hybrid PAKE via Serial Composition. Given two PAKE schemes PAKE_1 and PAKE_2, we propose how to construct hybrid PAKE via serial composition. Two parties $\mathcal{P}_i$ and $\mathcal{P}_j$ first run PAKE_1 with password pw to share PAKE_1's session key $\mathsf{Key} = (k_1, k_2)$, then run PAKE_2 with password k_1 to share PAKE_2's session key k_3. Finally they set the hybrid PAKE's session key as $\mathsf{sKey} := \mathrm{Hash}(k_2, k_3, trans)$. See the right part of Fig. 2.

To show UC security of this hybrid PAKE, we will justify the following two statements.

1. If PAKE_1 is a full DH-type PAKE, then we show that the hybrid PAKE via serial composition has UC security.
2. If PAKE_2 scheme has UC security in (Q)ROM and PAKE_1 is a DH-type PAKE with the good property of "perfect simulation of round messages and DH-keys", then we show that the hybrid PAKE via serial composition has UC security in (Q)ROM.

If PAKE_1 scheme is a full DH-type PAKE, then it is UC-secure and associated with a simulator Sim_1. Next we prove the validity of statement 1.

- Thanks to the UC security of PAKE_1, Sim_1 can help Sim simulate the PAKE_1's transcript and its session key $\mathsf{Key} = (k_1, k_2)$. Then Sim can invoke PAKE_2 with password k_1 to generate PAKE_2's transcript and its session key k_3. Lastly, Sim simulates hybrid PAKE's session key with $\mathsf{sKey} := \mathrm{Hash}(k_2, k_3, trans)$. Clearly, the transcript simulation is perfect, and hence task I is accomplished.
- In case of passive attacks or unsuccessful active attacks (with wrong pw), $\mathsf{Key} = (k_1, k_2)$ of PAKE_1 is uniform according to UC security of PAKE_1. Consequently, $\mathsf{sKey} = \mathrm{Hash}(k_2, k_3, trans)$ is uniform as well due to the uniformity of k_2, and hence task II is accomplished.
- In case of successful active attacks (with correct pw), Sim_1 can help Sim to extract the correct password, and keep the consistency of $\mathsf{Key} = (k_1, k_2)$, thanks to PAKE_1's UC security again. Since k_3 is derived from PAKE_2 with password k_1, the input $(k_2, k_3, trans)$ are consistent, so the session key $\mathsf{sKey} = \mathrm{Hash}(k_2, k_3, trans)$ keeps consistency to $\mathcal{Z}$ and $\mathcal{A}$, and hence task III is accomplished. Note that Sim_1 can help Sim to extract pw in case of active attacks, so Sim can tell the correctness of pw with the help of $\mathsf{Testpw}(pw)$.

For statement 2, we suppose that PAKE_2 has UC security in (Q)ROM and is associated with a simulator Sim_2, and we also suppose that PAKE_1 is a DH-type PAKE with the good property of "perfect simulation of round messages and DH-keys" with the four simulation algorithms $\mathsf{SimInit}$, $\mathsf{SimResp}$, $\mathsf{SimComp_i}$, $\mathsf{SimComp_r}$. Now we construct simulator Sim for hybrid PAKE as follows.

– Sim invokes $\mathsf{SimInit}$ and $\mathsf{SimResp}$ to generate transcript of PAKE_1, and invokes Sim_2 to generate transcript and session key k_3 of PAKE_2. Due to the good property of PAKE_1 and the UC security of PAKE_2, Sim gives an indistinguishable transcript of hybrid PAKE, and hence task I is accomplished.

– In case of passive attacks, the UC security of PAKE_2 guarantees the uniformity of k_3, which further implies the uniformity of $\mathsf{sKey} = \mathrm{Hash}(k_2, k_3, trans)$, and hence task II for passive attacks is accomplished.

– Upon an active attack, due to the UC security of PAKE_2, Sim_2 can extract the unique password guessing k_1 of PAKE_2. Then Sim searches the hash list to find (pw, m_1, m_2, w) s.t. $(k_1, k_2) = H(pw, m_1, m_2, w)$, where (m_1, m_2) is the transcript of PAKE_1 and w is the DH-key of PAKE_1 (i.e., $w = \mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$ or $w = \mathsf{SimComp_r}(td, pw, st_2, m_1, m_2)$). In this way, Sim extracts a unique guessing password pw from $\mathcal{A}$'s attacks. Then Sim resorts to $\mathsf{Testpw}(pw)$ to check the correctness of the password.
  - In case of unsuccessful active attacks (pw is not correct), adversary $\mathcal{A}$ guesses a wrong password, which leads to a wrong k_1 for PAKE_2. This implies an unsuccessful active attack for PAKE_2. In this case, the UC security of PAKE_2 guarantees the uniformity of k_3, which further implies the uniformity of $\mathsf{sKey} = \mathrm{Hash}(k_2, k_3, trans)$. Therefore, task II for unsuccessful active attacks is accomplished.
  - In case of successful active attacks (pw is correct and hence $\mathsf{Key} = (k_1, k_2)$ can be correctly derived from pw by Sim), Sim_2 can keep the consistency

of PAKE_2's session key k_3 due to PAKE_2's UC security. Consequently, Sim can set session key as $\mathsf{sKey} := \mathrm{Hash}(k_2, k_3, trans)$ to keep consistency between $\mathcal{A}$ and $\mathcal{Z}$, and hence task III is accomplished.

Note that Sim needs to extract password pw from k_1, and this extraction is realized by formalizing H as a random oracle. By resorting to the online-extractable technique [22], the extraction can also be realized in QROM where H is formalized as a quantum random oracle. If PAKE_2 has UC security in QROM, then the resulting hybrid PAKE has UC security in QROM as well.

We only give a high-level description of our generic construction of hybrid PAKE and their analysis for UC security. In fact, there are many subtleties to deal with in the formal constructions and security proofs. We refer to sections 3 and 4 for details.

2 Preliminary

If x is defined by y or y is assigned to x, we write $x := y$. For $\mu \in \mathbb{N}$, define $[\mu] := \{1, 2, \ldots, \mu\}$. Denote by $x \leftarrow_{\$} \mathcal{X}$ the procedure of sampling x from set $\mathcal{X}$ uniformly at random. All algorithms $y \leftarrow \mathcal{A}(x)$ are probabilistic take the security parameter 1^λ as implicit input unless stated otherwise. $X \equiv Y$ denotes that X and Y are independent identically distributed. $[a \stackrel{?}{=} b]$ checks whether $a = b$ or not, and it returns 1 if $a = b$ and returns 0 otherwise. Retrieving records $([a], b, [c], d)$ means records are retrieved according to b and d. ROM and QROM are discussed in Appendix A.1.

2.1 Password-Based Authenticated Key Exchange

A (two-round) PAKE scheme consists of four algorithms:

- Setup : It outputs a public parameter pp. All the remaining algorithms take pp as input, and we omit it for simplicity.
- $\mathsf{Init}(pw)$: It takes as input a password pw, and outputs a first-round message m_1 and a round state st.
- $\mathsf{Resp}(pw, m_1)$: It takes as input a password pw and the first-round message m_1, and outputs a second-round message m_2 and a session key Key.
- $\mathsf{Deri}(pw, st, m_2)$: It takes as input a password pw, a round state st and the second-round message m_2, and outputs a session key Key.

2.2 The Universal Composable Framework

We present a concise overview of the UC framework. Fig. 3 shows the picture of the "real world" execution of a protocol Π and the "ideal world" execution with a simulator Sim. The environment $\mathcal{Z}$ represents higher-level protocols invoking Π as a sub-protocol. The adversary $\mathcal{A}$ models a completely insecure network and

continuously interacts with $\mathcal{Z}$. Each party instance sends and receives messages via $\mathcal{A}$, and $\mathcal{A}$ can do dropping, injecting, and modifying messages at will.

In the real world, parties execute protocol instances as described in Π. They receive their inputs (e.g. passwords, session identifier, role) from $\mathcal{Z}$ and send their outputs (e.g. session key, sub-session identifier) to $\mathcal{Z}$.

In the ideal world, parties become "dummy" entities, passing inputs directly from $\mathcal{Z}$ to an ideal functionality $\mathcal{F}$, and outputs directly from $\mathcal{F}$ to $\mathcal{Z}$. The simulator Sim interacts with $\mathcal{Z}$ and $\mathcal{F}$. Sim must indistinguishably simulate network transcripts as in the real world and provide appropriate inputs to $\mathcal{F}$ to generate outputs indistinguishable from those produced by real-world instances.

In a nutshell, we say protocol Π securely emulates ideal functionality $\mathcal{F}$ if for any efficient adversary $\mathcal{A}$, there exists an efficient simulator Sim such that no efficient environment $\mathcal{Z}$ can distinguish between the actual protocol executions in the real world and the hypothetical executions simulated in the ideal world.

If protocol Π is constructed from some subprotocol Π', which securely emulates an ideal functionality $\mathcal{G}$, then the description of Π and its UC security may be proven in the $\mathcal{G}$-hybrid model. In the $\mathcal{G}$-hybrid model, Π' is replaced by $\mathcal{G}$, and parties and the adversary interact with $\mathcal{G}$ according to its specification. As for the UC security, Sim must additionally simulate the ideal functionality of $\mathcal{G}$.

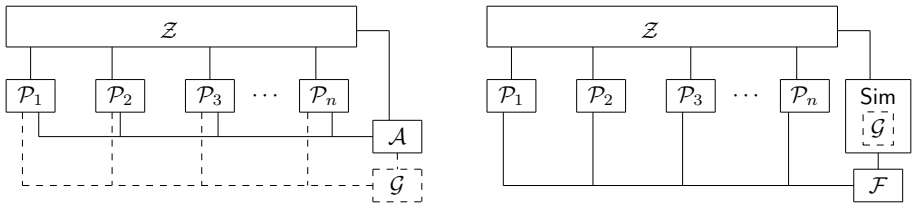

Fig. 3: Left: the real world $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$ (without dashed lines and dashbox), and the real world $\mathbf{Real}^{\mathcal{G}}_{\mathcal{Z},\mathcal{A}}$ in $\mathcal{G}$-hybrid model (with dashed lines and dashbox). Right: the ideal world $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$ (without dashbox) and the ideal world $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}[\mathcal{G}]}$ in $\mathcal{G}$-hybrid model (with dashbox).

2.3 PAKE in UC Framework

The original ideal functionality for PAKE was presented in [18]. Later it was extended to a multi-session version $\mathcal{F}_{\mathsf{PAKE}}$ in [19] and a lazy-extraction version $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ in [3]. The two extended versions are shown in Fig. 4.

Unlike the single-session PAKE functionality in [18], the multi-session extension $\mathcal{F}_{\mathsf{PAKE}}$ mandates that each individual protocol instance possesses a globally unique sub-session identifier *ssid*. The session identifier *sid* becomes a global parameter which all parties in the protocol can rely on. See Fig. 4.

The lazy-extraction version $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ was proposed by Abdalla et al. [3], and it permits adversary $\mathcal{A}$ to guess a party's password *pw* (at most once) even

after the session key is generated. If the guess is correct, $\mathcal{A}$ acquires the party's session key. Otherwise, $\mathcal{A}$ obtains only a random key. In comparison, in the original definition of $\mathcal{F}_{\mathsf{PAKE}}$ in [18], $\mathcal{F}_{\mathsf{PAKE}}$ does not reply $\mathcal{A}$ at all when $\mathcal{A}$ issues a late password guess after the session key generation. Notably, in the definition of $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ in [3], the adversary is not explicitly informed whether its password guess is correct. In this paper we also introduce a leaky variant of $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ that notifies the adversary of the password guess's outcome. UC security with $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ and its leaky variant may be not as strong as that with $\mathcal{F}_{\mathsf{PAKE}}$, but leaking bits of success or failure and late test after session key generation do reflect the features of the real world. Hence ideal functionality of $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ is a reasonable one and adopted in many works [3,5,29,11]. See Fig. 4.

Session Initialization

Upon receiving a query (NewSession, sid, $\mathcal{P}$, $ssid$, $\mathcal{CP}$, pw, role) **from** $\mathcal{P}$:
Send (NewSession, sid, $\mathcal{P}$, $ssid$, $\mathcal{CP}$, role) to $\mathcal{A}$
If this is the first NewSession query on $ssid$,
or this is the second NewSession query on $ssid$ and record $(\mathcal{CP}, ssid, \mathcal{P}, pw')$ exists:
Record $(\mathcal{P}, ssid, \mathcal{CP}, pw)$ and mark it **fresh**

Active Session Attacks

Upon receiving a query (Testpw, $sid, \mathcal{P}, ssid, pw'$) **from** $\mathcal{A}$:
If there is a **fresh** record $(\mathcal{P}, ssid, [\mathcal{CP}, pw])$:
If $pw' = pw$: mark the record **compromised** and reply with "correct guess"
If $pw' \neq pw$: mark the record **interrupted** and reply with "wrong guess"

Lazy Password Extraction

Upon receiving a query (RegisterTest, $sid, \mathcal{P}, ssid$) **from** $\mathcal{A}$:
If there is a **fresh** record $(\mathcal{P}, ssid, [\mathcal{CP}, pw])$:
mark the record **interrupted** and add a "**tested**" flag.

Upon receiving a query (LateTestPwd, $sid, \mathcal{P}, ssid, pw'$) **from** $\mathcal{A}$:
If there is a **completed** record $(\mathcal{P}, ssid, [\mathcal{CP}, pw])$ with flag **tested**:
retrieve the record $(\mathcal{P}, ssid, \mathcal{CP}, pw, \mathsf{Key})$, remove the flag **tested**
If $pw = pw'$: return Key and "correct guess" to $\mathcal{A}$
Else: sample a random $\mathsf{Key}' \leftarrow_{\$} \mathcal{K}$, return Key' and "wrong guess" to $\mathcal{A}$

Key Generation

Upon receiving a query (NewKey, $sid, \mathcal{P}, ssid, \mathsf{Key}^*$) **from** $\mathcal{A}$:
If there is a record $(\mathcal{P}, ssid, [\mathcal{CP}, pw])$ not marked **completed**:
If the record is **compromised**: set $\mathsf{Key} := \mathsf{Key}^*$
If the record is **fresh**, and there exists a **completed** record $(\mathcal{CP}, ssid, \mathcal{P}, pw)$ which was **fresh** when $\mathcal{CP}$ output $(sid, ssid, \mathsf{Key}')$: set $\mathsf{Key} := \mathsf{Key}'$
In all other cases, sample a random Key
Finally, mark the record $(\mathcal{P}, ssid, \mathcal{CP}, pw)$ **completed** and store $(\mathcal{P}, ssid, \mathcal{CP}, pw, \mathsf{Key})$

Fig. 4: The ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$ for PAKE. Adding the gray part to but deleting the part from $\mathcal{F}_{\mathsf{PAKE}}$ defines a lazy extraction version called $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$. Adding both gray and part to $\mathcal{F}_{\mathsf{PAKE}}$ defines a leaky version named $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$.

Very recently, Barbosa et al. proposed the so-called *Bare PAKE functionality* $\mathcal{F}_{\mathsf{bPAKE}}$ and its lazy-extraction variant $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ in [11]. We recall $\mathcal{F}_{\mathsf{bPAKE}}$ and

$\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ in Fig. 5. $\mathcal{F}_{\mathsf{bPAKE}}$ (and $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$) mainly offers two advantages over the original PAKE functionality $\mathcal{F}_{\mathsf{PAKE}}$.

- $\mathcal{F}_{\mathsf{bPAKE}}$ eliminates the need for pre-shared sub-session identifiers $ssid$ and the unique peer identifiers cid indicating counter party $\mathcal{CP}$. Each party's input to $\mathcal{F}_{\mathsf{bPAKE}}$ includes the global parameter sid, a locally unique instance identifier i, the party's identifier id, and a password pw. The sub-session identifier and peer identifier are generated as outputs of $\mathcal{F}_{\mathsf{bPAKE}}$. This simplification enhances practicality and reduces implementation hurdles.
- $\mathcal{F}_{\mathsf{bPAKE}}$ enables parties to reuse their session states. An instance can utilize its existing session state to communicate with multiple other instances. $\mathcal{F}_{\mathsf{bPAKE}}$ guarantees that reusing a single instance's state is equivalent to running multiple independent PAKE instances with the same password.

In $\mathcal{F}_{\mathsf{bPAKE}}$ (see Fig. 5), $\mathsf{PassiveNewKey}$ captures $\mathcal{A}$'s passive attacks of matching instances and corresponds to the NewKey query to $\mathcal{F}_{\mathsf{PAKE}}$ with a passive attack. $\mathsf{ActiveNewKey}$ captures $\mathcal{A}$'s active attacks and corresponds to the combination of Testpw and NewKey queries to $\mathcal{F}_{\mathsf{PAKE}}$. For example, to attack an instance $(sid, \mathcal{P}, i)$, $\mathcal{A}$ can query $\mathcal{F}_{\mathsf{bPAKE}}$ with $(\mathsf{ActiveNewKey}, sid, \mathcal{P}, i, pw', \mathsf{Key}', ssid, cpid)$. If $pw' = pw$, then $\mathcal{F}_{\mathsf{bPAKE}}$ sets the session key as Key' and sets the peer identifier of instance $(sid, \mathcal{P}, i)$ as cid. Otherwise, choose $\mathsf{Key} \leftarrow_{\$} \mathcal{K}$ as the session key.

3 Full DH-Type PAKE: Definition, UC security, and Parallel Composition

We introduce the concept of full DH-type PAKE in Subsec. 3.1, and then we prove that any full DH-type PAKE scheme achieves the bare PAKE functionality $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ in the UC framework in Subsec. 3.2. Finally, we show how to construct hybrid DH-type PAKE schemes by combining multiple DH-type PAKE protocols in parallel in Subsec. 3.3.

3.1 Definition of Full DH-Type PAKE

A Diffie-Hellman type (DH-type) PAKE protocol is a two-round PAKE and captured by five algorithms: $\mathsf{DHSetup}$ sets up the system; An initiator uses $\mathsf{DHInit}(pw)$ to generate the first-round message; A responder uses $\mathsf{DHResp}(pw)$ to generate the second-round message; The initiator uses $\mathsf{Comp_i}$ to generate the DH-key w; The responder uses $\mathsf{Comp_r}$ to generate the DH-key w; Finally, both the initiator and responder take the password pw, the DH-key w, and the transcript of the protocol as the input of a hash function to derive the session key Key from the hash function. The DH-type is featured by the fact that the generation of second-round message does not rely on the first-round message (and hence DH-type PAKE can be executed in a single simultaneous round). See Fig. 6 for details.

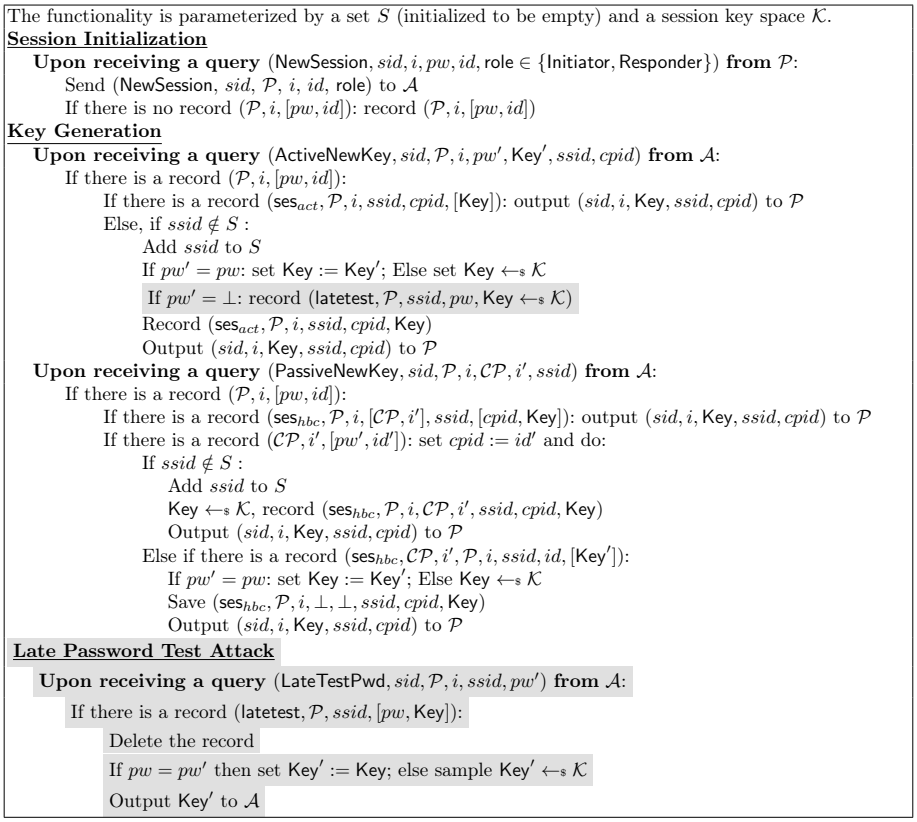

Fig. 5: The ideal functionality $\mathcal{F}_{\mathsf{bPAKE}}$ for bare PAKE. Adding gray part to $\mathcal{F}_{\mathsf{bPAKE}}$ defines a lazy extraction version of PAKE called $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$.

Definition 1. *A DH-type PAKE protocol is a two-round protocol with the following five algorithms.*

- $\mathsf{DHSetup}$*: It outputs a public parameter* pp *and a trapdoor* td*. All the remaining algorithms take* pp *as input, and we omit it for simplicity.*
- $\mathsf{DHInit}(pw)$*: It takes as input a password* pw*, and outputs a message* m_1 *and a secret state* st_1*.*
- $\mathsf{DHResp}(pw)$*: It takes as input a password* pw*, and outputs a message* m_2 *and a secret state* st_2*.*
- $\mathsf{Comp_i}(pw, st_1, m_1, m_2)$*: It is a deterministic algorithm, which takes as input a password* pw*, a secret state* st_1*, and two messages* m_1*,* m_2*, and outputs a DH-key* w*.*
- $\mathsf{Comp_r}(pw, st_2, m_1, m_2)$*: It is a deterministic algorithm, which takes as input a password* pw*, a secret state* st_2*, and two messages* m_1*,* m_2*, and outputs a DH-key* w*.*

Correctness. *For all* pw *and* $(\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}$*, it holds that*

$$\Pr\left[\begin{array}{c}(m_1, st_1) \leftarrow \mathsf{DHInit}(pw) \\ (m_2, st_2) \leftarrow \mathsf{DHResp}(pw)\end{array} : \mathsf{Comp_i}(pw, st_1, m_1, m_2) = \mathsf{Comp_r}(pw, st_2, m_1, m_2)\right] = 1.$$

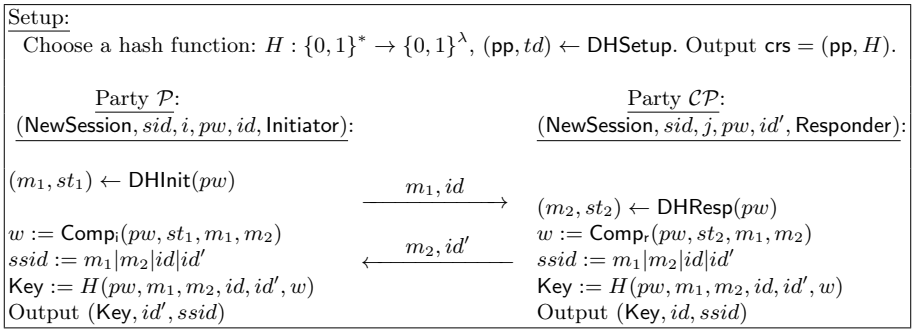

Fig. 6: DH-type PAKE protocol.

Definition 2 (Full DH-type PAKE). *A two-round PAKE protocol is a full DH-type PAKE, if it is a DH-type one* $(\mathsf{DHSetup}, \mathsf{DHInit}, \mathsf{DHResp}, \mathsf{Comp_i}, \mathsf{Comp_r})$ *associated with simulation algorithms* $(\mathsf{SimInit}, \mathsf{SimResp}, \mathsf{SimComp_i}, \mathsf{SimComp_r})$ *and satisfies five properties. The simulation algorithms are defined as follows.*

- $\mathsf{SimInit}(\mathsf{pp})$: *It takes as input a public parameter* pp*, and outputs a message* m_1 *and a secret state* st_1*.*
- $\mathsf{SimResp}(\mathsf{pp})$: *It takes as input a public parameter* pp*, and outputs a message* m_2 *and a secret state* st_2*.*

- $\mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$: *It is a deterministic algorithm, which takes as input a trapdoor td, a password pw, a state st_1, and two messages m_1, m_2 and outputs a key w.*
- $\mathsf{SimComp_r}(td, pw, st_2, m_1, m_2)$: *It is a deterministic algorithm, which takes as input a trapdoor td, a password pw, a state st_2, and two messages m_1, m_2 and outputs a key w.*

There are five properties defined for full DH-type PAKE protocols.

① **One-Wayness of the DH-Key:** The advantage function $\mathsf{Adv}^{\mathsf{ow}}_{\mathsf{key}}(\mathcal{A})$ is negligible for all PPT adversary $\mathcal{A}$ access to oracles $\mathcal{O}_1(m_1^*, st_1^*, \cdot, \cdot, \cdot)$ and $\mathcal{O}_2(m_2^*, st_2^*, \cdot, \cdot, \cdot)$, where $\mathsf{Adv}^{\mathsf{ow}}_{\mathsf{key}}(\mathcal{A}) :=$

$$\Pr\left[\begin{array}{c} (\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}, pw \leftarrow \mathcal{A}(\mathsf{pp}) \\ (m_1^*, st_1^*) \leftarrow \mathsf{DHInit}(pw) \\ (m_2^*, st_2^*) \leftarrow \mathsf{DHResp}(pw) \\ w \leftarrow \mathcal{A}^{\mathcal{O}_1(m_1^*, st_1^*, \cdot, \cdot, \cdot), \mathcal{O}_2(m_2^*, st_2^*, \cdot, \cdot, \cdot)}(\mathsf{pp}, pw, m_1^*, m_2^*) \end{array} : w = \mathsf{Comp_i}(pw, st_1^*, m_1^*, m_2^*)\right],$$

$\mathcal{O}_1(m_1^*, st_1^*, pw', m_2', w') := [w' \stackrel{?}{=} \mathsf{Comp_i}(pw', st_1^*, m_1^*, m_2')]$ and $\mathcal{O}_2(m_2^*, st_2^*, pw', m_1', w') := [w' \stackrel{?}{=} \mathsf{Comp_r}(pw', st_2^*, m_1', m_2^*)]$.

② **Perfect Simulation for Initiator:** For all $(\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}$, all pw, and all m_2, it holds that

$$(m_1, m_2, w) \equiv (m_1', m_2, w'),$$

where $(m_1, st_1) \leftarrow \mathsf{DHInit}(pw)$, $w := \mathsf{Comp_i}(pw, st_1, m_1, m_2)$, $(m_1', st_1') \leftarrow \mathsf{SimInit}(\mathsf{pp})$, $w' := \mathsf{SimComp_i}(td, pw, st_1', m_1', m_2)$.

③ **Perfect Simulation for Responder:** For all $(\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}$, all pw and all m_1, it holds that

$$(m_1, m_2, w) \equiv (m_1, m_2', w'),$$

where $(m_2, st_2) \leftarrow \mathsf{DHResp}(pw)$, $w := \mathsf{Comp_r}(pw, st_2, m_1, m_2)$, $(m_2', st_2') \leftarrow \mathsf{SimResp}(\mathsf{pp})$, and $w' := \mathsf{SimComp_r}(td, pw, st_2', m_1, m_2')$.

④ **Unique Password for Initiator:** The advantage function $\mathsf{Adv}^{\mathsf{Init}}_{\mathsf{Unipw}}(\mathcal{A})$ is negligible for all PPT adversary $\mathcal{A}$ with oracle access to $\mathcal{O}_1(td^*, \cdots), \mathcal{O}_2(td^*, \cdots)$ and $\mathcal{O}_1^*(td^*, st_1^*, m_1^*, \cdots)$, where $\mathsf{Adv}^{\mathsf{Init}}_{\mathsf{Unipw}}(\mathcal{A}) :=$

$$\Pr\left[\begin{array}{c} (\mathsf{pp}^*, td^*) \leftarrow \mathsf{DHSetup} \\ (m_1^*, st_1^*) \leftarrow \mathsf{SimInit}(\mathsf{pp}^*) \\ (pw, pw', m_2, w, w') \leftarrow \mathcal{A}^{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_1^*}(\mathsf{pp}^*, m_1^*) \end{array} : \begin{array}{c} pw \neq pw' \\ w = \mathsf{SimComp_i}(td^*, pw, st_1^*, m_1^*, m_2) \\ w' = \mathsf{SimComp_i}(td^*, pw', st_1^*, m_1^*, m_2) \end{array}\right],$$

$\mathcal{O}_1(td^*, pw, st_1, m_1, m_2, w) := [w \stackrel{?}{=} \mathsf{SimComp_i}(td^*, pw, st_1, m_1, m_2))]$, $\mathcal{O}_2(td^*, pw, st_2, m_1, m_2, w) := [w \stackrel{?}{=} \mathsf{SimComp_r}(td^*, pw, st_2, m_1, m_2)]$ and $\mathcal{O}_1^*(td^*, st_1^*, m_1^*, pw, m_2, w) := [w \stackrel{?}{=} \mathsf{SimComp_i}(td^*, pw, st_1^*, m_1^*, m_2)]$.

⑤ **Unique Password for Responder:** The following advantage function is negligible for any PPT adversary $\mathcal{A}$ with oracle access to $\mathcal{O}_1(td^*, \cdots)$,

$\mathcal{O}_2(td^*, \cdots)$, and $\mathcal{O}_2^*(td^*, st_2^*, m_2^*, \cdots)$, where $\mathsf{Adv}^{\mathsf{Resp}}_{\mathsf{Unipw}}(\mathcal{A}) :=$

$$\Pr\left[\begin{array}{c} (\mathsf{pp}^*, td^*) \leftarrow \mathsf{DHSetup} \\ (m_1, st_1) \leftarrow \mathcal{A}^{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_2^*}(\mathsf{pp}^*), \\ (m_2^*, st_2^*) \leftarrow \mathsf{SimResp}(\mathsf{pp}^*) \\ (pw, pw', w, w') \leftarrow \mathcal{A}^{\mathcal{O}_1, \mathcal{O}_2, \mathcal{O}_2^*}(st_1, \mathsf{pp}^*, m_1) \end{array} : \begin{array}{c} pw \neq pw' \\ w = \mathsf{SimComp_r}(td^*, pw, st_2^*, m_1, m_2^*) \\ w' = \mathsf{SimComp_r}(td^*, pw', st_2^*, m_1, m_2^*) \end{array}\right],$$

$\mathcal{O}_1(td^*, pw, st_1, m_1, m_2, w) := [w \stackrel{?}{=} \mathsf{SimComp_i}(td^*, pw, st_1, m_1, m_2)]$, $\mathcal{O}_2(td^*, pw, st_2, m_1, m_2, w) := [w \stackrel{?}{=} \mathsf{SimComp_r}(td^*, pw, st_2, m_1, m_2)]$, and $\mathcal{O}_2^*(td^*, st_2^*, m_2^*, pw, m_1, w) := [w \stackrel{?}{=} \mathsf{SimComp_r}(td^*, pw, st_2^*, m_1, m_2^*)]$.

Note that $A \equiv B$ in ② and ③ can be relaxed to statistical closeness between A and B.

3.2 UC Security of Full DH-Type PAKE

We will prove that any full DH-type PAKE has UC security in Theorem 1.

Theorem 1. *Any full DH-type PAKE per Def. 2, can securely emulate $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ and hence achieve UC security in the random oracle model.*

According to Corollary 1 in [11], one can easily transform a PAKE protocol that securely emulates $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ into a PAKE protocol that securely emulates $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ by setting $pw' := \mathcal{P}|\mathcal{CP}|sid|ssid|pw$, where pw is the password used in $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ and pw' is used in $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$. By applying this result to Theorem 1, we have the following corollary.

Corollary 1. *Any full DH-type PAKE can securely emulate $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ when the underlying password is replaced by $pw' := \mathcal{P}|\mathcal{CP}|sid|ssid|pw$.*

Moreover, according to [3], by adding mutual explicit authentication to the PAKE protocol, its ideal functionality $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ can be further strengthened to a slightly relaxed $\mathcal{F}_{\mathsf{PAKE}}$[5], and any PAKE emulating relaxed $\mathcal{F}_{\mathsf{PAKE}}$ implies perfect forward security in the IND model.

Proof of Theorem 1. Our goal is to construct Sim and show that $|\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}} \Rightarrow 1]|$ is negligible by employing a sequence of games, denoted as Game G_0-G_8. In this sequence, G_0 corresponds to $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$, while G_8 corresponds to $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$. We aim to show that these adjacent games are indistinguishable from the view of $\mathcal{Z}$.

Game G_0(real world). This is the real experiment $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}$. In this experiment, $\mathcal{Z}$ initializes a password for each party, 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 $\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}} \Rightarrow 1] = \Pr[\mathsf{G}_0 \Rightarrow 1]$.

[5] In relaxed $\mathcal{F}_{\mathsf{PAKE}}$, the adversary can try a late password test on an actively attacked session (at most once), but such sessions are guaranteed to terminate with an abort.

Game G_1(simulations for parties with pw). In this game, we introduce a simulator Sim who *additionally knows passwords* of all parties. Sim generates $\mathsf{crs} := (\mathsf{pp}, H)$ with $(\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}$ and keeps td. Then it simulates the parties to generate transcripts for instances of the PAKE protocol, just like G_0. With the knowledge of *passwords*, the simulations of the behaviors of all parties are perfect.

Moreover, Sim also simulates the random oracle H by maintaining a list $\mathcal{L}_H$. For a query x on $H(\cdot)$, if $(x, y) \in \mathcal{L}_H$, then Sim will return y as the reply. Otherwise, Sim will choose a random element y, record (x, y) in $\mathcal{L}_H$, and return y as the reply. By the ideal functionality of random oracles, Sim's simulation for H is also perfect. So we have $\Pr[\mathsf{G}_1 \Rightarrow 1] = \Pr[\mathsf{G}_0 \Rightarrow 1]$.

Game G_2(simulations of Key without pw in case of passive attacks). G_2 is the same as G_1, except for Sim's simulation of generating Key in case of passive attacks.

- When Sim simulates the session key $\mathsf{Key} := H(pw, m_1, m_2, id, id', w)$, it checks the followings. If both m_1 and m_2 are generated by Sim via $(m_1, st_1) \leftarrow \mathsf{DHInit}(pw)$ for instance $(\mathcal{P}, i)$ and $(m_2, st_2) \leftarrow \mathsf{DHResp}(pw, m_1)$ for instance $(\mathcal{CP}, j)$ and $w = \mathsf{Comp_i}(pw, st_1, m_1, m_2)$, then Sim samples $\mathsf{Key} \leftarrow_\$ \mathcal{K}$ rather than $\mathsf{Key} := H(pw, m_1, m_2, id, id', w)$. If $(\mathcal{P}, i)$ has identity id and $(\mathcal{CP}, j)$ has identity id', then Sim sets the random Key as the session key for both $(\mathcal{P}, i)$ and $(\mathcal{CP}, j)$.

Accordingly, in G_2, the generation of session keys does not need pw anymore in case of passive attacks.

G_2 is the same as G_1, except there exists a pair of instance $(\mathcal{P}, i)$ and $(\mathcal{CP}, j)$ s.t. event $\mathsf{Bad}_{i,j}$ happens.

$\mathsf{Bad}_{i,j}$**:** $\mathcal{A}$ has issued hash query $(pw, m_1, m_2, id, id', w)$ on oracle H satisfying $w = \mathsf{Comp_i}(pw, st_1, m_1, m_2)$, where $(m_1, st_1) \leftarrow \mathsf{DHInit}(pw)$ is generated by Sim for $(\mathcal{P}, i)$, and $(m_2, st_2) \leftarrow \mathsf{DHResp}(pw)$ is generated by Sim for $(\mathcal{CP}, j)$.

Define event $\mathsf{Bad} := \bigvee_{i,j} \mathsf{Bad}_{i,j}$. We have

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \Pr[\mathsf{Bad}] \leq \ell^2 \cdot \Pr[\mathsf{Bad}_{i,j}],$$

where ℓ is the maximal number of protocol instances. Next we show $\Pr[\mathsf{Bad}_{i,j}] \leq \mathsf{Adv}^{\mathsf{ow}}_{\mathsf{key}}(\mathcal{B}_{ow})$ with a security reduction: If Bad_{ij} happens, we construct algorithm $\mathcal{B}_{ow}$ breaking "① one-wayness of the DH-key" of the DH-type PAKE.

$\mathcal{B}_{ow}$ has public parameter pp, two messages m_1^*, m_2^*, and two oracles $\mathcal{O}_1(m_1^*, st_1^*, \cdots)$, $\mathcal{O}_2(m_2^*, st_2^*, \cdots)$. $\mathcal{B}_{ow}$ sets $\mathsf{crs} := (\mathsf{pp}, H)$, (m_1^*, id) and (m_2^*, id') as the output messages of $(\mathcal{P}, i)$ and $(\mathcal{CP}, j)$ respectively.

- If an initiator instance $(\mathcal{P}, i)$ with password pw receives a message $(m_2', \hat{id})$, $\mathcal{B}_{ow}$ will search in the hash query list $\mathcal{L}_H$ to see whether there exists $(pw, m_1^*, m_2', id, \hat{id}, [w, k]) \in \mathcal{L}_H$ s.t. $\mathcal{O}(m_1^*, st_1^*, pw, m_2', w) = 1$. If yes, then $\mathcal{B}_{ow}$ sets $\mathsf{Key} := k$. Otherwise, $\mathcal{B}_{ow}$ randomly samples $\mathsf{Key} \leftarrow_\$ \mathcal{K}$ and will reprogram

$H(pw, m_1^*, m_2', id, \hat{id}, w) := \mathsf{Key}$ if $\mathcal{A}$ later issues hash query $(pw, m_1^*, m_2', id, \hat{id}, w)$ such that $\mathcal{O}(m_1^*, st_1^*, pw, m_2', w) = 1$.

– If an responder instance $(\mathcal{CP}, j)$ receives a message (m_1', id), $\mathcal{B}_{ow}$ uses a similar strategy like initiator instance$(\mathcal{P}, i)$ to generate Key with help oracle $\mathcal{O}_2$ and pw. $\mathcal{B}_{ow}$ always keeps the initiator and the responder sharing the same session key in case of passive attacks.

$\mathcal{B}_{ow}$ can simulate round messages and session keys for other instances perfectly with help of passwords. It is easy to see $\mathcal{B}_{ow}$ simulates G_2 perfectly for $\mathcal{Z}$. Finally, $\mathcal{B}_{ow}$ can search in $\mathcal{L}_H$ to find a hash query record $(pw, m_1^*, m_2^*, [id, id', w, k])$ such that $\mathcal{O}(m_1^*, st_1^*, pw, m_2^*, w) = 1$. If such a record exists, then $\mathsf{Bad}_{i,j}$ happens since $\mathcal{O}(m_1^*, st_1^*, pw, m_2^*, w) = 1$ is equivalent to $w = \mathsf{Comp_i}(pw, st_1^*, m_1^*, m_2^*)$. Then $\mathcal{B}_{ow}$ submits w as its own answer. Clearly $\mathcal{B}_{ow}$ wins if $\mathsf{Bad}_{i,j}$ happens. So $\Pr[\mathsf{Bad}_{i,j}] \leq \mathsf{Adv}^{\mathsf{ow}}_{\mathsf{key}}(\mathcal{B}_{ow})$, and we have $|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \ell^2 \cdot \mathsf{Adv}^{\mathsf{ow}}_{\mathsf{key}}(\mathcal{B}_{ow})$.

Game G_3(simulation of m_1 without pw). G_3 is the same as G_2, except for Sim's simulation of generating m_1 and w.

– Sim generates m_1 by $(m_1, st_1) \leftarrow \mathsf{SimInit}(\mathsf{pp})$ rather than $(m_1, st_1) \leftarrow \mathsf{DHInit}(pw)$. Correspondingly, Sim computes $w := \mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$ rather than $w := \mathsf{Comp_i}(pw, st_1, m_1, m_2)$ in case of active attacks (recall that in case of passive attacks, Key is randomly chosen due to G_2', so pw or w are not needed at all).

By the "② Perfect Simulation for Initiator" property of the full DH-type PAKE protocol, G_3 is the same as G_2. So we have $\Pr[\mathsf{G}_3 \Rightarrow 1] = \Pr[\mathsf{G}_2 \Rightarrow 1]$.

Game G_4(simulation of m_2 without pw). G_4 is the same as G_3, except for the simulation of generating m_2 and w.

– Sim generates m_2 by $(m_2, st_2) \leftarrow \mathsf{SimResp}(\mathsf{pp})$ rather than $(m_2, st_2) \leftarrow \mathsf{DHResp}(pw)$. Correspondingly, Sim computes $w := \mathsf{SimComp_r}(td_2, pw, st_2, m_1, m_2)$ rather than $w := \mathsf{Comp_r}(pw, st_2, m_1, m_2)$ in case of active attacks (recall that in case of passive attacks, Key is randomly chosen due to G_2, so pw or w are not needed at all).

By the "③ Perfect Simulation for Responder" property of the DH-type PAKE protocol, G_4 is the same as G_3. So we have $\Pr[\mathsf{G}_4 \Rightarrow 1] = \Pr[\mathsf{G}_3 \Rightarrow 1]$.

Up to now, Sim does not use pw to generate round messages m_1, m_2 no matter in case of passive attacks or active attacks. Meanwhile, it does not use pw to generate DH-keys w in case of passive attacks, but still uses pw to compute DH-keys w in case of active attacks.

Game G_5(abort if there exist multiple password guesses for initiator). G_5 is the same as G_4, except Sim will abort if the following event BadI happens.

BadI: There exist an instance $(\mathcal{P}, i)$ and two hash queries $([pw], m_1, m_2, [id, id', w, k]) \in \mathcal{L}_H$ and $([pw'], m_1, m_2, [id, id', w', k']) \in \mathcal{L}_H$ s.t. $pw \neq pw'$, $w = \mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$ and $w' = \mathsf{SimComp_i}(td, pw', st_1, m_1, m_2)$, where m_1 is generated by Sim invoking $(m_1, st_1) \leftarrow \mathsf{SimInit}(\mathsf{pp})$ for $(\mathcal{P}, i)$.

We further define event BadI_j as the event that BadI happens for the j-th protocol instance. Then $\mathsf{BadI} = \bigvee_{j=1}^{\ell} \mathsf{BadI}_j$. If BadI does not happen, then G_5 is identical to G_4. So $|\Pr[\mathsf{G}_5 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \Pr[\mathsf{BadI}] \leq \sum_{j=1}^{\ell} \Pr[\mathsf{BadI}_j]$.

Now we show if BadI_j happens, then we can construct a reduction algorithm $\mathcal{B}_{init}$ breaking the "④ Unique Password for Initiator" property of the DH-type PAKE protocol.

$\mathcal{B}_{init}$ obtains pp^* and m_1^* from its own challenger, and has access three oracles $\mathcal{O}_1(td^*, \cdots)$, $\mathcal{O}_2(td^*, \cdots)$ and $\mathcal{O}_1^*(td^*, st_1^*, m_1^*, \cdots)$, where pp^* is generated by $\mathsf{DHSetup}$ and m_1^* is generated by $(m_1^*, st_1^*) \leftarrow \mathsf{SimInit}(\mathsf{pp}^*)$. The goal of $\mathcal{B}_{init}$ is to find two valid password guesses for initiator, i.e., to find (pw, pw', m_2, w, w') s.t. $w = \mathsf{SimComp_i}(td^*, pw, st_1^*, m_1^*, m_2)$ and $w' = \mathsf{SimComp_i}\ (td^*, pw', st_1^*, m_1^*, m_2)$ but $pw \neq pw'$. To this end, $\mathcal{B}_{init}$ simulates G_5 to $\mathcal{Z}$ as follows.

$\mathcal{B}_{init}$ first sets $\mathsf{crs} := (\mathsf{pp}^*, H)$. For all but j protocol instances, $\mathcal{B}_{init}$ can simply generate m_1 by $(m_1, st_1) \leftarrow \mathsf{SimInit}(\mathsf{pp})$ for initiators and m_2 by $(m_2, st_2) \leftarrow \mathsf{SimResp}(\mathsf{pp})$ for responders, just like Sim does in G_5. It can also simulate the generation of session key Key with the help of passwords and oracles $\mathcal{O}_1(td^*, \cdots)$, $\mathcal{O}_2(td^*, \cdots)$. Let us take a responder instance $(\mathcal{CP}, i)$ as an example. Let pw be the password used by $(\mathcal{CP}, i)$. Upon receiving a message (m_1, id), $\mathcal{B}_{init}$ simulates Key for $(\mathcal{CP}, i)$ as follow. It first invokes $(m_2, st_2) \leftarrow \mathsf{SimResp}(\mathsf{pp})$ and outputs (m_2, id'). If there exists $(pw, m_1, m_2, [id, id', w, k]) \in \mathcal{L}_H$ s.t. $\mathcal{O}_2(td^*, pw, st_2, m_1, m_2, w) = 1$, then set $\mathsf{Key} := k$. Otherwise, it randomly samples $\mathsf{Key} \leftarrow_{\$} \mathcal{K}$. If later $\mathcal{A}$ issues a hash query $(pw, m_1, m_2, id, id', w)$ such that $\mathcal{O}_2(td^*, pw, st_2, m_1, m_2, w) = 1$, then $\mathcal{B}_{init}$ reprograms $H(pw, m_1, m_2, id, id', w) := \mathsf{Key}$ and stores $(pw, m_1, m_2, id, id', w, \mathsf{Key})$ in $\mathcal{L}_H$. For initiator instance (not the j-th instance), $\mathcal{B}_{init}$ has a similar simulation of Key but with the help of $\mathcal{O}_1(td^*, \cdots)$.

As for the j-th protocol instance, say $(\mathcal{P}^*, i^*)$, if $\mathcal{P}^*$ is not an initiator, then BadI_j does not happen and $\mathcal{B}_{init}$ aborts. Otherwise, $\mathcal{B}_{init}$ sets m_1^* and the identity id of $(\mathcal{P}^*, i^*)$ as the round message output of $(\mathcal{P}^*, i^*)$. Let pw^* be the password used by $(\mathcal{P}^*, i^*)$. Upon receiving (m_2, id'), $\mathcal{B}_{init}$ simulates the generation of session key as follows. If there exists $(pw^*, m_1^*, m_2, id, id', [w, k]) \in \mathcal{L}_H$ s.t. $\mathcal{O}_1^*(td^*, st_1^*, m_1^*, pw^*, m_2, w) = 1$, then set $\mathsf{Key} := k$. Otherwise, it randomly samples $\mathsf{Key} \leftarrow_{\$} \mathcal{K}$. If later $\mathcal{A}$ issues hash query $(pw^*, m_1^*, m_2, id, id', w)$ such that $\mathcal{O}_1^*(td^*, st_2^*, m_1^*, pw^*, m_2, w) = 1$, then $\mathcal{B}_{init}$ reprograms $H(pw^*, m_1^*, m_2, id, id', w)$ $:= \mathsf{Key}$ and stores $(pw^*, m_1^*, m_2, id, id', w, \mathsf{Key})$ in $\mathcal{L}_H$.

Finally, $\mathcal{B}_{init}$ checks whether BadI_j happens: it goes through all records $([pw], m_1^*, [m_2, id, id', w]) \in \mathcal{L}_H$ and resorts to $\mathcal{O}_1^*(td^*, st_1^*, m_1^*, pw, m_2, w)$ to determine $[w \stackrel{?}{=} \mathsf{SimComp_i}(td^*, pw, st_1^*, m_1^*, m_2)]$. If there are two records $(pw, m_1^*, m_2, id, id', w, k)$ and $(pw', m_1^*, m_2, id, id', w', k')$ such that $pw \neq pw'$ and $\mathcal{O}_1^*(td^*, st_1^*, m_1^*, pw, m_2, w) = \mathcal{O}_1^*(td^*, st_1^*, m_1^*, pw', m_2, w') = 1$, then BadI_j happens. In this case, $\mathcal{B}_{init}$ submits (pw, pw', m_2, w, w') as its answer to its challenger.

Note that $\mathcal{B}_{init}$ perfectly simulates G_4 for $\mathcal{Z}$, just like Sim. If BadI_j happens, then $\mathcal{B}_{init}$ breaks the "④ Unique Password for Initiator" property. So we have $\Pr[\mathsf{BadI}_j] \leq \mathsf{Adv}^{\mathsf{Init}}_{\mathsf{Unipw}}(\mathcal{B}_{init})$ and
$|\Pr[\mathsf{G}_5 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \Pr[\mathsf{BadI}] \leq \sum_{j=1}^{\ell} \Pr[\mathsf{BadI}_j] \leq \ell \cdot \mathsf{Adv}^{\mathsf{Init}}_{\mathsf{Unipw}}(\mathcal{B}_{init})$.

Game G_6(abort if there exist multiple password guesses for responder). G_6 is the same as G_5, except Sim will abort if the following event BadR happens.

BadR: There exist an instance $(\mathcal{P}, i)$ and two hash queries $([pw], m_1, m_2, [id, id', w, k]) \in \mathcal{L}_H$ and $([pw'], m_1, m_2, [id, id', w', k']) \in \mathcal{L}_H$ s.t. $pw \neq pw'$, $w = \mathsf{SimComp_r}(td, pw, st_2, m_1, m_2)$ and $w' = \mathsf{SimComp_r}(td, pw', st_2, m_1, m_2)$, where (m_2, id') is generated by Sim invoking $(m_2, st_2) \leftarrow \mathsf{SimResp}(\mathsf{pp})$ for $(\mathcal{P}, i)$.

We further define event BadR_j as the event that BadR happens for the j-th protocol instances. With a similar argument as G_5, we have $|\Pr[\mathsf{G}_6 \Rightarrow 1] - \Pr[\mathsf{G}_5 \Rightarrow 1]| \leq \Pr[\mathsf{BadR}] \leq \sum_{j=1}^{\ell} \Pr[\mathsf{BadR}_j] \leq \ell \cdot \mathsf{Adv}^{\mathsf{Resp}}_{\mathsf{Unipw}}(\mathcal{B}_{resp})$.

Game G_7 (simulations of Key without pw in case of active attacks). G_7 is the same as G_6, except for the simulation of Key in case of active attacks.

- For an initiator instance $(\mathcal{P}, i)$, suppose that (m_1, id) is the message generated by Sim for $(\mathcal{P}, i)$, and (m_2, id') is a message received from $\mathcal{A}$. We assume that (m_2, id') is not generated by Sim, and hence (m_2, id') results from $\mathcal{A}$'s active attack. In this case, Sim searches in the hash list $\mathcal{L}_H$. If there exists $([pw], m_1, m_2, id, id', [w, k]) \in \mathcal{L}_H$ s.t. $w = \mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$ (at most one due to G_6), then Sim checks whether pw is the password used by $(\mathcal{P}, i)$. if yes, then $\mathcal{A}$ must have implemented a successful active attack on $(\mathcal{P}, i)$, so Sim sets $\mathsf{Key} := k$. If there exists no such record (i.e., $\mathcal{A}$'s active attack fails), Sim randomly samples $\mathsf{Key} \leftarrow_{\$} \mathcal{K}$ and reprograms $H(pw, m_1, m_2, id, id', w) := \mathsf{Key}$ for $\mathcal{A}$'s later hash query $(pw, m_1, m_2, id, id', w)$ s.t. $w = \mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$.
- For a responder instance $(\mathcal{CP}, j)$, suppose that (m_2, id') is the message generated by Sim for $(\mathcal{CP}, j)$, and (m_1, id) is the message received from $\mathcal{A}$. We assume that (m_1, id) is not generated by Sim, and hence (m_1, id) results from $\mathcal{A}$'s active attack. In this case, Sim searches in the hash list $\mathcal{L}_H$. If there exists $([pw], m_1, m_2, id, id', [w, k]) \in \mathcal{L}_H$ s.t. $w = \mathsf{SimComp_r}(td, pw, st_2, m_1, m_2)$ (at most one due to G_6), then Sim checks whether pw is the password used by $(\mathcal{CP}, j)$. If yes, then Sim sets $\mathsf{Key} := k$. If there exists no such record, Sim randomly samples $\mathsf{Key} \leftarrow_{\$} \mathcal{K}$ and reprograms $H(pw, m_1, m_2, id, id', w) := \mathsf{Key}$ for $\mathcal{A}$'s later hash query $(pw, m_1, m_2, id, id', w)$ s.t. $w = \mathsf{SimComp_r}(td, pw, st_2, m_1, m_2)$.

According to the ideal functionality of random oracle and the reprogramming technique, we know the changes are conceptual. So $\Pr[\mathsf{G}_7 \Rightarrow 1] = \Pr[\mathsf{G}_6 \Rightarrow 1]$.

Note that Sim almost gets rid of using pw when dealing with $\mathcal{A}$'s active attacks, except that Sim still uses the party's password pw to check the correctness of pw' in record $(pw', \ldots)$ of $\mathcal{L}_H$ so as to identify $\mathcal{A}$'s successful active attacks during the session key generation.

Game G_8(integration of Sim with $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$). In the final game G_8, Sim completely gets rid of pw. More precisely, Sim accesses $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ via interfaces $\mathsf{PassiveNewKey}$, $\mathsf{ActiveNewKey}$ and $\mathsf{LateTestPwd}$ to check the correctness of passwords and assigns session keys for party instances. The full description of Sim is shown in Fig. 11 in Appendix B.1.

We consider the following two cases, covering both passive and active attacks.

Passive Attacks: In this case, Sim generates session keys via the $\mathsf{PassiveNewKey}$ interface from $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$.

- For an instance $(\mathcal{P}, i)$ that receives a message (m_1, id) or (m_2, id') where (m_1, id) or (m_2, id') is generated by Sim for some instance $(\mathcal{CP}, j)$, Sim sets $ssid := m_1|m_2|id|id'$ and sends $(\mathsf{PassiveNewKey}, sid, \mathcal{P}, i, \mathcal{CP}, j, ssid)$ to $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$.

According to its ideal functionality, $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ assigns a uniform session key to instance $(\mathcal{P}, i)$ and keeps the session keys of $(\mathcal{P}, i)$ and $(\mathcal{CP}, j)$ consistent. Therefore, G_8 and G_7 are the same in this case.

Active Attacks: In this case, Sim generates session keys via $\mathsf{ActiveNewKey}$ interface and may reprogram the random oracle H with help of the $\mathsf{LateTestPwd}$ interface.

- For an initiator instance $(\mathcal{P}, i)$, suppose that (m_1, id) is generated by Sim invoking $(m_1, st_1) \leftarrow \mathsf{SimInit}(\mathsf{pp})$ for $(\mathcal{P}, i)$. Upon receiving message (m_2, id') that is generated by $\mathcal{A}$, Sim resorts to $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ to generate session key for $(\mathcal{P}, i)$. Sim searches in $\mathcal{L}_H$ to find $([pw], m_1, m_2, id, id', [w, K]) \in \mathcal{L}_H$ s.t. $w = \mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$. If there exists such a record (at most one), then Sim sends $(\mathsf{ActiveNewKey}, sid, \mathcal{P}, i, pw, K, ssid, id')$ to $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$. If there exists no such record, then Sim sends $(\mathsf{ActiveNewKey}, sid, \mathcal{P}, i, \perp, \perp, ssid, id')$. Later, if $\mathcal{A}$ issues a hash query on $([pw], m_1, m_2, id, id', [w])$ for H s.t. $w = \mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$, then Sim sends $(\mathsf{LateTestPwd}, sid, \mathcal{P}, i, ssid, pw)$ to $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ and obtains Key'. Then Sim programs $H(pw, m_1, m_2, id, id', w) := \mathsf{Key}'$.
- For a responder instance $(\mathcal{CP}, j)$, suppose that (m_2, id') is generated by Sim invoking $(m_2, st_2) \leftarrow \mathsf{SimResp}(\mathsf{pp})$ for $(\mathcal{CP}, j)$. Upon receiving message (m_1, id) that is generated by $\mathcal{A}$, Sim uses $\mathsf{ActiveNewKey}$ and $\mathsf{LateTestPwd}$ to generate session key Key in a similar way as above.

According to the specification of $\mathsf{ActiveNewKey}$, if pw in the record is the correct password, then $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ sets $\mathsf{Key} := H(pw, m_1, m_2, id, id', w)$. Otherwise $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ randomly samples $\mathsf{Key} \leftarrow_{\$} \mathcal{K}$. Furthermore, Sim can reprogram $H(pw, m_1, m_2, id, id', w)$ with help of $\mathsf{LateTestPwd}$ to keep consistence. In this case, G_8 and G_7 are the same.

So we have $\Pr[\mathsf{G}_8 \Rightarrow 1] = \Pr[\mathsf{G}_7 \Rightarrow 1]$.

Now that Sim completely gets rid of pw in the simulation, G_8 is exactly $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}}$. Therefore, $\Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}} \Rightarrow 1] = \Pr[\mathsf{G}_8]$.

Finally, by combining all the statements across G_0-G_8, we know that

$$|\Pr[\mathbf{Real}_{\mathcal{Z},\mathcal{A}} \Rightarrow 1] - \Pr[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}} \Rightarrow 1]| \leq \ell \cdot (\ell \cdot \mathsf{Adv}^{\mathsf{ow}}_{\mathsf{key}}(\mathcal{B}_{ow}) + \mathsf{Adv}^{\mathsf{Init}}_{\mathsf{Unipw}}(\mathcal{B}_{init}) + \mathsf{Adv}^{\mathsf{Resp}}_{\mathsf{Unipw}}(\mathcal{B}_{resp})).$$

□

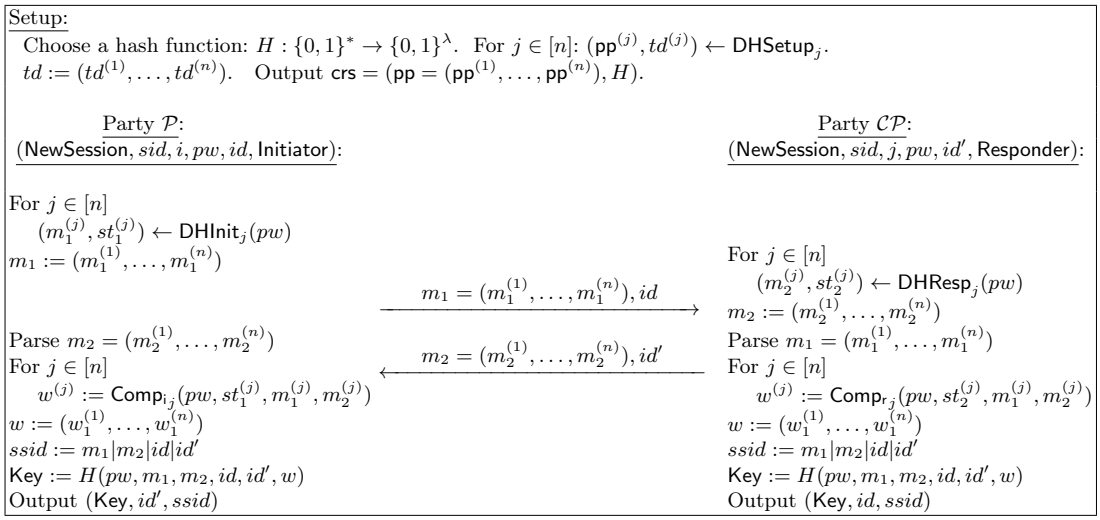

Fig. 7: Hybrid DH-type PAKE $\mathsf{HPAKE}_{\mathsf{DH}}$ constructed via parallel composition of $\mathsf{PAKE}_1, \ldots, \mathsf{PAKE}_n$, where $\mathsf{PAKE}_j = (\mathsf{DHSetup}_j, \mathsf{DHInit}_j, \mathsf{DHResp}_j, \mathsf{Comp}_{\mathsf{i}_j}, \mathsf{Comp}_{\mathsf{r}_j})$ for $j \in [n]$ are DH-type PAKEs.

3.3 Hybrid DH-type PAKE via Parallel Composition

Suppose $\mathsf{PAKE}_1, \ldots, \mathsf{PAKE}_n$ are DH-type PAKE schemes with $\mathsf{PAKE}_j = (\mathsf{DHSetup}_j, \mathsf{DHInit}_j, \mathsf{DHResp}_j, \mathsf{Comp}_{\mathsf{i}_j}, \mathsf{Comp}_{\mathsf{r}_j})$ for $j \in [n]$. Then we can compose $\mathsf{PAKE}_1, \ldots, \mathsf{PAKE}_n$ in parallel to obtain a new Hybrid DH-type PAKE scheme $\mathsf{HPAKE}_{\mathsf{DH}}$. The full description of scheme $\mathsf{HPAKE}_{\mathsf{DH}}$ is given in Fig. 7.

Theorem 2. *The hybrid PAKE scheme* $\mathsf{HPAKE}_{\mathsf{DH}}$ *in Fig. 7, constructed via parallel composition of* $\mathsf{PAKE}_1, \ldots, \mathsf{PAKE}_n$*, is a full DH-type PAKE, as long as one component* PAKE_j *is a full DH-type PAKE and the others* $\{\mathsf{PAKE}_i\}_{i \in [n] \setminus \{j\}}$ *are DH-type PAKE (not necessarily full) with statistical properties of* ②, ③.

Proof. Clearly $\mathsf{HPAKE}_{\mathsf{DH}}$ is a DH-type one as long as $\mathsf{PAKE}_1, \ldots, \mathsf{PAKE}_n$ are DH-type ones. And its correctness follows from the correctness of PAKE_1, ..., and PAKE_n.

Define $\mathsf{SimComp}_{\mathsf{i}}$ and $\mathsf{SimComp}_{\mathsf{r}}$ for $\mathsf{HPAKE}_{\mathsf{DH}}$ as follows.

$w \leftarrow \mathsf{SimComp}_{\mathsf{i}}(td, pw, st_1, m_1, m_2)$	$w \leftarrow \mathsf{SimComp}_{\mathsf{r}}(td, pw, st_2, m_1, m_2)$
Parse $td := (td^{(1)}, \ldots, td^{(n)})$	Parse $td := (td^{(1)}, \ldots, td^{(n)})$
Parse $st_1 := (st_1^{(1)}, \ldots, st_1^{(n)})$	Parse $st_2 := (st_2^{(1)}, \ldots, st_2^{(n)})$
Parse $m_1 = (m_1^{(1)}, \ldots, m_1^{(n)})$	Parse $m_1 = (m_1^{(1)}, \ldots, m_1^{(n)})$
Parse $m_2 := (m_2^{(1)}, \ldots, m_2^{(n)})$	Parse $m_2 := (m_2^{(1)}, \ldots, m_2^{(n)})$
$w^{(j)} = \mathsf{SimComp}_{\mathsf{i}_j}(td^{(j)}, pw, st_1^{(j)}, m_1^{(j)}, m_2^{(j)})$	$w^{(j)} = \mathsf{SimComp}_{\mathsf{r}_j}(td^{(j)}, pw, st_2^{(j)}, m_1^{(j)}, m_2^{(j)})$
Return $w = (w^{(1)}, \ldots, w^{(n)})$	Return $w = (w^{(1)}, \ldots, w^{(n)})$

Now we prove that $\mathsf{HPAKE}_{\mathsf{DH}}$ is a full DH-type one by showing that $\mathsf{HPAKE}_{\mathsf{DH}}$ has properties of ①-⑤.

Given the statistical properties ②③ of $\{\mathsf{PAKE}_j\}_{j\in[n]}$, hybrid arguments across n components guarantee that $\mathsf{HPAKE}_{\mathsf{DH}}$ has the properties of ② and ③.

Suppose a single component, say PAKE_j, has the properties of ①, ④ and ⑤. Then property "① One-wayness of DH-Key" of PAKE_j implies that $w^{(j)}$ is hard to compute, which further implies that $w := (w_1^{(1)}, \ldots, w_1^{(n)})$ is hard to compute for PPT adversaries as well. Therefore $\mathsf{HPAKE}_{\mathsf{DH}}$ has the property of ①. Property "④ unique password for Initiator" of PAKE_j implies there exists at most one pair $(pw, w^{(j)})$ satisfying $w^{(j)} = \mathsf{SimComp_i}_j(td^{(j)*}, pw, st_1^{(j)*}, m_1^{(j)*}, m_2^{(j)})$. Note that each $w^{(i)}$ is determined by pw when $m_1^{(i)}$ and $m_2^{(i)}$ are fixed. This implies that there exists at most one pair $(pw, w = (w^{(1)}, \ldots, w^{(n)})$ such that $w^{(i)} = \mathsf{SimComp_i}_j(td^{(i)*}, pw, st_1^{(i)*}, m_1^{(i)*}, m_2^{(i)})$ holds for all $i \in [n]$. Therefore, $\mathsf{HPAKE}_{\mathsf{DH}}$ has the property of ④. With a similar argument, $\mathsf{HPAKE}_{\mathsf{DH}}$ has the property of ⑤.

By Theorem 1 and Corollary 1, the hybrid PAKE scheme $\mathsf{HPAKE}_{\mathsf{DH}}$ in Fig. 7 is a full DH-type PAKE and can securely emulates both $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ and $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$.

4 Hybrid PAKE via Serial Composition of DH-Type PAKE and Other PAKE

Given a DH-type PAKE scheme $\mathsf{PAKE}_1 = (\mathsf{DHSetup}, \mathsf{DHInit}, \mathsf{DHResp}, \mathsf{Comp_i}, \mathsf{Comp_r})$ and a (two-round) PAKE scheme $\mathsf{PAKE}_2 = (\mathsf{Setup}, \mathsf{Init}, \mathsf{Resp}, \mathsf{Deri})$, we show how to use them to construct a Hybrid PAKE scheme HPAKE via serial composition of PAKE_1 and PAKE_2. In the serial composition, PAKE_1 will obtain its session key $k_1|k_2$, and then the first part k_1 is used as the password of PAKE_2. Next, PAKE_2 will obtain its session key k_3. Finally, the session key of HPAKE is computed by the hash value of the second part k_2 of PAKE_1's session key, PAKE_2's the session key, and the transcript of HPAKE, as shown in Fig. 8.

Let $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ be the (lazy extraction version of) ideal functionality of PAKE_1 providing interfaces NewSession1, Testpw1, RegisterTest1, LateTestPwd1 and NewKey1. Let $\mathcal{F}_{\mathsf{PAKE2}}$ be the ideal functionality of PAKE_2 providing interfaces NewSession2, Testpw2, RegisterTest2, LateTestPwd2 and NewKey2.

Let $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$ denote the leaky and lazy extraction version of ideal functionality of HPAKE providing interfaces NewSession, Testpw, RegisterTest, LateTestPwd, and NewKey, and $\mathcal{F}_{\mathsf{PAKE}}$ denote the normal ideal functionality of HPAKE, providing interfaces NewSession, Testpw, and NewKey.

In the following theorems, we prove the UC security of HPAKE can rely on either the UC security of PAKE_1 or the UC security of PAKE_2.

Theorem 3. *The PAKE scheme* HPAKE *in Fig.8 securely emulates* $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$ *if the underlying* $\mathsf{PAKE}_1 = (\mathsf{Setup}, \mathsf{DHInit}, \mathsf{DHResp}, \mathsf{Comp_i}, \mathsf{Comp_r})$ *securely emulates* $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$*, and hash function* G *works as random oracles.*

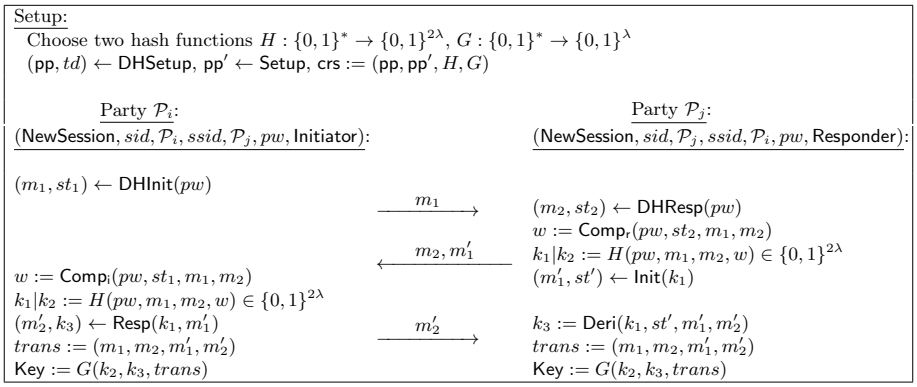

Fig. 8: Our Hybrid PAKE scheme HPAKE constructed via serial composition of a DH-type PAKE_1 and another PAKE_2. Here we use a two-round PAKE_2 for simplicity, but any (multi-round) PAKE_2 works as well.

Theorem 4. *The PAKE scheme* HPAKE *in Fig. 8 securely emulates* $\mathcal{F}_{\mathsf{PAKE}}$ *if the underlying* $\mathsf{PAKE}_2 = (\mathsf{Setup}, \mathsf{Init}, \mathsf{Resp}, \mathsf{Deri})$ *securely emulates* $\mathcal{F}_{\mathsf{PAKE}}$, $\mathsf{PAKE}_1 = (\mathsf{Setup}, \mathsf{DHInit}, \mathsf{DHResp}, \mathsf{Comp_i}, \mathsf{Comp_r})$ *is a DH-type PAKE satisfying statistical properties ②③ as per Def. 2, and hash functions* H, G *are (quantum-accessible) random oracles.*

Proof of Theorem 3. The proof sketch has shown in technique overview in section 1. So we move the full proof in Appendix C to save space.

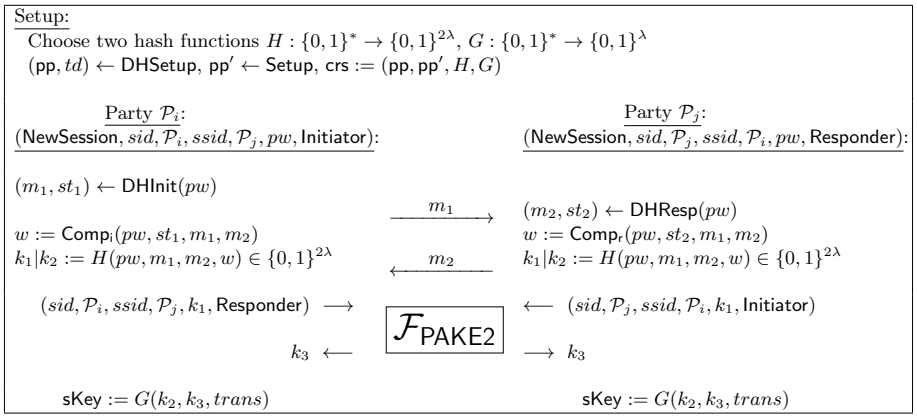

Fig. 9: HPAKE in $\mathcal{F}_{\mathsf{PAKE2}}$-hybrid model.

Proof of Theorem 4. We will prove it in the $\mathcal{F}_{\mathsf{PAKE2}}$-hybrid model, where the underlying PAKE_2 protocol is replaced by $\mathcal{F}_{\mathsf{PAKE2}}$, as shown in Fig. 9. Our goal is to

construct a simulator Sim and show that $\left|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}^{\mathcal{F}_{\mathsf{PAKE2}}} \Rightarrow 1\right] - \Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}[\mathcal{F}_{\mathsf{PAKE2}}]} \Rightarrow 1\right]\right|$ is negligible by employing a series of games.

Game G_0(real world). This is the $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}^{\mathcal{F}_{\mathsf{PAKE2}}}$ world where adversary $\mathcal{A}$ can access ideal functionality $\mathcal{F}_{\mathsf{PAKE2}}$ via interfaces like $\mathsf{Testpw2}$ and $\mathsf{NewKey2}$.

Game G_1(simulating each party with password pw). In this game, we introduce a simulator Sim who *additionally knows passwords* of all parties. Sim generates $\mathsf{crs} := (\mathsf{pp}, \mathsf{pp}', H, G)$ with $(\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}$ and $\mathsf{pp}' \leftarrow \mathsf{Setup}$ and keeps td. Then with help of passwords, it simulates the parties to generate transcripts (m_1, m_2) and session keys sKey for instances of the HPAKE protocol, just like G_0. Sim also simulates the underlying $\mathcal{F}_{\mathsf{PAKE2}}$ by itself with passwords. With the knowledge of *passwords*, Sim's simulations of $\mathcal{F}_{\mathsf{PAKE2}}$ and the behaviors of all parties are perfect. We have $\Pr[\mathsf{G}_1 \Rightarrow 1] = \Pr[\mathsf{G}_0 \Rightarrow 1]$.

Game G_2(simulating transcripts m_1, m_2 without pw). G_2 is the same as G_1, except for Sim's simulation of m_1 and m_2.

- For an initiator instance $(\mathcal{P}_i, ssid)$, Sim generates m_1 by $(m_1, st_1) \leftarrow \mathsf{SimInit}(\mathsf{pp})$ rather than $(m_1, st_1) \leftarrow \mathsf{DHInit}(pw)$. Correspondingly, when receiving the second message m_2, Sim computes $w := \mathsf{SimComp_i}(td, pw, st_1, m_1, m_2)$ rather than $w := \mathsf{Comp_i}(pw, st_1, m_1, m_2)$.
- For a responder instance $(\mathcal{P}_j, ssid)$, upon receiving a first-round message m_1, Sim generates m_2 by $(m_2, st_2) \leftarrow \mathsf{SimResp}(\mathsf{pp})$ rather than $(m_2, st_2) \leftarrow \mathsf{DHResp}(pw)$. Correspondingly, Sim computes $w := \mathsf{SimComp_r}(td, pw, st_2, m_1, m_2)$ rather than $w := \mathsf{Comp_r}(pw, st_2, m_1, m_2)$.

By the "② Perfect Simulation for Initiator" and "③ Perfect Simulation for Responder" properties of the DH-type PAKE_1, G_2 is identical to G_1. So we have

$$\Pr[\mathsf{G}_2 \Rightarrow 1] = \Pr[\mathsf{G}_1 \Rightarrow 1].$$

Game G_3(resorting to online-extractable QRO simulator $\mathcal{S}$ for H). In G_3, to simulate random oracle H, Sim invokes the random oracle simulator $\mathcal{S} = (\mathcal{S}.RO, \mathcal{S}.E)$ as per Theorem 5 (See Appendix A.2) with respect to function $f : f(x, y_1|y_2) := y_1$, and it also extracts the hash query from the value k_1. More precisely, Sim invokes $\mathcal{S}.RO(x)$ to simulate the quantum random oracle H. For each of $\mathcal{A}$'s queries $(\mathsf{Testpw2}, sid, \mathcal{P}, ssid, k_1)$ to the simulated $\mathcal{F}_{\mathsf{PAKE2}}$, Sim invokes $\mathcal{S}.E(k_1)$ to extract $(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w})$.

The only difference between G_3 and G_2 is the simulation of oracle H. We define the function $f(x, y_1|y_2) := y_1$ where $y_1|y_2 \in \{0,1\}^{2\lambda}$. It is easy to see $\Gamma(f) = 2^\lambda$ and $\Gamma'(f) = 2^\lambda$ (c.f. Def. 3 in Appendix A.2). The invocation of $\mathcal{S}.E(k_1)$ corresponds to Sim conducting measurements via $\mathcal{S}.E(k_1)$ w.r.t. f.

Since Sim invokes $\mathcal{S}.E$ at most ℓ times (limited by the number of instances), and $\mathcal{A}$ issues at most q quantum random oracle queries, Theorem 5(a) shows that $|\Pr[\mathsf{G}_3 \Rightarrow 1] - \Pr[\mathsf{G}_2 \Rightarrow 1]| \leq \frac{8\ell(q+\ell)}{\sqrt{2^{\lambda-1}}}$.

Game G_4(answer $\mathsf{Testpw2}$ with help of online-extractable QRO simulator $\mathcal{S}$). In G_4, $\mathsf{Testpw2}$ is answered by Sim with help of $\mathcal{S} = (\mathcal{S}.RO, \mathcal{S}.E)$.

– Upon receiving the $(\mathsf{Testpw2}, sid, \mathcal{P}, ssid, k_1')$ query from $\mathcal{A}$ to the underlying $\mathcal{F}_{\mathsf{PAKE2}}$, Sim invokes $\mathcal{S}.E(k_1')$ to extract $(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w})$. If $\hat{pw} = pw$, $(\hat{m}_1, \hat{m}_2)$ are transcripts sent and received by $(\mathcal{P}, ssid)$, and $\hat{w} = \mathsf{SimComp_i}(td, \hat{pw}, st_1, \hat{m}_1, \hat{m}_2)$ or $\hat{w} = \mathsf{SimComp_r}(td, \hat{pw}, st_2, \hat{m}_1, \hat{m}_2)$, then Sim returns "correct guess" to $\mathcal{A}$. Otherwise, Sim returns "wrong guess" to $\mathcal{A}$.

Recall that in G_3, for a $(\mathsf{Testpw2}, sid, \mathcal{P}, ssid, k_1')$ query, Sim computes $w := \mathsf{SimInit}(td, pw, st_1, m_1, m_2)$ or $w := \mathsf{SimResp}(td, pw, st_2, m_1, m_2)$, and $k_1|k_2 := H(pw, m_1, m_2, w)$, where pw is the correct password of $(\mathcal{P}, ssid)$ and m_1, m_2 are the messages sent and received by $(\mathcal{P}, ssid)$. If $k_1' = k_1$, then Sim returns "correct guess". Otherwise, Sim returns "wrong guess".

If $(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w}) = (pw, m_1, m_2, w)$, G_4 is the same as G_3. Now we consider the case of $(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w}) \neq (pw, m_1, m_2, w)$. Given $k_1|k_2 := H(pw, m_1, m_2, w)$, we further consider the following two cases.

Case I: $(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w}) \neq (pw, m_1, m_2, w)$ **and** $k_1' = k_1$**.** This case is bounded by Theorem 5(b).

Case II: $(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w}) \neq (pw, m_1, m_2, w)$ **and** $k_1' \neq k_1$**.** In this case, Sim in G_3 must return "wrong guess". If Sim in G_4 returns "correct guess", it must hold $\hat{pw} = pw$ and $(\hat{m}_1, \hat{m}_2) = (m_1, m_2)$. Recall that $\mathsf{SimComp_i}$ and $\mathsf{SimComp_r}$ are deterministic functions, so $\hat{w} = w$, which is contradictory to $(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w}) \neq (pw, m_1, m_2, w)$. Therefore, G_4 is the same as G_3 in this case.

According to Theorem 5(b), we have

$$|\Pr[\mathsf{G}_4 \Rightarrow 1] - \Pr[\mathsf{G}_3 \Rightarrow 1]| \leq \frac{8\ell(q+1)}{\sqrt{2^{\lambda-1}}} + \frac{40e^2(q+\ell+1)^3+2}{2^\lambda}.$$

Game G_5(simulating session keys sKey). G_5 is the same as G_4, except for the simulation of session keys sKey.

– Sim checks whether $\mathcal{A}$ ever issued a $(\mathsf{Testpw2}, sid, \mathcal{P}, ssid, k_1')$ query to the underlying $\mathcal{F}_{\mathsf{PAKE2}}$ and obtained "correct guess".
(1) If yes, then Sim computes $\mathsf{sKey} := G(k_2, k_3, trans)$ just like G_4.
(2) If no and there exists a party $\mathcal{CP}$ with session key sKey' that shares the same m_1, m_2, k_3, then Sim sets $\mathsf{sKey} := \mathsf{sKey}'$. Due to the correctness of HPAKE, G_5 and G_4 are the same in this case.
(3) If no and no other instance shares m_1, m_2, k_3 with $(\mathcal{P}, ssid)$, Sim randomly samples $\mathsf{sKey} \leftarrow_\$ \mathcal{K}$.

Note that (1) means a successful active attack, (2) means a passive attack, and (3) means a passive attack or an unsuccessful active attack.

Recall that in case (3), we have $\mathsf{sKey} := G(k_2, k_3, trans)$ in G_4. So G_5 and G_4 are the same except case (3) happens. According to the specification of $\mathcal{F}_{\mathsf{PAKE2}}$, $\mathcal{F}_{\mathsf{PAKE2}}$ will output a random $k_3 \leftarrow_\$ \{0,1\}^\lambda$ in case (3). Then Lemma 1 guarantees that the function $G(\cdot, k_3, \cdot)$ is a pseudo-random function in QROM, except with probability $\frac{2q}{\sqrt{2^\lambda}}$. Then different transcripts $trans$ in different instances make sure

that the outputs of quantum random oracle $G(k_2, k_3, trans)$ are independent and uniformly distributed in case (3). So $|\Pr[\mathsf{G}_5 \Rightarrow 1] - \Pr[\mathsf{G}_4 \Rightarrow 1]| \leq \frac{2q}{\sqrt{2^\lambda}}$.

Game G_6(Integration of Sim with $\mathcal{F}_{\mathsf{PAKE}}$). Up to now only Sim's simulation of transcripts (m_1, m_2) does not use password. In G_6, Sim will completely get rid of passwords. It will use interfaces of $\mathcal{F}_{\mathsf{PAKE}}$ (in the ideal world for HPAKE) to answer $\mathcal{A}$'s Testpw2 and NewKey2 queries to $\mathcal{F}_{\mathsf{PAKE2}}$ without using passwords. As long as no password is needed for answering Testpw2 query to $\mathcal{F}_{\mathsf{PAKE2}}$, Sim does not need passwords for generating session key sKey either (see G_5).

- For $(\mathsf{NewSession}, sid, \mathcal{P}, ssid, \mathcal{CP}, \cdot, \mathsf{role})$ from $\mathcal{P}$, if this is the first NewSession query on $ssid$, or this is the second NewSession query on $ssid$ and $(\mathcal{CP}, ssid, \mathcal{P}, \cdot)$ exists, then Sim records $(\mathcal{P}, ssid, \mathcal{CP}, \cdot)$ and marks it **fresh**.
- For $\mathcal{A}$'s query $(\mathsf{Testpw2}, sid, \mathcal{P}, ssid, k_1')$ to $\mathcal{F}_{\mathsf{PAKE2}}$, if there is a **fresh** record $(\mathcal{P}, ssid, [\mathcal{CP}, \cdot])$, Sim first invokes $\mathcal{S}.E(k_1')$ to extract $(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w})$. If $(\hat{m}_1, \hat{m}_2)$ are not transcripts sent and received by $(\mathcal{P}, ssid)$, or $\hat{w} \neq \mathsf{SimComp_i}(td, \hat{pw}, st_1, \hat{m}_1, \hat{m}_2)$ and $\hat{w} \neq \mathsf{SimComp_r}(td, \hat{pw}, st_2, \hat{m}_1, \hat{m}_2)$, then Sim marks the record **interrupted** and replies with "wrong guess" to $\mathcal{A}$. Otherwise, Sim issues query $(\mathsf{Testpw}, sid, \mathcal{P}, ssid, \hat{pw})$ to $\mathcal{F}_{\mathsf{PAKE}}$. If $\mathcal{F}_{\mathsf{PAKE}}$ returns "correct guess", then Sim marks the record as **compromised** and replies with "correct guess" to $\mathcal{A}$. Otherwise, Sim marks the record as **interrupted** and replies with "wrong guess" to $\mathcal{A}$.
- For $\mathcal{A}$'s query $(\mathsf{NewKey2}, sid, \mathcal{P}, ssid, \mathsf{Key}^*)$ to $\mathcal{F}_{\mathsf{PAKE2}}$, if record $(\mathcal{P}, ssid, [\mathcal{CP}, \cdot])$ is not marked **completed**, do the followings.
  - If the record is **compromised**, Sim sets $k_3 := \mathsf{Key}^*$, computes $\mathsf{sKey}^* = G(k_2, k_3, trans)$.
  - In all other cases, Sim sets $\mathsf{sKey}^* = \perp$.

  Sim issues $(\mathsf{NewKey}, sid, \mathcal{P}, ssid, \mathsf{sKey}^*)$ to $\mathcal{F}_{\mathsf{PAKE}}$ and marks the record $(\mathcal{P}, ssid, \mathcal{CP}, \cdot)$ **completed**.

According to the specification of $\mathcal{F}_{\mathsf{PAKE}}$, for $(\mathsf{Testpw}, sid, \mathcal{P}, ssid, \hat{pw})$ query, $\mathcal{F}_{\mathsf{PAKE}}$ returns "correct guess" iff $\hat{pw} = pw$. Therefore, Sim's simulation of answer to Testpw2 in G_6 is the same as that in G_5, but without pw. At the same time, if $\mathcal{A}$'s Testpw2 query results in "correct guess", then the instance is compromised. In G_6, we have $\mathsf{sKey}^* = G(k_2, k_3, trans)$ for compromised instance. Accordingly, for $(\mathsf{NewKey}, sid, \mathcal{P}, ssid, \mathsf{sKey}^*)$ query, $\mathcal{F}_{\mathsf{PAKE}}$ will set the session key of $(\mathcal{P}, ssid)$ with $\mathsf{sKey} := \mathsf{sKey}^* = G(k_2, k_3, trans)$. For uncorrupted $(\mathcal{P}, ssid)$, $\mathcal{F}_{\mathsf{PAKE}}$ will either keep sKey consistent between partnered instances in case of passive attacks or sample a random key as the session key sKey in other cases. Therefore, the simulation of session keys sKey in G_6 is also identical to G_5. So $\Pr[\mathsf{G}_6 \Rightarrow 1] = \Pr[\mathsf{G}_5 \Rightarrow 1]$.

Now Sim can simulate the whole experiment only with access to the ideal functionality $\mathcal{F}_{\mathsf{PAKE}}$ and does not need passwords anymore. The full description of simulator Sim is shown in Fig. 13 in Appendix B.3. G_6 is exactly $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}[\mathcal{F}_{\mathsf{PAKE2}}]}$. Therefore, $\Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}[\mathcal{F}_{\mathsf{PAKE2}}]} \Rightarrow 1\right] = \Pr[\mathsf{G}_6 \Rightarrow 1]$.

Finally, by combining all the statements across G_0-G_6, we have

$$\left|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}^{\mathcal{F}_{\mathsf{PAKE2}}}\right] - \Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}[\mathcal{F}_{\mathsf{PAKE2}}]}\right]\right| \leq \frac{16\ell(q+\ell)}{\sqrt{2^{\lambda-1}}} + \frac{40e^2(q+\ell+1)^3+2}{2^\lambda} + \frac{2q}{\sqrt{2^\lambda}}.$$

5 Instantiations

We show that PAKE schemes SPAKE2[7], CrsX-GA-PAKE[4,25], and TBPEKE[32] are all fall into the framework of full DH-type PAKE. Due to short of space, we put the formal proofs in Appendix D. Recall that Abdalla et al. proved that SPAKE2 and TBPEKE can securely emulate $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}$ in [3]. In comparison, our results are stronger since we prove that SPAKE2, TBPEKE, and CrsX-GA-PAKE can securely emulate a better ideal functionality $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$. Moreover, we give the first security proof for CrsX-GA-PAKE in the UC framework.

We also have lots of choices for PAKE schemes from post-quantum assumption, like those in [33,12,9,30]. These PAKE schemes securely emulate $\mathcal{F}_{\mathsf{PAKE}}$.

Different choices of full DH-type PAKEs and UC-secure PAKE via parallel composition or serial composition result in different hybrid PAKE schemes reflecting different features. We list a few of them, whose UC security can either rely on traditional assumptions or post-quantum assumptions.

• **Round-Optimal Hybrid PAKE.** Composing SPAKE2 and CrsX-GA-PAKE in parallel results in a *round-optimal* hybrid PAKE based on assumptions over groups or isogenies.

• **Efficient Hybrid PAKE.** Composing SPAKE2 and the recent EKE-based PAKE scheme CHIC [9] in serial results in an *efficient* three-round hybrid PAKE scheme based on assumptions over groups or lattices.

• **Hybrid PAKE in QROM.** Composing SPAKE2 and $\mathsf{PAKE}^{\mathsf{QRO}}$[30] in serial results in a four-round hybrid PAKE scheme, and it has *UC security in QROM* as long as the underlying $\mathsf{PAKE}^{\mathsf{QRO}}$ is.

References

1. Anssi views on the post-quantum cryptography transition (2022), https://cyber.gouv.fr/en/publications/anssi-views-post-quantum-cryptography-transition
2. Cryptographic algorithms and key lengths (2024), https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile
3. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 278–307. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_10
4. Abdalla, M., Eisenhofer, T., Kiltz, E., Kunzweiler, S., Riepel, D.: Password-authenticated key exchange from group actions. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 699–728. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_24

5. Abdalla, M., Haase, B., Hesse, J.: Security analysis of CPace. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 711–741. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92068-5_24
6. Abdalla, M., Haase, B., Hesse, J.: CPace, a balanced composable PAKE. Internet-Draft draft-irtf-cfrg-cpace-12, Internet Engineering Task Force (Sep 2024), https://datatracker.ietf.org/doc/draft-irtf-cfrg-cpace/12/, work in Progress
7. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Berlin, Heidelberg (Feb 2005). https://doi.org/10.1007/978-3-540-30574-3_14
8. Alamati, N., De Feo, L., Montgomery, H., Patranabis, S.: Cryptographic group actions and applications. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part II. LNCS, vol. 12492, pp. 411–439. Springer, Cham (Dec 2020). https://doi.org/10.1007/978-3-030-64834-3_14
9. Arriaga, A., Barbosa, M., Jarecki, S., Skrobot, M.: C'est très CHIC: A compact password-authenticated key exchange from lattice-based KEM. Cryptology ePrint Archive, Paper 2024/308 (2024), https://eprint.iacr.org/2024/308
10. Barbosa, M., Connolly, D., Duarte, J.D., Kaiser, A., Schwabe, P., Varner, K., Westerbaan, B.: X-wing. CiC **1**(1), 21 (2024). https://doi.org/10.62056/a3qj89n4e
11. Barbosa, M., Gellert, K., Hesse, J., Jarecki, S.: Bare PAKE: Universally composable key exchange from just passwords. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part II. LNCS, vol. 14921, pp. 183–217. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68379-4_6
12. Beguinet, H., Chevalier, C., Pointcheval, D., Ricosset, T., Rossi, M.: GeT a CAKE: Generic transformations from key encaspulation mechanisms to password authenticated key exchanges. In: Tibouchi, M., Wang, X. (eds.) ACNS 23International Conference on Applied Cryptography and Network Security, Part II. LNCS, vol. 13906, pp. 516–538. Springer, Cham (Jun 2023). https://doi.org/10.1007/978-3-031-33491-7_19
13. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Berlin, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_11
14. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy. pp. 72–84. IEEE Computer Society Press (May 1992). https://doi.org/10.1109/RISP.1992.213269
15. Bindel, N., Hale, B.: A note on hybrid signature schemes. Cryptology ePrint Archive, Report 2023/423 (2023), https://eprint.iacr.org/2023/423
16. Bindel, N., Hamburg, M., Hövelmanns, K., Hülsing, A., Persichetti, E.: Tighter proofs of CCA security in the quantum random oracle model. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 61–90. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-36033-7_3
17. Bindel, N., Herath, U., McKague, M., Stebila, D.: Transitioning to a quantum-resistant public key infrastructure. In: Lange, T., Takagi, T. (eds.) Post-Quantum Cryptography - 8th International Workshop, PQCrypto 2017. pp. 384–405. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59879-6_22
18. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Berlin, Heidelberg (May 2005). https://doi.org/10.1007/11426639_24

19. Canetti, R., Rabin, T.: Universal composition with joint state. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 265–281. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_16
20. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 395–427. Springer, Cham (Dec 2018). https://doi.org/10.1007/978-3-030-03332-3_15
21. Crockett, E., Paquin, C., Stebila, D.: Prototyping post-quantum and hybrid key exchange and authentication in TLS and SSH. Cryptology ePrint Archive, Report 2019/858 (2019), https://eprint.iacr.org/2019/858
22. Don, J., Fehr, S., Majenz, C., Schaffner, C.: Online-extractability in the quantum random-oracle model. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part III. LNCS, vol. 13277, pp. 677–706. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07082-2_24
23. Giacon, F., Heuer, F., Poettering, B.: KEM combiners. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 190–218. Springer, Cham (Mar 2018). https://doi.org/10.1007/978-3-319-76578-5_7
24. Haase, B., Labrique, B.: AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. IACR TCHES **2019**(2), 1–48 (2019). https://doi.org/10.13154/tches.v2019.i2.1-48, https://tches.iacr.org/index.php/TCHES/article/view/7384
25. Ishibashi, R., Yoneyama, K.: Compact password authenticated key exchange from group actions. In: Simpson, L., Baee, M.A.R. (eds.) ACISP 23. LNCS, vol. 13915, pp. 220–247. Springer, Cham (Jul 2023). https://doi.org/10.1007/978-3-031-35486-1_11
26. Jablon, D.P.: Strong password-only authenticated key exchange. Comput. Commun. Rev. **26**(5), 5–26 (1996). https://doi.org/10.1145/242896.242897, https://doi.org/10.1145/242896.242897
27. Katz, J., Rosenberg, M.: LATKE: A framework for constructing identity-binding PAKEs. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part II. LNCS, vol. 14921, pp. 218–250. Springer, Cham (Aug 2024). https://doi.org/10.1007/978-3-031-68379-4_7
28. Katz, J., Vaikuntanathan, V.: Smooth projective hashing and password-based authenticated key exchange from lattices. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 636–652. Springer, Berlin, Heidelberg (Dec 2009). https://doi.org/10.1007/978-3-642-10366-7_37
29. Liu, X., Liu, S., Han, S., Gu, D.: EKE meets tight security in the Universally Composable framework. In: Boldyreva, A., Kolesnikov, V. (eds.) PKC 2023, Part I. LNCS, vol. 13940, pp. 685–713. Springer, Cham (May 2023). https://doi.org/10.1007/978-3-031-31368-4_24
30. Lyu, Y., Liu, S., Han, S.: Universal composable password authenticated key exchange for the post-quantum world. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part VII. LNCS, vol. 14657, pp. 120–150. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58754-2_5
31. Okamoto, T., Pointcheval, D.: The gap-problems: A new class of problems for the security of cryptographic schemes. In: Kim, K. (ed.) PKC 2001. LNCS, vol. 1992, pp. 104–118. Springer, Berlin, Heidelberg (Feb 2001). https://doi.org/10.1007/3-540-44586-2_8
32. Pointcheval, D., Wang, G.: VTBPEKE: Verifier-based two-basis password exponential key exchange. In: Karri, R., Sinanoglu, O., Sadeghi, A.R., Yi, X. (eds.)

ASIACCS 17. pp. 301–312. ACM Press (Apr 2017). https://doi.org/10.1145/3052973.3053026
33. Santos, B.F.D., Gu, Y., Jarecki, S.: Randomized half-ideal cipher on groups with applications to UC (a)PAKE. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part V. LNCS, vol. 14008, pp. 128–156. Springer, Cham (Apr 2023). https://doi.org/10.1007/978-3-031-30589-4_5
34. Shoup, V.: Security analysis of itSPAKE2+. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part III. LNCS, vol. 12552, pp. 31–60. Springer, Cham (Nov 2020). https://doi.org/10.1007/978-3-030-64381-2_2

Appendix

A More Preliminaries

A.1 ROM and QROM

In the Random Oracle Model (ROM), a cryptographic hash function $H : \mathcal{X} \to \mathcal{Y}$ is idealized as a truly random function $\mathsf{RF} : \mathcal{X} \to \mathcal{Y}$. And any adversary needs to query H on inputs $x \in \mathcal{X}$ to learn the hash values $H(x)$.

In the quantum world, a quantum algorithm $\mathcal{A}$ can perform superposition queries to the random oracle H, and then oracle H behaves as a unitary operation $|x\rangle|y\rangle \mapsto |x\rangle|y \oplus H(x)\rangle$. In this case, H becomes a quantum random oracle (QRO). The QRO model supports classical queries x on H, and this can be formalized as setting query register and output register to be $|x\rangle|0\rangle$, and measuring the output register after the unitary operation $|x\rangle|0\rangle \mapsto |x\rangle|0 \oplus H(x)\rangle$.

A.2 Useful Lemmas in QROM

We recall the online-extractable technique [22] and a corollary [16] of O2H Lemma.

Definition 3 ([22]). *Let $f : \mathcal{X} \times \{0,1\}^n \to \mathcal{C}$ be an arbitrary fixed function. Define $\Gamma(f) := \max_{x,c} |\{y | f(x,y) = c\}|$ and $\Gamma'(f) := \max_{x \neq x', y'} |\{y | f(x,y) = f(x',y')\}|$.*

Theorem 5 (Summary of Corollary 4 in [22]). *For any fixed deterministic function $f : \mathcal{X} \times \{0,1\}^n \to \mathcal{C}$ and a random oracle RO, there exists an extractable RO-simulator $\mathcal{S} = (\mathcal{S}.RO, \mathcal{S}.E)$ satisfying the following properties.*

- *$\mathcal{S}.RO$ simulates random oracle RO.*
- *$\mathcal{S}.E$ extracts element $\hat{x} \in \mathcal{X}$ from element $t \in \mathcal{C}$.*
- *Let $\mathbf{x} = (x_1, x_2, ..., x_\ell)$ be a randomized classical values, and W be a quantum register with a state $\rho_W^{\boldsymbol{x}}$ that depends on $\mathbf{x}$. Let $\delta([x,W]_{\mathsf{G}}, [x,W]_{\mathsf{G}'})$ be the trace distance of the respective density matrices in game G and in game G'. For any quantum algorithm $\mathcal{A}$ that outputs $\mathbf{t} = (t_1, ..., t_\ell)$ in (possibly) different rounds, and outputs $\boldsymbol{x} \in \mathcal{X}^\ell$ and W at the end of the run, if $\mathcal{A}$ makes q (quantum) queries in total, define $\mathsf{RO}(\mathbf{x}) := (\mathsf{RO}(x_1), ..., \mathsf{RO}(x_\ell))$ and $\mathcal{S}.RO(\mathbf{x}) := (\mathcal{S}.RO(x_1), ..., \mathcal{S}.RO(x_\ell))$ then*
  (a) $\delta([\mathbf{t}, \mathbf{x}, \mathsf{RO}(\mathbf{x}), W]_{\mathsf{Exp}^{\mathsf{RO}}_{\mathcal{A}}}, [\mathbf{t}, \mathbf{x}, S.RO(\mathbf{x}), W]_{\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}}) \leq 8\ell(q+\ell)\sqrt{2\Gamma(f)/2^n}$,
  (b) $\Pr\left[\exists i : x_i \neq \hat{x}_i \wedge f(x_i, \mathcal{S}.RO(x_i)) = t_i \text{ in } \mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}\right] \leq 8\ell(q+1)\sqrt{2\Gamma/2^n} + \frac{40e^2(q+\ell+1)^3\Gamma'(f)+2}{2^n}$,
  where $\mathsf{Exp}^{\mathsf{RO}}_{\mathcal{A}}$ and $\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}$ are described in Fig. 10.

$\underline{\mathsf{Exp}^{\mathsf{RO}}_{\mathcal{A}}}$:	$\underline{\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}}$
$(\mathbf{t}, st) \leftarrow \mathcal{A}^{\mathsf{RO}}$	$(\mathbf{t}, st) \leftarrow \mathcal{A}^{\mathcal{S}.RO}$
$(\mathbf{x}, W) \leftarrow \mathcal{A}^{\mathsf{RO}}(st)$	$(\hat{x}_1, ..., \hat{x}_\ell) \leftarrow (\mathcal{S}.E(t_1), ..., \mathcal{S}.E(t_\ell))$
Output $(\mathbf{t}, \mathbf{x}, W)$	$(\mathbf{x}, W) \leftarrow \mathcal{A}^{\mathcal{S}.RO}(st)$
	Output $(\mathbf{t}, \mathbf{x}, W)$

Fig. 10: The original experiment $\mathsf{Exp}^{\mathsf{RO}}_{\mathcal{A}}$ executed by $\mathcal{A}$ equipped with RO, and simulated experiment $\mathsf{Exp}^{\mathcal{S}}_{\mathcal{A}}$ executed by $\mathcal{A}$ equipped with $\mathcal{S} = (\mathcal{S}.RO, \mathcal{S}.E)$.

Lemma 1 (PRF from QROM, Corollary 1 from [16]). *Let $G : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a quantum-accessible random oracle. The function $f(k, x) := G(k|x)$ can be used as a quantum-accessible PRF with a key $k \leftarrow_\$ \mathcal{K}$. More precisely, for any quantum algorithm $\mathcal{A}$ making at most q queries to G and any number of queries to oracle $f(k, \cdot)$ such that $f(k, x^*)$ is never queried, its advantage satisfies*

$$\mathsf{Adv}^{\mathsf{ps}}_{\mathsf{PRF}}(\mathcal{A}) := \Big|\Pr\left[k \leftarrow_\$ \mathcal{K}, x^* \leftarrow \mathcal{A}^{f(k,\cdot)}; y := f(k, x^*) : \mathcal{A}^{f(k,\cdot)}(x^*, y) \Rightarrow 1\right] - \Pr\left[k \leftarrow_\$ \mathcal{K}, x^* \leftarrow \mathcal{A}^{f(k,\cdot)}; y \leftarrow_\$ \mathcal{Y} : \mathcal{A}^{f(k,\cdot)}(x^*, y) \Rightarrow 1\right]\Big| \leq \frac{2q}{\sqrt{|\mathcal{K}|}}.$$

A.3 Computational Assumptions

Definition 4 (Gap-CDH Assumption[31]). *The Gap-CDH Assumption holds over a group $\mathbb{G}$ of prime order q and generator g, if for any PPT adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{Gap\text{-}CDH}}_{\mathbb{G},g,q}(\mathcal{A}) := \Pr\left[x, y \leftarrow_\$ \mathbb{Z}_q, Z \leftarrow \mathcal{A}^{\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)}(g^x, g^y) : Z = g^{xy}\right] = \mathsf{negl}(\lambda),$$

where $\mathsf{DDH}(h, h^x, h^y, Z) := [Z \stackrel{?}{=} h^{xy}]$.

Definition 5 (Gap-SDH Assumption[32]). *The Gap Simultaneous Diffie-Hellman (Gap-SDH) assumption holds over a group $\mathbb{G}$ of prime order q and generator g, if for any PPT adversary $\mathcal{A}$,*

$$\mathsf{Adv}^{\mathsf{Gap\text{-}SDH}}_{\mathbb{G},g,q}(\mathcal{A}) := \Pr\left[\begin{array}{c} X \leftarrow_\$ \mathbb{G}, a, b \leftarrow_\$ \mathbb{Z}_q, \\ (Y \neq 1, R, S) \leftarrow \mathcal{A}^{\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)}(X, X^a, X^b) \end{array} : \begin{array}{c} R = Y^{1/a} \\ S = Y^{1/b} \end{array}\right],$$

where $\mathsf{DDH}(h, h^x, h^y, Z) := [Z \stackrel{?}{=} h^{xy}]$

Cryptographic Group Actions. We will focus on group actions where $\mathbb{G}$ is *abelian* and the action is *regular*. We recall the notion of restricted effective group actions (REGA) as follows.

Definition 6 (Restricted Effective Group Action [8]). *A group action $(\mathbb{G}, \mathcal{X}, \star)$ is a restricted effective group action (REGA) if properties 1-5 are satisfied.*

1. *The group $\mathbb{G}$ is generated by a set $\{g_1, \ldots, g_n\}$.*
2. *The group $\mathbb{G}$ is finite and $n = \mathsf{poly}(\log |\mathbb{G}|)$.*
3. *The set $\mathcal{X}$ is finite and there exist PPT algorithms for membership testing and for computing unique representation of set element.*
4. *There exists a distinguished element $\tilde{x} \in \mathcal{X}$, called the origin, such that its representation is known.*
5. *There exists an efficient algorithm that given g_i in the generating set and any $x \in \mathcal{X}$, outputs $g_i \star x$ and $g_i^{-1} \star x$ where $i \in [n]$.*

With a REGA, we can use the generating set to approximate the random sampling process of $g \leftarrow_\$ \mathbb{G}$. The regularity of the $(\mathbb{G}, \mathcal{X}, \star)$ enables an efficient algorithm to sample $x \leftarrow_\$ \mathcal{X}$ uniformly. We can instantiate REGA with isogeny-based group actions CSIDH [20].

Definition 7 (GA-GapCDH Assumption[4]). *The Group Action Gap Computation Diffie-Hellman (GA-GapCDH) holds over a* $\mathsf{REGA} = (\mathbb{G}, \mathcal{X}, \star, \tilde{x})$*, if for any PPT adversary* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{REGA}}(\mathcal{A}) = \mathsf{negl}(\lambda)$*, where*

$$\mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{REGA}}(\mathcal{A}) := \Pr\left[(g,h) \leftarrow_\$ \mathbb{G}, z \leftarrow \mathcal{A}^{\mathsf{GA\text{-}DDH}(\cdot,\cdot,\cdot,\cdot)}(g \star \tilde{x}, h \star \tilde{x}) : z = (g \cdot h) \star \tilde{x}\right],$$

and $\mathsf{GA\text{-}DDH}(x, u \star x, s \star x, z) := [z \stackrel{?}{=} (u \cdot s) \star x]$.

Definition 8 (DSim-GA-GapCDH Assumption[4]). *The Double Simultaneous GA-GapCDH (DSim-GA-GapCDH) assumption holds over a* $\mathsf{REGA} = (\mathbb{G}, \mathcal{X}, \star, \tilde{x})$*, if for any PPT adversary* $\mathcal{A}$*,* $\mathsf{Adv}^{\mathsf{DSim\text{-}GA\text{-}GapCDH}}_{\mathsf{REGA}}(\mathcal{A}) = \mathsf{negl}(\lambda)$*, where*

$$\mathsf{Adv}^{\mathsf{DSim\text{-}GA\text{-}GapCDH}}_{\mathsf{REGA}}(\mathcal{A}) := \Pr\left[\begin{array}{cc} (g_0, g_1, h_0, h_1) \leftarrow_\$ \mathbb{G}^4 & z_0 = (g_0^{-1} h_0) \star z \\ (x_0, x_1) := (g_0 \star \tilde{x}, g_1 \star \tilde{x}) & z_1 = (g_0^{-1} h_1) \star z \\ (y_0, y_1) := (h_0 \star \tilde{x}, h_1 \star \tilde{x}) & : \; z_2 = (g_1^{-1} h_0) \star z \\ (z, z_0, z_1, z_2, z_3) \leftarrow \mathcal{A}^{\mathsf{GA\text{-}DDH}(\cdot,\cdot,\cdot,\cdot)}(x_0, x_1, y_0, y_1) & z_3 = (g_1^{-1} h_1) \star z \end{array}\right],$$

and $\mathsf{GA\text{-}DDH}(x, u \star x, s \star x, w) := [w \stackrel{?}{=} (u \cdot s) \star x]$.

Remark 1. In our paper, we use the gap-variants of computational assumptions, which is slightly stronger than the assumptions used in [5,4,25].

B Descriptions of Simulators.

B.1 Description of Simulator in Theorem 1

Initialization

Sim maintains lists $\mathcal{L}_H, \mathcal{T}, \mathsf{sent}, \mathsf{recv}$ (all initialized to be empty) in the simulation

- $\mathcal{L}_H$: store records to simulate random oracles H.
- $\mathcal{T}$: store RO records to reprogram.
- sent : store messages sent by party instances
- recv : store messages received by party instances

Sim invokes $(\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}$

Sim outputs $\mathsf{crs} := (\mathsf{pp}, H)$ and stores td

PAKE Sessions

on $(\mathsf{NewSession}, sid, \mathcal{P}, i, id, \mathsf{role})$ from $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$:

If $\mathsf{role} = \mathsf{Initiator}$: $(m, st) \leftarrow \mathsf{SimInit}(\mathsf{pp})$

Else $\mathsf{role} = \mathsf{Responder}$: $(m, st) \leftarrow \mathsf{SimResp}(\mathsf{pp})$

$\mathsf{sent} := \mathsf{sent} \cup \{(\mathcal{P}, i, m, st, id, \mathsf{role})\}$, send (m, id) from $\mathcal{P}$ to $\mathcal{A}$.

on (m', id') from $\mathcal{A}$ as a message from $\mathcal{CP}$ to $(\mathcal{P}, i)$:

Retrieve the record $(\mathcal{P}, i, [m, st, id, \mathsf{role}]) \in \mathsf{sent}$, ignore message if no such record exists

If $\mathsf{role} = \mathsf{Initiator}$: set $(m_1, id_1, m_2, id_2) := (m, id, m', id')$

Else $\mathsf{role} = \mathsf{Responder}$: set $(m_1, id_1, m_2, id_2) := (m', id', m, id)$

Set $ssid := m_1 | m_2 | id_1 | id_2$

If $\exists([\mathcal{CP}, j,]m', [st'], id', \mathsf{role}' \neq \mathsf{role}) \in \mathsf{sent}$:

Send $(\mathsf{PassiveNewKey}, sid, \mathcal{P}, i, \mathcal{CP}, j, ssid)$ to $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$

Else If $\exists([pw], m_1, m_2, id_1, id_2, [w, k]) \in \mathcal{L}_H$:

If $\mathsf{role} = \mathsf{Initiator} \wedge w = \mathsf{SimComp_i}(td, pw, st, m_1, m_2)$

$\vee \mathsf{role} = \mathsf{Responder} \wedge w = \mathsf{SimComp_r}(td, pw, st, m_1, m_2)$:

Send $(\mathsf{ActiveNewKey}, sid, \mathcal{P}, i, pw, k, ssid, id')$ to $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$

Else:

Send $(\mathsf{ActiveNewKey}, sid, \mathcal{P}, i, \perp, \perp, ssid, id')$ to $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$

$\mathcal{T} := \mathcal{T} \cup \{(\mathcal{P}, i, m_1, m_2, id_1, id_2, st, \mathsf{role})\}$.

Random Oracle

on $H(pw, m_1, m_2, id, id', w)$ from A:

If $\exists(pw, m_1, m_2, id, id', w, [k]) \in \mathcal{L}_H$: return k

If $([\mathcal{P}, i], m_1, m_2, id, id', [st, \mathsf{role}]) \in \mathcal{T}$:

If $\mathsf{role} = \mathsf{Initiator} \wedge w = \mathsf{SimComp_i}(td, pw, st, m_1, m_2)$

$\vee \mathsf{role} = \mathsf{Responder} \wedge w = \mathsf{SimComp_r}(td, pw, st, m_1, m_2)$:

Send $(\mathsf{LateTestPwd}, sid, \mathcal{P}, i, m_1 | m_2 | id | id', pw)$ to $\mathcal{F}_{\mathsf{bPAKE}^{\mathsf{LE}}}$ and obtain a result k

$\mathcal{L}_H := \mathcal{L}_H \cup \{(pw, m_1, m_2, id, id', w, k)\}$, return k

In other cases: $k \leftarrow_{\$} \mathcal{K}$, $\mathcal{L}_H := \mathcal{L}_H \cup \{(pw, m_1, m_2, id, id', w, k)\}$, return k

Fig. 11: Simulator Sim for Theorem 1.

B.2 Description of Simulator in Theorem 3

Initialization
Sim maintains a list $\mathcal{L}_G$ to store records to simulate random oracle G
Sim invokes $(\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}$, $\mathsf{pp}' \leftarrow \mathsf{Setup}$, outputs $\mathsf{crs} := (\mathsf{pp}, \mathsf{pp}', H, G)$.
Ideal Functionality $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$
on $(\mathsf{NewSession1}, sid, \mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ from $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$:
  Send $(\mathsf{NewSession1}, sid, \mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ to $\mathcal{A}$, store $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ as a **fresh** record
on $(\mathsf{Testpw1}, sid, \mathcal{P}, ssid, pw)$ from $\mathcal{A}$:
  Send $(\mathsf{Testpw}, sid, \mathcal{P}, ssid, pw)$ to $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$ and forward $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$'s answer to $\mathcal{A}$
  If $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$ returns "correct guess": mark $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ **compromised**
on $(\mathsf{RegisterTest1}, sid, \mathcal{P}, ssid)$ from $\mathcal{A}$:
  If $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ is **fresh**: mark $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ **interrupted** and add a "**tested**" flag
on $(\mathsf{LateTestPwd1}, sid, \mathcal{P}, ssid, pw')$ from $\mathcal{A}$:
  If there exists a **completed** $(\mathcal{P}, ssid, [\mathcal{CP}, \mathsf{role}])$ record with flag **tested**:
    Send $(\mathsf{Testpw}, sid, \mathcal{P}, ssid, pw)$ to $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$, remove the **tested** flag
    If $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$ returns "correct guess":
      Mark $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ **compromised**, retrieve the record $(\mathcal{P}, ssid, [\mathsf{Key}, k_3, trans])$, return Key to $\mathcal{A}$
    Else: $\mathsf{Key}' \leftarrow_{\$} \{0,1\}^{2\lambda}$, return Key' to $\mathcal{A}$
  Else if there exists a **completed** $(\mathcal{P}, ssid, [\mathcal{CP}, \mathsf{role}])$ record with flag **latetest**:
    Send $(\mathsf{LateTestPwd}, sid, \mathcal{P}, ssid, pw')$ to $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$, remove the **latetest** flag
    If $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$ returns sKey and "correct guess":
      Retrieve the record $(\mathcal{P}, ssid, [\mathsf{Key} = k_1 | k_2, k_3, trans])$, $\mathcal{L}_G := \mathcal{L}_G \cup \{(k_2, k_3, trans, \mathsf{sKey})\}$, return Key to $\mathcal{A}$
    Else: $\mathsf{Key}' \leftarrow_{\$} \{0,1\}^{2\lambda}$, return Key' to $\mathcal{A}$
on $(\mathsf{NewKey1}, sid, \mathcal{P}, ssid, \mathsf{Key}^*)$ from $\mathcal{A}$:
  If there is a record $(\mathcal{P}, ssid, [\mathcal{CP}, \mathsf{role}])$ not marked **completed**:
    If the record is **compromised**: set $\mathsf{Key} := \mathsf{Key}^*$
    If the record is **fresh** and there is a **completed** record $(\mathcal{CP}, ssid, \mathcal{P}, \mathsf{role}' \neq \mathsf{role})$
    which was **fresh** when $(\mathcal{CP}, ssid)$ outputs Key': retrieve $(\mathcal{CP}, ssid, [\mathsf{Key}'])$, set $\mathsf{Key} := \mathsf{Key}'$
    In all other cases: set $\mathsf{Key} \leftarrow_{\$} \{0,1\}^{2\lambda}$
    Store $(\mathcal{P}, ssid, \mathsf{Key})$, mark $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ **completed**, return Key to $\mathcal{P}$
PAKE Sessions
on Key from Sim as a message from $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ to $(\mathcal{P}, ssid)$:
  Retrieve the record $(\mathcal{P}, ssid, [\mathcal{CP}, \mathsf{role}])$, parse $\mathsf{Key} = k_1 | k_2 \in \{0,1\}^{2\lambda}$
  $(m_1', st_1') \leftarrow \mathsf{Init}(k_1)$, store $(\mathcal{P}, ssid, st_1')$, return m_1' from $\mathcal{P}$ to $\mathcal{A}$
on m_1' from $\mathcal{A}$ as a message from $\mathcal{CP}$ to $(\mathcal{P}, ssid)$:
  Retrieve the record $(\mathcal{P}, ssid, [\mathcal{CP}, \mathsf{role}])$ and $(\mathcal{P}, ssid, [\mathsf{Key} = k_1 | k_2])$
  $(m_2', k_3) \leftarrow \mathsf{Resp}(k_1, m_1')$, return m_2' from $\mathcal{P}$ to $\mathcal{A}$
  Store $(\mathcal{P}, ssid, [\mathsf{Key}, k_3, trans])$
  If Sim has queried Testpw to $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$ and obtained "correct guess": set $\mathsf{sKey}^* := G(k_2, k_3, trans)$; Else: set $\mathsf{sKey}^* := \perp$
  Send $(\mathsf{NewKey}, sid, \mathcal{P}, ssid, \mathsf{sKey}^*)$ to $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$
  If the record $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ with a **tested** flag: change the flag into **latetest**
on m_2' from $\mathcal{A}$ as a message from $\mathcal{CP}$ to $(\mathcal{P}, ssid)$:
  Retrieve the record $(\mathcal{P}, ssid, [\mathcal{CP}, \mathsf{role}])$, $(\mathcal{P}, ssid, [\mathsf{Key} = k_1 | k_2])$ and $(\mathcal{P}, ssid, st_1')$
  $k_3 \leftarrow \mathsf{Deri}(k_1, st_1', m_1', m_2')$
  Store $(\mathcal{P}, ssid, [\mathsf{Key}, k_3, trans])$
  If Sim has queried Testpw to $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$ and obtained "correct guess": set $\mathsf{sKey}^* := G(k_2, k_3, trans)$; Else: set $\mathsf{sKey}^* := \perp$
  Send $(\mathsf{NewKey}, sid, \mathcal{P}, ssid, \mathsf{sKey}^*)$ to $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$
  If the record $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ has a **tested** flag: change the flag into **latetest**
Random Oracle
on $G(k_2, k_3, trans)$ from A:
  If $\exists (k_2, k_3, trans, \mathsf{sKey}) \in \mathcal{L}_G$: return sKey
  $\mathsf{sKey} \leftarrow_{\$} \mathcal{K}$, $\mathcal{L}_G := \mathcal{L}_G \cup \{(k_2, k_3, trans, \mathsf{sKey})\}$, return sKey

Fig. 12: Simulator Sim for Theorem 3.

B.3 Description of Simulator in Theorem 4

Initialization
Sim invokes $(\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}$, $\mathsf{pp}' \leftarrow \mathsf{Setup}$, outputs $\mathsf{crs} := (\mathsf{pp}, \mathsf{pp}', H, G)$ and stores td
PAKE Sessions
on $(\mathsf{NewSession}, sid, \mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ from $\mathcal{F}_{\mathsf{PAKE}}$:
  If $\mathsf{role} = \mathsf{Initiator}$: $(m, st) \leftarrow \mathsf{SimInit}(\mathsf{pp})$ Else: $(m, st) \leftarrow \mathsf{SimResp}(\mathsf{pp})$
  Store the record $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role}, m, st)$, and output m to $\mathcal{A}$
on m' from $\mathcal{A}$ as a message from $\mathcal{CP}$ to $(\mathcal{P}, ssid)$:
  Retrieve the record $(\mathcal{P}, ssid, [\mathcal{CP}, \mathsf{role}, m, st])$
  Output $(\mathsf{NewSession2}, sid, \mathcal{P}, ssid, \mathcal{CP}, \mathsf{role})$ as a message from (the underlying) $\mathcal{F}_{\mathsf{PAKE2}}$ to $\mathcal{A}$
  If $\mathsf{role} = \mathsf{Initiator}$: $(m_1, m_2) := (m, m')$ Else: $(m_1, m_2) := (m', m)$
  Create a **fresh** record $(\mathcal{P}, ssid, \mathcal{CP}, \mathsf{role}, m_1, m_2, st)$
Ideal Functionality $\mathcal{F}_{\mathsf{PAKE2}}$
on $(\mathsf{Testpw2}, sid, \mathcal{P}, ssid, k_1)$ from $\mathcal{A}$:
  Retrieve the record $(\mathcal{P}, ssid, [\mathcal{CP}, \mathsf{role}, m_1, m_2, st])$ and ignore the query if the record is not **fresh**
  Mark the record **interrupted**, $(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w}) \leftarrow \mathcal{S}.E(k_1)$
  Send $(\mathsf{Testpw}, sid, \mathcal{P}, ssid, \hat{pw})$ to $\mathcal{F}_{\mathsf{PAKE}}$
  If $\mathcal{F}_{\mathsf{PAKE}}$ returns "wrong guess" to Sim: return "wrong guess" to $\mathcal{A}$
  If $(\hat{m}_1, \hat{m}_2) \neq (m_1, m_2)$: return "wrong guess" to $\mathcal{A}$
  If $\mathsf{role} = \mathsf{Initiator} \wedge \hat{w} \neq \mathsf{SimComp_i}(td, \hat{pw}, st, m_1, m_2)$: return "wrong guess" to $\mathcal{A}$
  If $\mathsf{role} = \mathsf{Responder} \wedge \hat{w} \neq \mathsf{SimComp_r}(td, \hat{pw}, st, m_1, m_2)$: return "wrong guess" to $\mathcal{A}$
  Mark the record as **compromised**, compute $k_1|k_2 := \mathcal{S}.RO(\hat{pw}, \hat{m}_1, \hat{m}_2, \hat{w})$, store a record $(\mathcal{P}, ssid, k_1, k_2)$
  Return "correct guess" to $\mathcal{A}$
on $(\mathsf{NewKey2}, sid, \mathcal{P}, ssid, \mathsf{Key}^*)$ from $\mathcal{A}$:
  If there is a record $(\mathcal{P}, ssid, [\mathcal{CP}, \mathsf{role}, m, st])$ not marked **completed**:
    If the record is **compromised**:
      Retrieve the record $(\mathcal{P}, ssid, [k_1, k_2])$, send $(\mathsf{NewKey}, sid, \mathcal{P}, ssid, G(k_2, k_3 = \mathsf{Key}^*, trans))$ to $\mathcal{F}_{\mathsf{PAKE}}$
  In all other cases, send $(\mathsf{NewKey}, sid, \mathcal{P}, ssid, \perp)$ to $\mathcal{F}_{\mathsf{PAKE}}$
Random Oracle
on $H(m)$ from A:
  $|Y\rangle \leftarrow \mathcal{S}.RO(m)$, return $|Y\rangle$

Fig. 13: Simulator Sim for Theorem 4.

C Proof of Theorem 3

We will prove it in the $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$-hybrid model, where the underlying PAKE_1 protocol is replaced by $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$, as shown in Fig. 14.

Game G_0(real world). This is the real world experiment $\mathbf{Real}_{\mathcal{Z},\mathcal{A}}^{\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}}$. In this game, $\mathcal{Z}$ initializes a password for each party instance, sees the interactions among parties, ideal functionality $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ and the adversary $\mathcal{A}$.

Note that in G_0, the adversary $\mathcal{A}$ can use Testpw1 and NewKey1 interfaces of the underlying $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ to implement active attacks.

We have

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}^{\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}} \Rightarrow 1\right].$$

Game G_1(simulations for parties with pw). In this game, we introduce a simulator Sim who *additionally knows passwords* of all parties. Sim generates $\mathsf{crs} := (\mathsf{pp}, \mathsf{pp}', H, G)$ with $(\mathsf{pp}, td) \leftarrow \mathsf{DHSetup}$ and $\mathsf{pp}' \leftarrow \mathsf{Setup}$ and keeps

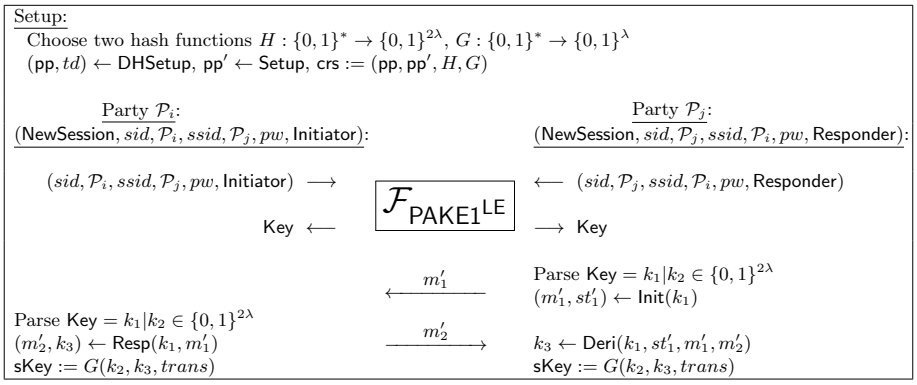

Fig. 14: HPAKE in $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$-hybrid model.

td. Then it simulates the ideal functionality $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ and parties to generate PAKE_1's session key Key, transcript m_1', m_2' and the session key sKey of HPAKE, just like G_0. With the knowledge of *passwords*, the simulations are perfect.

Moreover, Sim also simulates the random oracle G by maintaining a list $\mathcal{L}_G$. By the ideal functionality of random oracles, Sim's simulation for G is also perfect. So we have

$$\Pr[\mathsf{G}_1 \Rightarrow 1] = \Pr[\mathsf{G}_0 \Rightarrow 1].$$

Note that given PAKE_1's session key $\mathsf{Key} = k_1|k_2$, Sim can always invoke $(m_2', k_3) \leftarrow \mathsf{Resp}(k_1, m_1')$ for initiator instance, and invoke $(m_1', st_1') \leftarrow \mathsf{Init}(k_1))$ and $k_3 \leftarrow \mathsf{Deri}(k_1, st_1', m_1', m_2')$ for responder instance. Therefore, k_3, m_1' and m_2' can be properly simulated as long as $\mathsf{Key} = k_1|k_2$ is.

Game G_2(simulating session key sKey in case of no successful active attacks.) G_2 is the same as G_1, except for Sim's generation of session keys.

- For an initiator instance $(\mathcal{P}, ssid)$, upon the generation of session key sKey, if the underlying $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ (simulated by Sim) did not ever return "correct guess" before, then Sim generates sKey via $\mathsf{sKey} \leftarrow_{\$} \mathcal{K}$ rather than $\mathsf{sKey} := G(k_2, k_3, trans)$. If later $\mathcal{A}$ queries $(\mathsf{LateTestPwd1}, sid, ssid, \mathcal{P}, pw)$ with correct password pw, then Sim returns the simulated PAKE_1's session key $\mathsf{Key} = k_1|k_2$ to $\mathcal{A}$, and reprograms $G(k_2, k_3, trans) := \mathsf{sKey}$ for consistence, where k_3 is generated by Sim with algorithm Resp for $(\mathcal{P}, ssid)$.
- For a responder instance $(\mathcal{P}_j, ssid)$, if there exists an initiator instance $([\mathcal{P}_i], ssid)$ with session key sKey' s.t. they share the same Key and transcripts, then Sim assigns $\mathsf{sKey} := \mathsf{sKey}'$ rather than $\mathsf{sKey} := G(k_2, k_3, trans)$. This change is conceptional due to the correctness of HPAKE. Otherwise, $(\mathcal{P}_j, ssid)$ generates sKey just like the initiator instance shown before.

According to the specification of $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$, $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ did not ever reply "correct guess" implies that $\mathcal{A}$ did not ever issue $\mathsf{Testpw1}$ query with the correct password (i.e.,$\mathcal{A}$'s active attack is not successful), then $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ must have sampled a uniform $\mathsf{Key} = k_1|k_2$ which is independent of $\mathcal{A}$'s view. Then the

uniformity of $k_1|k_2$ leads to the uniformity of $\mathsf{sKey} := G(k_2, k_3, trans)$ unless $\mathcal{A}$ has already queried $G(k_2, k_3, trans)$ before the generation of sKey. If $\mathcal{A}$ later queries $(\mathsf{LateTestPwd1}, sid, ssid, \mathcal{P}, pw)$ with a correct pw, $\mathcal{A}$ will obtain $\mathsf{Key} = k_1|k_2$ from $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ (simulated by Sim). In this case, Sim will reprogram $G(k_2, k_3, trans) := \mathsf{sKey}$ to keep consistency. Therefore, G_2 is the same as G_1 unless $\mathcal{A}$ has already queried $G(k_2, k_3, trans)$ before the generation of sKey, which happens with probability at most $q \cdot 2^{-\lambda}$. So we have

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \frac{q}{2^\lambda},$$

where q is the number of hash queries from $\mathcal{A}$.

We stress that Sim still uses passwords to decide whether $\mathcal{A}$'s active attack is successful. If not, sKey is randomly chosen and consistency is kept between initiator and responder as did in G_2; If yes, Sim still computes $\mathsf{sKey} := G(k_2, k_3, trans)$ as did in G_1.

Game G_3(Simulating $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ and sKey with $\mathcal{F}^{\mathsf{leaky}}_{\mathsf{PAKE}^{\mathsf{LE}}}$). Recall that in G_2, Sim still uses passwords to simulate $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$. It also uses passwords to decide whether $\mathcal{A}$'s active attack is successful or not, so as to generate sKey in different way. In G_3, Sim invokes interfaces of $\mathcal{F}^{\mathsf{leaky}}_{\mathsf{PAKE}^{\mathsf{LE}}}$ (in its ideal world) to simulate $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ and the generation of session key sKey without passwords.

- For $\mathcal{A}$'s query $(\mathsf{Testpw1}, sid, \mathcal{P}, ssid, pw')$ to $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$, Sim just forwards query $(\mathsf{Testpw}, sid, \mathcal{P}, ssid, pw')$ to $\mathcal{F}^{\mathsf{leaky}}_{\mathsf{PAKE}^{\mathsf{LE}}}$ and returns the answer of $\mathcal{F}^{\mathsf{leaky}}_{\mathsf{PAKE}^{\mathsf{LE}}}$ to $\mathcal{A}$.
- For $\mathcal{A}$'s query $(\mathsf{RegisterTest1}, sid, \mathcal{P}, ssid)$ to $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$, if $\mathcal{A}$ did not ever ask $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ for $(\mathsf{LateTestPwd1}, sid, \mathcal{P}, ssid, pw')$ before the generation of sKey for $(\mathcal{P}, ssid)$, Sim issues $(\mathsf{RegisterTest}, sid, \mathcal{P}, ssid)$ to $\mathcal{F}^{\mathsf{leaky}}_{\mathsf{PAKE}^{\mathsf{LE}}}$.
- For $\mathcal{A}$'s query $(\mathsf{NewKey1}, sid, ssid, \mathcal{P}, \mathsf{Key}^*)$ to $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$, if $\mathcal{A}$ has ever queried $(\mathsf{Testpw1}, sid, \mathcal{P}, ssid, pw')$ to $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ (simulated by Sim) and obtained "correct guess", Sim returns $\mathsf{Key} := \mathsf{Key}^*$. If there exists an instance $([\mathcal{CP}], ssid)$ with Key' and $\mathcal{A}$ has never queried $\mathsf{Testpw1}$ or $\mathsf{RegisterTest1}$ on both $(\mathcal{P}, ssid)$ and $(\mathcal{CP}, ssid)$ to $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$, Sim returns $\mathsf{Key} := \mathsf{Key}'$. In other cases, Sim returns $\mathsf{Key} \leftarrow_{\$} \{0,1\}^{2\lambda}$.
- For $\mathcal{A}$'s query $(\mathsf{LateTestPwd1}, sid, ssid, \mathcal{P}, pw')$ to $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$, Sim considers the following two cases.
  1. The session key sKey is not generated for instance $(\mathcal{P}, ssid)$ yet. In this case, Sim just issues $(\mathsf{Testpw}, sid, \mathcal{P}, ssid, pw')$ query to $\mathcal{F}^{\mathsf{leaky}}_{\mathsf{PAKE}^{\mathsf{LE}}}$. If $\mathcal{F}^{\mathsf{leaky}}_{\mathsf{PAKE}^{\mathsf{LE}}}$ returns "correct guess", then Sim returns its simulated $\mathsf{Key} = k_1|k_2$ to $\mathcal{A}$. Otherwise, Sim samples and returns $\mathsf{Key}' \leftarrow_{\$} \{0,1\}^{2\lambda}$ to $\mathcal{A}$.
  2. The session key sKey has been generated for instance $(\mathcal{P}, ssid)$. In this case, Sim issues $(\mathsf{LateTestPwd}, sid, ssid, \mathcal{P}, pw')$ to $\mathcal{F}^{\mathsf{leaky}}_{\mathsf{PAKE}^{\mathsf{LE}}}$. If $\mathcal{F}^{\mathsf{leaky}}_{\mathsf{PAKE}^{\mathsf{LE}}}$ returns sKey and "correct guess", then Sim returns its simulated $\mathsf{Key} = k_1|k_2$ to $\mathcal{A}$ and reprograms $G(k_2, k_3, trans) := \mathsf{sKey}$. Otherwise, Sim samples $\mathsf{Key}' \leftarrow_{\$} \{0,1\}^{2\lambda}$ and returns Key' to $\mathcal{A}$.

– For the generation of session key sKey for instance $(\mathcal{P}, ssid)$, Sim issues query $(\mathsf{NewKey}, sid, \mathcal{P}, ssid, \mathsf{sKey}^*)$ to $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$, where $\mathsf{sKey}^* := G(k_2, k_3, trans)$ if Sim obtains "correct guess" from Testpw query, and $\mathsf{sKey}^* = \perp$ otherwise.

According to the specification of $\mathcal{F}_{\mathsf{PAKE}^{\mathsf{LE}}}^{\mathsf{leaky}}$, G_3 is a concept change of G_2, so

$$\Pr[\mathsf{G}_3 \Rightarrow 1] = \Pr[\mathsf{G}_2 \Rightarrow 1].$$

Now Sim has simulated $\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}$ and the generation of session keys $\mathsf{Key}, \mathsf{sKey}$ without password. Recall that m_1', m_2' can be seen as a (randomized) function of $\mathsf{Key} = k_1|k_2$. So the simulations of m_1', m_2' do not need passwords as well. Now that Sim completely gets rid of pw in the simulation, G_3 is exactly $\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}[\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}]}$. The full description of simulator Sim is shown in Fig. 12 in Appendix B.2. Therefore,

$$\Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}[\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}]} \Rightarrow 1\right] = \Pr[\mathsf{G}_3 \Rightarrow 1].$$

Finally, by combining all the statements across G_0-G_3, we know that

$$\left|\Pr\left[\mathbf{Real}_{\mathcal{Z},\mathcal{A}}^{\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}} \Rightarrow 1\right] - \Pr\left[\mathbf{Ideal}_{\mathcal{Z},\mathsf{Sim}[\mathcal{F}_{\mathsf{PAKE1}^{\mathsf{LE}}}]} \Rightarrow 1\right]\right| \leq \frac{q}{2^\lambda}.$$

□

D Instantiations of Full DH-type PAKE

The required computational assumptions in this section are shown in Appendix A.3.

D.1 DH-type PAKE: SPAKE2

SPAKE2 was proposed in [7]. We recall SPAKE2 in Fig. 15. The folllowing lemma shows that it is a full DH-type PAKE.

Lemma 2. *SPAKE2 is a full DH-type PAKE.*

Proof. We prove that SPAKE2 has correctness and the corresponding properties.

The correctness follows by the fact that $\mathsf{Comp_i}(\mathsf{pp}, pw, st_1, m_1, m_2) = (Y/B^{pw})^u = V^u = g^{uv} = U^v = (X/A^{pw})^u = \mathsf{Comp_r}(\mathsf{pp}, pw, st_2, m_1, m_2)$.

① **One-Wayness of the DH-Key.** We show that $\mathsf{Adv}_{\mathsf{key}}^{\mathsf{ow}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbb{G},g,q}^{\mathsf{Gap\text{-}CDH}}(\mathcal{B}_{\mathsf{CDH}})$.

Suppose the reduction algorithm $\mathcal{B}_{\mathsf{CDH}}$ receives $(U = g^u, V = g^v)$ and a DDH oracle $\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)$, and its goal is to compute g^{uv}. $\mathcal{B}_{\mathsf{CDH}}$ randomly samples $\mathsf{pp} := (A, B) \leftarrow_\$ \mathbb{G}$ and $pw \leftarrow_\$ \mathcal{PW}$, and computes $m_1^* := U \cdot A^{pw}$, $m_2^* := V \cdot B^{pw}$. Finally, $\mathcal{B}_{\mathsf{CDH}}$ sends $(\mathsf{pp} = (A, B), pw, m_1^*, m_2^*)$ as the one-wayness challenge to $\mathcal{A}$. Besides, $\mathcal{B}_{\mathsf{CDH}}$ uses $\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)$ oracle to simulate oracles $\mathcal{O}_1(m_1^*, u, \cdot, \cdot, \cdot)$ and $\mathcal{O}_2(m_2^*, v, \cdot, \cdot, \cdot)$ for $\mathcal{A}$. The oracle simulations are perfect since $\mathcal{O}(m_1^*, st_1^* = u, pw', m_2', w') = [\mathsf{Comp_i}(pw', st_1^* = u, m_1^*, m_2') \stackrel{?}{=} w'] = [w' \stackrel{?}{=} (m_2'/B^{pw'})^u] =$

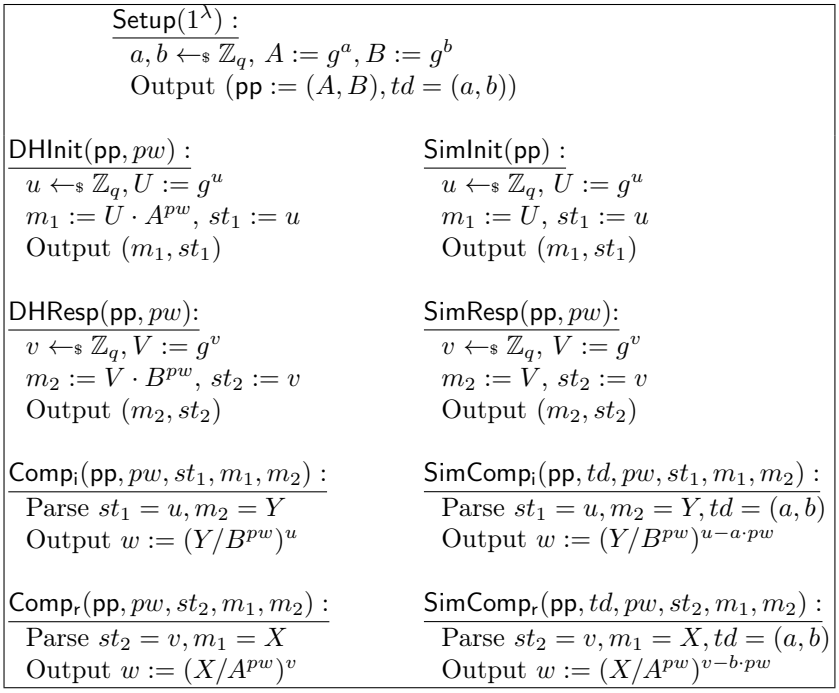

Fig. 15: The SPAKE2 protocol and the simulation algorithms.

$\mathsf{DDH}(g, U, m_2'/B^{pw'}, w')$ and $\mathcal{O}(m_2^*, st_2^* = v, pw', m_1', w') = [\mathsf{Comp}_{\mathsf{r}}(pw', st_2^* = v, m_1', m_2^*) \stackrel{?}{=} w'] = [w' \stackrel{?}{=} (m_1'/A^{pw'})^v] = \mathsf{DDH}(g, V, m_1'/A^{pw'}, w')$. If $\mathcal{A}$ outputs w, then $\mathcal{B}_{\mathsf{CDH}}$ submits w to its Gap-CDH challenger. Clearly, $\mathcal{B}_{\mathsf{CDH}}$ wins iff $\mathcal{A}$ wins.

② **Perfect Simulation for Initiator.** Recall that $m_1 = g^u \cdot A^{pw}$ for a uniform u when $(m_1, st_1) \leftarrow \mathsf{DHInit}(\mathsf{pp})$ and $m_1' = g^{u'}$ for a uniform u' when $(m_1', st_1') \leftarrow \mathsf{SimComp_i}(\mathsf{pp})$. Define $u := u' - a \cdot pw$, and then we have $m_1' = g^{u'} = g^u \cdot A^{pw}$, where the uniformity of u' implies the uniformity of u. So the distributions of w are the same when computed by u and $u' - a \cdot pw$. Recall that m_2 is independent of m_1 (and m_1') and w is determined by m_1, m_2 (resp. m_1', m_2). Therefore, we have $(m_1, m_2, w) \equiv (m_1', m_2, w)$.

③ **Perfect Simulation for Responder:** It is just a mirror symmetry of ②, so we omit it.

④ **Unique Password for Initiator:** We prove it with a security reduction. If there exists an adversary $\mathcal{A}$ breaking the "unique password for initiator" property, then we construct a reduction algorithm $\mathcal{B}_{init}$ that breaks the Gap-CDH assumption.

$\mathcal{B}_{init}$ receives (g^b, g^u) from the Gap-CDH challenger along with a $\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)$ oracle, and its goal is to compute g^{bu}. $\mathcal{B}_{init}$ sets $B := g^b$ and samples $a \leftarrow_{\$} \mathbb{Z}_q$ to compute $A := g^a$. Then $\mathcal{B}_{init}$ sends $(\mathsf{pp}^* = (A, B), m_1^* = g^u)$ to $\mathcal{A}$. Here $\mathcal{B}_{init}$ implicitly sets $td^* = (a, b)$. Next, $\mathcal{B}_{init}$ simulates oracles $\mathcal{O}_1(td^*, \cdot, \cdot, \cdot, \cdot, \cdot), \mathcal{O}_2(td^*, \cdot, \cdot, \cdot)$ and $\mathcal{O}_1^*(td^*, st_1^*, m_1^*, \cdot, \cdot, \cdot)$ in the following way.

- $\mathcal{O}_1(td^*, pw, st_1, m_1, m_2, w) = [w \stackrel{?}{=} (m_2/B^{pw})^{st_1}] = \mathsf{DDH}(g, m_1 = g^{st_1}, m_2/B^{pw}, w)$;
- $\mathcal{O}_2(td^*, pw, st_2, m_1, m_2, w) = [w \stackrel{?}{=} (m_1/A^{pw})^{st_2}] = \mathsf{DDH}(g, m_2 = g^{st_2}, m_1/A^{pw}, w)$;
- $\mathcal{O}_1^*(td^*, st_1^* = u, m_1^* = g^u, pw, m_2, w) = [w \stackrel{?}{=} (m_2/B^{pw})^u] = \mathsf{DDH}(g, m_1^*, m_2/B^{pw}, w)$.

Finally, if $\mathcal{A}$ outputs (pw, pw', m_2, w, w') and wins the game, then it holds that $pw \neq pw'$, $w = (m_2/B^{pw})^u$ and $w' = (m_2/B^{pw'})^u$. Therefore, $w/w' = (B^{pw'-pw})^u = (g^{bu})^{pw'-pw}$, which means $g^{bu} = (w/w')^{\frac{1}{pw'-pw}}$. So $\mathcal{B}_{init}$ just submits $(w/w')^{\frac{1}{pw'-pw}}$ to the Gap-CDH challenger and wins the game.

Therefore, $\mathsf{Adv}^{\mathsf{Init}}_{\mathsf{Unipw}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Gap\text{-}CDH}}_{\mathbb{G},g,q}(\mathcal{B}_{init})$.

⑤ **Unique Password for Responder:** It is just a mirror symmetry of ④, so we omit it. □

D.2 DH-type PAKE: (Crs)X-GA-PAKE

The (Crs)X-GA-PAKE protocol was proposed in [4,25]. We review it in Fig. 16 and show that it is a full DH-type PAKE in the following lemma.

Lemma 3. *The protocol CrsX-GA-PAKE is a full DH-type PAKE.*

Proof. We prove that CrsX-GA-PAKE has correctness and the corresponding properties.

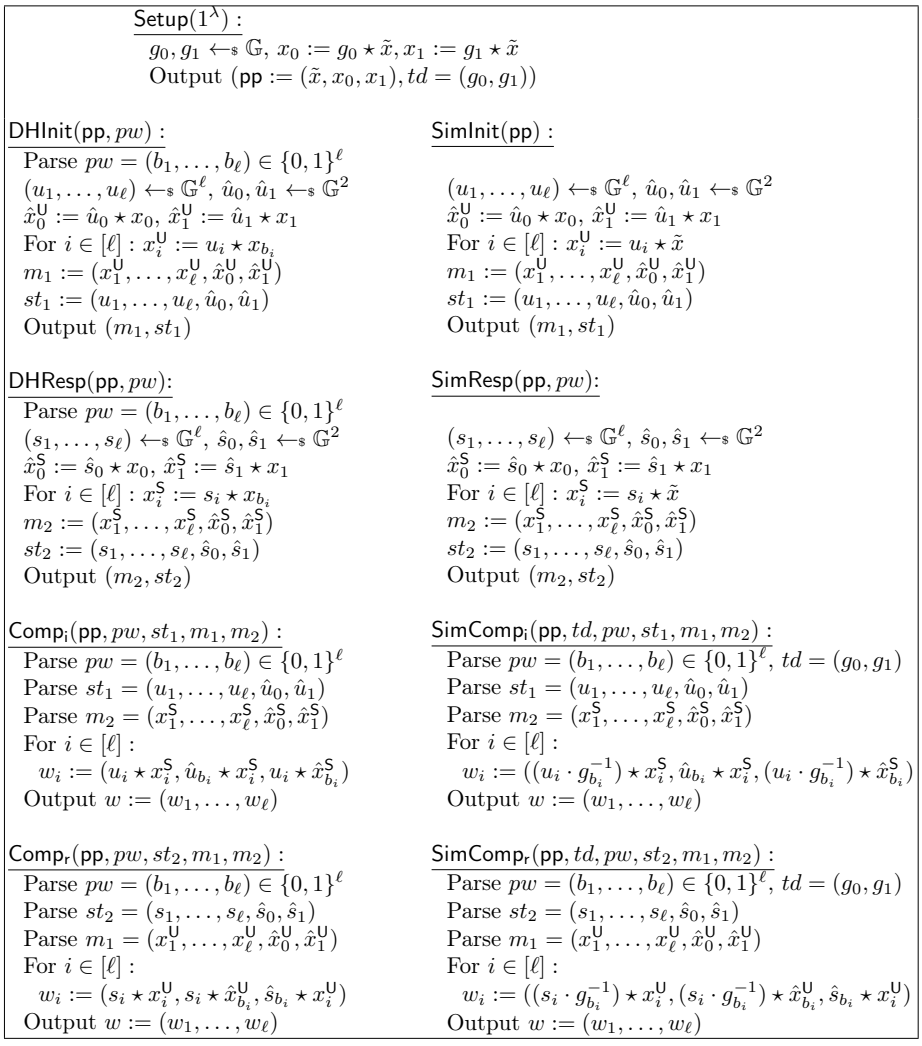

Fig. 16: The (Crs)X-GA-PAKE protocol and the simulation algorithms.

The correctness follows by

$$\begin{aligned}
\mathsf{Comp_i}(\mathsf{pp}, pw, st_1, m_1, m_2) &= \{(u_i \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \star x_i^{\mathsf{S}}, u_i \star \hat{x}_{b_i}^{\mathsf{S}})\}_{i\in[\ell]} \\
&= \{((u_i s_i) \star x_{b_i}, (\hat{u}_{b_i} s_i) \star x_{b_i}, (u_i \hat{s}_{b_i}) \star x_{b_i})\}_{i\in[\ell]} \\
&= \{(s_i \star (u_i \star x_{b_i}), s_i \star (\hat{u}_{b_i} \star x_{b_i}), \hat{s}_{b_i} \star (u_i \star x_{b_i}))\}_{i\in[\ell]} \\
&= \{(s_i \star x_i^{\mathsf{U}}, s_i \star \hat{x}_{b_i}^{\mathsf{U}}, \hat{s}_{b_i} \star x_i^{\mathsf{U}})\}_{i\in[\ell]} \\
&= \mathsf{Comp_r}(\mathsf{pp}, pw, st_2, m_1, m_2).
\end{aligned}$$

① **One-Wayness of DH-Key:** We show that $\mathsf{Adv}^{\mathsf{ow}}_{\mathsf{key}}(\mathcal{A}) \le \mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{REGA}}(\mathcal{B}_{\mathsf{GA\text{-}GapCDH}})$.

Suppose that the reduction algorithm $\mathcal{B}_{\mathsf{GA\text{-}GapCDH}}$ receives $U = u\star\tilde{x}$, $S = s\star\tilde{x}$ and an oracle $\mathsf{GA\text{-}DDH}(\cdot,\cdot,\cdot,\cdot)$. Its goal is to compute $Z = (us)\star\tilde{x}$ to break the GA-GapCDH assumption.

$\mathcal{B}_{\mathsf{GA\text{-}GapCDH}}$ generates $x_0 := g_0 \star \tilde{x}$, $x_1 := g_1 \star \tilde{x}$ with $g_0, g_1 \leftarrow_{\$} \mathbb{G}$, and sets $\mathsf{pp} := (x_0, x_1)$. Suppose $\mathcal{A}$ outputs $pw = (b_1, \ldots, b_\ell)$ after receiving pp. Next, $\mathcal{B}_{\mathsf{GA\text{-}GapCDH}}$ generates $m_1^* = (x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}), m_2^* = (x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}})$ using the same algorithms as DHInit and DHResp, except that it sets $x_1^{\mathsf{U}} := U$ and $x_1^{\mathsf{S}} := S$. $\mathcal{B}_{\mathsf{GA\text{-}GapCDH}}$ sets $st_1^* = (?, u_2, \ldots, u_\ell, \hat{u}_0, \hat{u}_1)$ and $st_2^* = (?, s_2, \ldots, s_\ell, \hat{s}_0, \hat{s}_1)$, where "?" in st_1^* is implicitly set as $ug_{b_1}^{-1}$ and "?" in st_2^* as $sg_{b_1}^{-1}$. Then it sends $(\mathsf{pp}, pw, m_1^*, m_2^*)$ to $\mathcal{A}$. For any query (pw', m_2', w') to oracle $\mathcal{O}_1(m_1^*, st_1^*, \cdot, \cdot, \cdot)$, we know that $\mathcal{O}_1(m_1^*, st_1^*, pw', m_2', w') = [\mathsf{Comp_i}(pw', st_1^*, m_1^*, m_2') \stackrel{?}{=} w']$. Suppose $w'' := (w_1'', \ldots, w_\ell'') = \mathsf{Comp_i}(pw', st_1^*, m_1^*, m_2')$. Parse $pw' = (b_1', \ldots, b_\ell')$, $w' = (w_1', \ldots, w_\ell')$ and $m_2' = (x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}})$. Then $x_1^{\mathsf{U}} = U = u\star\tilde{x} = (ug_{b_1'}^{-1}) \star x_{b_1'}$, $x_1^{\mathsf{S}} = S = s\star\tilde{x} = (sg_{b_1'}^{-1}) \star x_{b_1'}$ are simulated perfectly due to the uniformity of u and s. Further parse $w_1' = (w_{1,1}', w_{1,2}', w_{1,2}')$ and $w_1'' = (w_{1,1}'', w_{1,2}'', w_{1,2}'')$. Then $\mathcal{B}_{\mathsf{GA\text{-}GapCDH}}$ can use st_1^* to compute $w_{1,2}'', w_2'', \ldots, w_\ell''$. Now $\mathcal{B}$ tests $[w' \stackrel{?}{=} w'']$: it first checks that whether $w_{1,2}' = w_{1,2}'', w_2' = w_2'', \ldots, w_\ell' = w_\ell''$. If yes it will further checks whether $w_{1,1}' = w_{1,1}''$ and $w_{1,3}' = w_{1,3}''$ by resorting to its own oracle $\mathsf{GA\text{-}DDH}(x_{b_1'}, U, x_1^{\mathsf{S}}, w_{1,1}')$ and $\mathsf{GA\text{-}DDH}(x_{b_1'}, U, \hat{x}_{b_1'}^{\mathsf{S}}, w_{1,3}')$. Recall that $U = (ug_{b_1'}^{-1}) \star x_{b_1'}$ and $w_{1,1}'' = (ug_{b_1'}^{-1}) \star x_1^{\mathsf{S}}$ and $w_{1,3}'' = (ug_{b_1'}^{-1}) \star \hat{x}_{b_1'}^{\mathsf{S}}$. Therefore, $w_{1,1}' = w_{1,1}''$ and $w_{1,3}' = w_{1,3}''$ iff both $\mathsf{GA\text{-}DDH}(x_{b_1'}, U, x_1^{\mathsf{S}}, w_{1,1}')$ and $\mathsf{GA\text{-}DDH}(x_{b_1'}, U, \hat{x}_{b_1'}^{\mathsf{S}}, w_{1,3}')$ output 1. In this way, $\mathcal{B}_{\mathsf{GA\text{-}GapCDH}}$ perfectly simulates oracle $\mathcal{O}_1(m_1^*, st_1^*, \cdot, \cdot, \cdot)$. A similar argument can show that $\mathcal{B}_{\mathsf{GA\text{-}GapCDH}}$ perfectly simulates oracle $\mathcal{O}_2(m_2^*, st_2^*, \cdot, \cdot, \cdot)$. This yields $\mathsf{Adv}^{\mathsf{ow}}_{\mathsf{key}}(\mathcal{A}) \le \mathsf{Adv}^{\mathsf{GA\text{-}GapCDH}}_{\mathsf{REGA}}(\mathcal{B}_{\mathsf{GA\text{-}GapCDH}})$.

If $\mathcal{A}$ outputs $w = (w_1, \ldots, w_\ell)$ with $w_1 = (z_1, z_2, z_3)$, then $\mathcal{B}_{\mathsf{GA\text{-}GapCDH}}$ returns $g_{b_1} \star z_1$ as its answer to the GA-GapCDH challenge. If $\mathcal{A}$ wins, then $\mathcal{B}_{\mathsf{GA\text{-}GapCDH}}$ also wins, since $z_1 = (ug_{b_1}^{-1}) \star x_1^{\mathsf{S}} = (ug_{b_1}^{-1}) \star (s\star\tilde{x})$ and $g_{b_1} \star z_1 = (us)\star\tilde{x}$.

② **Perfect Simulation for Initiator:** Suppose $\mathsf{pp} = (x_0 = g_0\star\tilde{x}, x_1 = g_1\star\tilde{x})$, $td = (g_0, g_1)$ and $pw = (b_1, \ldots, b_\ell) \in \{0,1\}^\ell$. For $m_1 = (x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}})$, DHInit and $\mathsf{SimInit}$ have the same way of computing $\hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}}$. As for x_i^{U}, DHInit computes $x_i^{\mathsf{U}} := u_i \star x_{b_i} = (u_i g_{b_i}) \star \tilde{x}$ with $u_i \leftarrow_{\$} \mathbb{G}$, while $\mathsf{SimInit}$ samples $u_i' \leftarrow_{\$} \mathbb{G}$ and computes $x_i^{\mathsf{U}} := u_i' \star \tilde{x}$. The uniformity of u_i and u_i' implies that x_i^{U} by DHInit is statistically identical to x_i^{U} by $\mathsf{SimInit}$. Moreover, $\mathsf{Comp_i}$ computes $w_i := (u_i \star x_i^{\mathsf{S}}, \hat{u}_{b_i} \star x_i^{\mathsf{S}}, u_i \star \hat{x}_{b_i}^{\mathsf{S}})$ while $\mathsf{SimComp_i}$ computes $w_i := ((u_i' \cdot g_{b_i}^{-1}) \star$

$x_i^{\mathsf{S}}, \hat{u}_{b_i} \star x_i^{\mathsf{S}}, (u_i' \cdot g_{b_i}^{-1}) \star \hat{x}_{b_i}^{\mathsf{S}})$. If we define $u_i := (u_i' \cdot g_{b_i}^{-1})$ for $\mathsf{SimComp_i}$, then u_i has the same distribution as u_i', and the computation of w_i by $\mathsf{SimComp_i}$ is exactly the same as that by $\mathsf{Comp_i}$. Therefore, $\mathsf{SimInit}$ and $\mathsf{SimComp_i}$ perfectly simulate DHInit and $\mathsf{Comp_i}$.

③ **Perfect Simulation for Responder:** The argument is similiar to ②.

④ **Unique Password for Initiator:** We show that if there exists an adversary $\mathcal{A}$ breaking the "unique password for initiator" property, then we can construct an adversary $\mathcal{B}_{init}$ to break the DSim-GA-GapCDH assumption.

$\mathcal{B}_{init}$ receives four set elements (x_0, x_1, y_0, y_1) as its challenge, and has access to a $\mathsf{GA\text{-}DDH}(\cdot,\cdot,\cdot,\cdot)$ oracle. Suppose $(x_0, x_1, y_0, y_1) = (g_0 \star \tilde{x}, g_1 \star \tilde{x}, h_0 \star \tilde{x}, h_1 \star \tilde{x})$. Then the goal of $\mathcal{B}_{init}$ is to compute five set elements (z, z_0, z_1, z_2, z_3) s.t. $(z_0, z_1, z_2, z_3) = ((g_0^{-1}h_0) \star z, (g_0^{-1}h_1) \star z, (g_1^{-1}h_0) \star z, (g_1^{-1}h_1) \star z)$. To this end, $\mathcal{B}_{init}$ sets $\mathsf{pp}^* := (x_0, x_1)$ and generates $m_1^* = (x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}})$ in the following way. For each $i \in [\ell]$, $\mathcal{B}_{init}$ randomly samples $u_i^* \leftarrow_{\$} \mathbb{G}$ and sets $x_i^{*\mathsf{U}} := u_i^* \star y_0$. It also samples $\hat{u}_0^*, \hat{u}_1^* \leftarrow_{\$} \mathbb{G}$ and sets $\hat{x}_0^{*\mathsf{U}} := \hat{u}_0^* \star y_1$ and $\hat{x}_1^{*\mathsf{U}} := \hat{u}_1^* \star y_1$. Then $\mathcal{B}_{init}$ sends $(\mathsf{pp}^* = (x_0, x_1), m_1^* = (x_1^{*\mathsf{U}}, \ldots, x_\ell^{*\mathsf{U}}, \hat{x}_0^{*\mathsf{U}}, \hat{x}_1^{*\mathsf{U}}))$ to $\mathcal{A}$. It holds that

$$x_i^{*\mathsf{U}} = u_i^* \star y_0 = (u_i^* h_0) \star \tilde{x},\ \hat{x}_0^{*\mathsf{U}} = \hat{u}_0^* \star y_1 = (\hat{u}_0^* h_1 g_0^{-1}) \star x_0,\ \hat{x}_1^{*\mathsf{U}} = \hat{u}_1^* \star y_1 = (\hat{u}_1^* h_1 g_1^{-1}) \star x_1.$$

Hence $\mathcal{B}_{init}$ implicitly sets $td^* = (g_0, g_1)$ and $st_1^* = (u_1^* h_0, \ldots, u_\ell^* h_0, \hat{u}_0^* h_1 g_0^{-1}, \hat{u}_1^* h_1 g_1^{-1})$. $\mathcal{B}_{init}$ can simulate oracles $\mathcal{O}_1(td^*, \cdots)$, $\mathcal{O}_2(td^*, \cdots)$, $\mathcal{O}_1^*(td^*, st_1^*, m_1^*, \cdots)$ in the following way.

- $\mathcal{O}_1(td^*, pw, st_1, m_1, m_2, w) : \mathcal{B}_{init}$ first parses $pw = (b_1, \ldots, b_\ell)$, $st_1 = (u_1, \ldots, u_\ell, \hat{u}_0, \hat{u}_1)$, $m_2 = (x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}})$, and $w = ((w_{1,1}, w_{1,2}, w_{1,3}), \ldots, (w_{\ell,1}, w_{\ell,2}, w_{\ell,3}))$. For each $i \in [\ell]$, $\mathcal{B}_{init}$ needs to check the following three conditions: $[w_{i,1} \stackrel{?}{=} (u_i \cdot g_{b_i}^{-1}) \star x_i^{\mathsf{S}}]$, $[w_{i,2} \stackrel{?}{=} \hat{u}_{b_i} \star x_i^{\mathsf{S}}]$ and $[w_{i,3} \stackrel{?}{=} (u_i \cdot g_{b_i}^{-1}) \star \hat{x}_{b_i}^{\mathsf{S}}]$. If all the three conditions are satisfied for each $i \in [\ell]$, $\mathcal{B}_{init}$ returns 1; otherwise, it returns 0. The second condition can be checked by $\hat{u}_{b_i}$ and x_i^{S} directly, while the first and third conditions can be checked by $\mathcal{B}_{init}$'s queries to oracle $\mathsf{GA\text{-}DDH}(\tilde{x}, u_i^{-1} \star x_{b_i}, w_{i,1}, x_i^{\mathsf{S}})$ and $\mathsf{GA\text{-}DDH}(\tilde{x}, u_i^{-1} \star x_{b_i}, w_{i,3}, \hat{x}_{b_i}^{\mathsf{S}})$ respectively. Clearly, the oracle simulation is perfect.
- $\mathcal{O}_2(td^*, pw, st_2, m_1, m_2, w) : \mathcal{B}_{init}$ first parses $pw = (b_1, \ldots, b_\ell)$, $st_2 = (s_1, \ldots, s_\ell, \hat{s}_0, \hat{s}_1)$, $m_1 = (x_1^{\mathsf{U}}, \ldots, x_\ell^{\mathsf{U}}, \hat{x}_0^{\mathsf{U}}, \hat{x}_1^{\mathsf{U}})$, and $w = ((w_{1,1}, w_{1,2}, w_{1,3}), \ldots, (w_{\ell,1}, w_{\ell,2}, w_{\ell,3}))$. For each $i \in [\ell]$, $\mathcal{B}_{init}$ needs to check the following three conditions: $[w_{i,1} \stackrel{?}{=} (s_i \cdot g_{b_i}^{-1}) \star x_i^{\mathsf{U}}]$, $[w_{i,2} \stackrel{?}{=} (s_i \cdot g_{b_i}^{-1}) \star \hat{x}_{b_i}^{\mathsf{U}}]$ and $[w_{i,3} \stackrel{?}{=} \hat{s}_{b_i} \star x_i^{\mathsf{U}}]$. If all the three conditions are satisfied for each $i \in [\ell]$, $\mathcal{B}_{init}$ returns 1; otherwise, it returns 0. The third condition can be checked by $\hat{s}_{b_i}$ and x_i^{U} directly, while the first and second conditions can be checked by $\mathcal{B}_{init}$'s queries to oracle $\mathsf{GA\text{-}DDH}(\tilde{x}, s_i^{-1} \star x_{b_i}, w_{i,1}, x_i^{\mathsf{U}})$ and $\mathsf{GA\text{-}DDH}(\tilde{x}, s_i^{-1} \star x_{b_i}, w_{i,2}, \hat{x}_{b_i}^{\mathsf{U}})$ respectively.
- $\mathcal{O}_1^*(td^*, st_1^*, m_1^*, pw, m_2, w) : \mathcal{B}_{init}$ first parses $m_1^* = (x_1^{*\mathsf{U}}, \ldots, x_\ell^{*\mathsf{U}}, \hat{x}_0^{*\mathsf{U}}, \hat{x}_1^{*\mathsf{U}})$, $pw = (b_1, \ldots, b_\ell)$, $m_2 = (x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}})$, and $w = ((w_{1,1}, w_{1,2}, w_{1,3}), \ldots, (w_{\ell,1}, w_{\ell,2}, w_{\ell,3}))$. Recall that $x_i^{*\mathsf{U}} = (u_i^* h_0) \star \tilde{x}, \hat{x}_0^{*\mathsf{U}} = (\hat{u}_0^* h_1 g_{b_i}^{-1}) \star x_{b_i},\ \hat{x}_1^{*\mathsf{U}} = (\hat{u}_1^* h_1 g_{b_i}^{-1}) \star x_{b_i}$. For each $i \in [\ell]$, $\mathcal{B}_{init}$ needs to check the following three conditions: $[w_{i,1} \stackrel{?}{=} (u_i^* h_0 \cdot g_{b_i}^{-1}) \star x_i^{\mathsf{S}}]$, $[w_{i,2} \stackrel{?}{=} (\hat{u}_{b_i}^* h_1 g_{b_i}^{-1}) \star x_i^{\mathsf{S}}]$ and $[w_{i,3} \stackrel{?}{=} (u_i^* h_0 \cdot$

$g_{b_i}^{-1}) \star \hat{x}_{b_i}^{\mathsf{S}}]$. If all the three conditions are satisfied for each $i \in [\ell]$, $\mathcal{B}_{init}$ returns 1; otherwise, it returns 0. The three conditions can be checked by $\mathcal{B}_{init}$'s queries to oracle $\mathsf{GA\text{-}DDH}(x_{b_i}, y_0, x_i^{\mathsf{S}}, u_i^{*-1} \star w_{i,1})$, $\mathsf{GA\text{-}DDH}(x_{b_i}, y_1, x_i^{\mathsf{S}}, \hat{u}_{b_i}^{*-1} \star w_{i,2})$ and $\mathsf{GA\text{-}DDH}(x_{b_i}, y_0, \hat{x}_{b_i}^{\mathsf{S}}, u_i^{*-1} \star w_{i,3})$ respectively, since $y_0 = h_0 \star \tilde{x} = (h_0 g_{b_i}^{-1} g_{b_i}) \star \tilde{x} = (h_0 g_{b_i}^{-1}) \star x_{b_i}$ and $y_1 = h_1 \star \tilde{x} = (h_1 g_{b_i}^{-1} g_{b_i}) \star \tilde{x} = (h_1 g_{b_i}^{-1}) \star x_{b_i}$.

Finally $\mathcal{A}$ outputs (pw, pw', m_2, w, w'). Suppose that $\mathcal{A}$ breaks the "④ unique password for initiator" property, i.e., $\mathcal{A}$ wins, then there must exist $i \in [\ell]$ such that pw and pw' are different in the i-th bit. W.l.o.g. we assume that i-th bit of pw is 0 and i-th bit of pw' is 1. Parse $m_2 = (x_1^{\mathsf{S}}, \ldots, x_\ell^{\mathsf{S}}, \hat{x}_0^{\mathsf{S}}, \hat{x}_1^{\mathsf{S}})$, $w = (w_1 = (w_{1,1}, w_{1,2}, w_{1,3}), \ldots, w_\ell = (w_{\ell,1}, w_{\ell,2}, w_{\ell,3}))$ and $w' = (w'_1 = (w'_{1,1}, w'_{1,2}, w'_{1,3}), \ldots, w'_\ell = (w'_{\ell,1}, w'_{\ell,2}, w'_{\ell,3}))$. $\mathcal{A}$ wins implies that $w_{i,1} = (u_i^* h_0 \cdot g_0^{-1}) \star x_i^{\mathsf{S}}$, $w_{i,2} = (\hat{u}_0^* h_1 g_0^{-1}) \star x_i^{\mathsf{S}}$, $w'_{i,1} = (u_i^* h_0 \cdot g_1^{-1}) \star x_i^{\mathsf{S}}$, $w'_{i,2} = (\hat{u}_1^* h_1 g_1^{-1}) \star x_i^{\mathsf{S}}$. Then, $\mathcal{B}_{init}$ submits $z := x_i^{\mathsf{S}}$, $z_0 := u_i^{*-1} \star w_{i,1}$, $z_1 := \hat{u}_0^{*-1} \star w_{i,2}$, $z_2 := u_i^{*-1} \star w'_{i,1}$, $z_3 := \hat{u}_1^{*-1} \star w'_{i,2}$ as it answer to its challenge. Clearly, $\mathcal{B}_{init}$ breaks the DSim-GA-GapCDH assumption as long as $\mathcal{A}$ wins. Thus we have $\mathsf{Adv}^{\mathsf{Init}}_{\mathsf{Unipw}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{DSim\text{-}GA\text{-}GapCDH}}_{\mathsf{REGA}}(\mathcal{B}_{init})$

⑤ **Unique Password for Responder:** The argument is similiar to ④. □

D.3 DH-type PAKE: TBPEKE

TBPEKE was proposed in [32]. We review it in Fig. 17 and show that it is a full DH-type PAKE in the following lemma.

Lemma 4. *TBPEKE is a full DH-type PAKE.*

Proof. We prove that TBPEKE has correctness and the corresponding properties.

The correctness follows by the fact that $\mathsf{Comp_i}(\mathsf{pp}, pw, st_1, m_1, m_2) = (Y)^x = (g_{pw})^{xy} = (X)^y = \mathsf{Comp_r}(\mathsf{pp}, pw, st_2, m_1, m_2)$.

① **One-Wayness of the DH-Key.** We show that $\mathsf{Adv}^{\mathsf{ow}}_{\mathsf{key}}(\mathcal{A}) \leq \mathsf{Adv}^{\mathsf{Gap\text{-}CDH}}_{\mathbb{G},g,q}(\mathcal{B}_{\mathsf{CDH}})$.

Suppose the reduction algorithm $\mathcal{B}_{\mathsf{CDH}}$ receives $(U = g^u, V = g^v)$ and a DDH oracle $\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)$, and its goal is to compute g^{uv}. $\mathcal{B}_{\mathsf{CDH}}$ randomly samples $\mathsf{pp} := (A = g^a, B = g^b) \leftarrow_{\$} \mathbb{G}$ and $pw \leftarrow_{\$} \mathcal{PW}$. Define $z := a + b \cdot \mathsf{P}(pw)$. It holds that $g_{pw} = A \cdot B^{\mathsf{P}(pw)} = g^z$. B_{CDH} randomly samples $\alpha, \beta \leftarrow_{\$} \mathbb{Z}_q$ and computes $m_1^* := U^{\alpha \cdot z} = (g_{pw})^{u\alpha}$, $m_2^* := V^{\beta \cdot z} = (g_{pw})^{v\beta}$. Finally, $\mathcal{B}_{\mathsf{CDH}}$ sends $(\mathsf{pp} = (A, B), pw, m_1^*, m_2^*)$ as the one-wayness challenge to $\mathcal{A}$. Besides, $\mathcal{B}_{\mathsf{CDH}}$ uses $\mathsf{DDH}(\cdot,\cdot,\cdot)$ oracle to simulate oracles $\mathcal{O}_1(m_1^*, u\alpha, \cdot, \cdot, \cdot)$ and $\mathcal{O}_2(m_2^*, v\beta, \cdot, \cdot, \cdot)$ for $\mathcal{A}$. The oracle simulations are perfect since $\mathcal{O}(m_1^*, st_1^* = u\alpha, pw', m_2', w') = [\mathsf{Comp_i}(pw', st_1^* = u\alpha, m_1^*, m_2') \stackrel{?}{=} w'] = [w' \stackrel{?}{=} (m_2')^{u\alpha}] = \mathsf{DDH}(g, U^\alpha, m_2', w')$ and $\mathcal{O}(m_2^*, st_2^* = v\beta, pw', m_1', w') = [\mathsf{Comp_r}(pw', st_2^* = v\beta, m_1', m_2^*) \stackrel{?}{=} w'] = [w' \stackrel{?}{=} (m_1')^{v\beta}] = \mathsf{DDH}(g, V^\beta, m_1', w')$. If $\mathcal{A}$ outputs w, then $\mathcal{B}_{\mathsf{CDH}}$ submits $w^{1/(\alpha\beta z)}$ to its Gap-CDH challenger. Clearly, $\mathcal{B}_{\mathsf{CDH}}$ wins iff $\mathcal{A}$ wins.

② **Perfect Simulation for Initiator.** For $(m_1, st_1) \leftarrow \mathsf{DHInit}(\mathsf{pp})$, we have $m_1 = (g_{pw})^x = g^{(a + \mathsf{P}(pw) \cdot b)x}$ for a uniform x, and for $(m_1', st_1') \leftarrow \mathsf{SimComp_i}(\mathsf{pp})$,

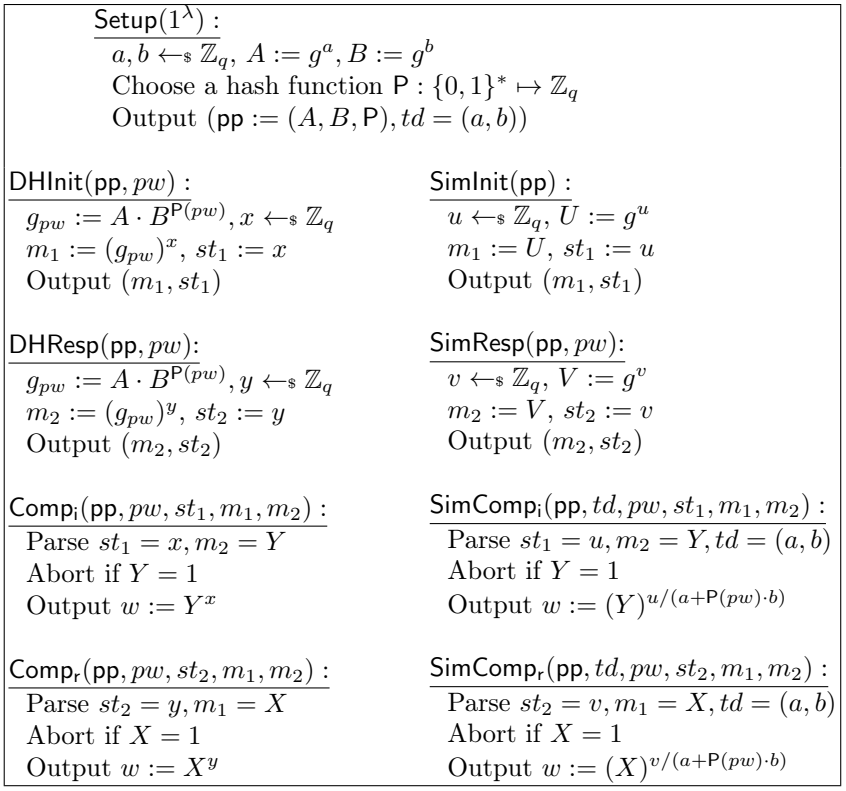

Fig. 17: The TBPEKE protocol and the simulation algorithms.

we have $m_1' = g^{x'}$ for a uniform x'. Define $x := x'/(a + \mathsf{P}(pw) \cdot b)$, and then we have $m_1' = g^{x'} = (g_{pw})^x$, where the uniformity of x' implies the uniformity of x. So the distributions of w are the same when computed by x and $x'/(a+\mathsf{P}(pw)\cdot b)$. Recall that m_2 is independent of m_1 (and m_1') and w is determined by m_1, m_2 (resp. m_1', m_2). Therefore, we have $(m_1, m_2, w) \equiv (m_1', m_2, w)$.

③ **Perfect Simulation for Responder:** It is just a mirror symmetry of ②, so we omit it.

④ **Unique Password for Initiator:** We prove it with a security reduction in the random oracle model. Suppose the hash function P works as a random oracle. If there exists an adversary $\mathcal{A}$ breaking the "unique password for initiator" property, then we construct a reduction algorithm $\mathcal{B}_{init}$ that breaks the Gap-SDH assumption.

$\mathcal{B}_{init}$ receives $(X = g^x, X^{y_1}, X^{y_2})$ from the Gap-SDH challenger along with a $\mathsf{DDH}(\cdot,\cdot,\cdot,\cdot)$ oracle, and its goal is to compute $(Y \neq 1, R = Y^{1/y_1}, S = Y^{1/y_2})$. $\mathcal{B}_{init}$ randomly chooses $i, j \in [q]$ and programs $\mathsf{P}(pw_i) := p_1$ and $\mathsf{P}(pw_j) := p_2$, where pw_i and pw_j is i-th and j-th hash queries from $\mathcal{A}$. Then $\mathcal{B}_{init}$ computes $B := (X^{y_1}/X^{y_2})^{1/(p_1-p_2)} = g^b$ and $A := X^{y_1}/B^{p_1} = g^a$. Note that $A \cdot B^{p_1} = X^{y_1}$ and $A \cdot B^{p_2} = X^{y_2}$. $\mathcal{B}_{init}$ sends $(\mathsf{pp}^* = (A, B), m_1^* = X)$ to $\mathcal{A}$. Here $\mathcal{B}_{init}$ implicitly sets $td^* = (a, b)$. Next, $\mathcal{B}_{init}$ simulates oracles $\mathcal{O}_1(td^*,\cdot,\cdot,\cdot,\cdot,\cdot), \mathcal{O}_2(td^*,\cdot,\cdot,\cdot)$ and $\mathcal{O}_1^*(td^*, st_1^*, m_1^*,\cdot,\cdot,\cdot)$ in the following way.

- $\mathcal{O}_1(td^*, pw, st_1, m_1, m_2, w) = [w \stackrel{?}{=} (m_2)^{\frac{st_1}{(a+\mathsf{P}(pw)\cdot b)}}] = [w^{a+\mathsf{P}(pw)\cdot b} \stackrel{?}{=} (m_2)^{st_1}]$
  $= \mathsf{DDH}(g, A \cdot B^{\mathsf{P}(pw)}, w, (m_2)^{st_1})$;
- $\mathcal{O}_2(td^*, pw, st_2, m_1, m_2, w) = [w \stackrel{?}{=} (m_1)^{\frac{st_2}{(a+\mathsf{P}(pw)\cdot b)}}] = \mathsf{DDH}(g, A \cdot B^{\mathsf{P}(pw)}, w, (m_1)^{st_2})$;
- $\mathcal{O}_1^*(td^*, st_1^* = x, m_1^* = X, pw, m_2, w) = [w \stackrel{?}{=} (m_2)^{\frac{x}{a+\mathsf{P}(pw)\cdot b}}]$
  $= [w^{\frac{a+\mathsf{P}(pw)\cdot b}{x}} \stackrel{?}{=} m_2] = \mathsf{DDH}(X, w, A \cdot B^{\mathsf{P}(pw)}, m_2)$.

Finally, $\mathcal{A}$ outputs (pw, pw', m_2, w, w'), then $\mathcal{B}_{init}$ submits (m_2, w, w') to the Gap-SDH challenger.

If $\mathcal{A}$ wins the game and $\mathsf{P}(pw) = p_1$ and $\mathsf{P}(pw') = p_2$, then $w = (m_2)^{\frac{x}{a+b\cdot p_1}} = (m_2)^{1/y_1}$ and $w' = (m_2)^{\frac{x}{a+b\cdot p_2}} = (m_2)^{1/y_2}$, so $\mathcal{B}$ wins in this case. The probability that pw is the i-th hash query and pw' is the j-th hash query is $1/q^2$. Therefore, $\mathcal{B}$ wins with probability $1/q^2$ as long as $\mathcal{A}$ wins. So we have $\mathsf{Adv}^{\mathsf{Init}}_{\mathsf{Unipw}}(\mathcal{A}) \leq q^2 \cdot \mathsf{Adv}^{\mathsf{Gap\text{-}SDH}}_{\mathbb{G},g,q}(\mathcal{B}_{init})$.

⑤ **Unique Password for Responder:** It is just a mirror symmetry of ④, so we omit it. □

Table of Contents