Maximizing the Utility of Cryptographic Setups: Secure PAKEs, with either functional RO or CRS*

Yuting Xiao† Rui Zhang‡ Hong-Sheng Zhou§

October 11, 2024

Abstract

For Password-Based Authenticated Key Exchange (PAKE), an idealized setup such as random oracle (RO) or a trusted setup such as common reference string (CRS) is a must in the universal composability (UC) framework (Canetti, FOCS 2001). Given the potential failure of a CRS or RO setup, it is natural to consider distributing trust among the two setups, resulting a CRS-or-RO-setup (i.e., CoR-setup).

However, the infeasibility highlighted by Katz et al. (PODC 2014) suggested that it is impossible to construct UC-secure PAKE protocols with a straightforward CoR-setup (i.e., either the CRS is functional but the RO is compromised, or the RO is functional but the CRS is compromised). To circumvent this impossibility result, we investigate how to design UC-secure PAKE protocols with a *fine-grained CoR-setup*, where *either* the CRS is functional but the RO is non-functional, *or* vice versa. Different from the straightforward CoR-setup, a fine-grained non-functional setup is not necessarily completely compromised and fully controlled by the adversary; Instead, we consider this non-functional setup may still offer certain security properties. Certainly, the non-functional setup alone should be useless for achieving UC-security.

We present a UC-secure PAKE protocol under two conditions: *either* the CRS is functional while the RO is non-functional (falling back to a collision-resistant hash function), *or* the RO is functional while the CRS is non-functional (falling back to a global CRS). Before presenting our construction, we first prove that a global CRS setup alone is *insufficient* for achieving UC-secure PAKE. This impossibility result highlights the non-triviality of our approach.

To obtain our construction, we introduce several techniques as follows:

(1) We propose a new variant of Non-Interactive Key Exchange (NIKE), called *homomorphic NIKE with associated functions*, which captures key properties of existing RO-based PAKE protocols. This new primitive serves as an important component in our construction.
(2) We develop a *"Brute Force" extraction strategy* which allows us to provide security analysis for our UC-secure PAKE with a fine-grained CoR-setup for polynomial-sized password spaces.
(3) We introduce a novel *password space extension technique* that enables the expansion of PAKE protocols from polynomial-sized to arbitrary-sized password spaces.
(4) Finally, to ensure provable security for our password space extension in UC-secure PAKEs, we modify existing PAKE functionalities to prevent responses that reveal the correctness of password guesses. This is a reasonable adjustment, as our protocol provides only implicit authentication.

We further present a PAKE protocol in the BPR framework (Bellare, Pointcheval, Rogaway, EuroCrypt 2000), assuming *either* the CRS is functional while the RO falls back to a collision-resistant hash function, *or* the RO is functional but the CRS trapdoor is allowed to be learned by the adversary.

*An earlier version of this paper was titled "Fine-Grained Setup Combiners for PAKEs."

†School of Computer Science and Technology, Dongguan University of Technology. Email: xiaoyuting@dgut.edu.cn. Initial work carried out while at Institute of Information Engineering, Chinese Academy of Sciences.

‡Institute of Information Engineering, Chinese Academy of Sciences. Email: r-zhang@iie.ac.cn

§Department of Computer Science, Virginia Commonwealth University. Email: hszhou@vcu.edu.

Contents

1 Introduction

In modern cryptography, we follow a rigorous provable security approach. New cryptographic schemes and protocols must have security proofs, demonstrating that they meet specified security definitions under given assumptions and setups. These setups[1] include: (1) "trusted setups", which assume trusted initialization processes, such as the common reference string (CRS) where all parties access a string generated by a trusted entity [BFM88, BSMP91]; and (2) "idealized setups", such as the random oracle (RO), where a function is assumed to behave as a truly random function [BR93] accessible by all parties.

RO vs. CRS: which setup is better? Clearly these setups are *incomparable*. Take succinct non-interactive argument of knowledge (SNARK) as an example. Many fast SNARKs have been developed using a CRS (e.g., [Gro10, GGPR13]; please see [Tha22] for a great survey). However, in a real-world blockchain project (e.g., Zcash [Zca, BCG+14]), it could be hard to convince people that the CRS trapdoor for the SNARK protocol was properly eliminated during the setup phase. Then in such applications, RO-based[2] SNARKs can be more desirable, as unlike the CRS, *no trapdoor information is required* for setting up the RO. It is important to note that this does not imply that an idealized setup is perfect. It is already established that ROs within a provably secure protocol may never be instantiated using real-world hash functions [CGH98, GK03, BBP04, MRH04, DOP05].

Multi-setups. It remains unclear whether using RO is definitively better than using CRS, or vice versa. Consequently, one might naturally attempt to *distribute trust among multiple setups*, hoping that even if most setups fail, the desired security can still be achieved as long as at least one remains functional. Indeed, along this way, researchers have developed interesting ideas for constructing protocols with *multi-setups* [GO07, GK08, GGJS11, KKZZ14, Yon14]. Groth and Ostrovsky [GO07] were the first to propose a multi-CRS model for non-interactive zero-knowledge proof (NIZK), in which the minority of the CRS's can fail (i.e., be adversarially generated). Then, Goyal and Katz [GK08] studied secure multi-party computation (MPC) in the Universal Composability (UC) framework, under a *combination of a CRS and the honest majority of protocol parties* (i.e., either the CRS is compromised but a majority of the parties are honest, or the CRS is not compromised but a majority of the parties are no longer honest). Garg et al. [GGJS11] initiated the research of feasibility of UC-secure MPC with potentially different setups that can fail in certain ways. Katz et al. [KKZZ14] unified and extended the above results, by showing that MPC with dishonest majority of setups is impossible in the UC framework, where the adversary can actively corrupt setups.

1.1 Related work on PAKEs, and our main question

In this paper, we focus on *Password-based Authenticated Key Exchange (PAKE)* protocols. A PAKE protocol enables two users to authenticate each other over an open network using a shared *low-entropy* password, and output a shared *high-entropy* session key. Typically, a PAKE protocol is considered to be secure if the best attack is *online password-guessing* where the adversary is able to validate one password-guess per session. Regarding the formal security definitions for PAKEs, Bellare, Pointcheval and Rogaway [BPR00] presented a widely-used *game-based* security definition, denoted as BPR-security. Later, Canetti, Halevi, Katz, Lindell and MacKenzie [CHK+05] introduced a security definition in the UC framework, denoted as UC-security. It was shown that UC-security implies BPR-security [CHK+05].

Constructing PAKEs with RO/CRS. PAKEs have been constructed using a CRS [KOY01, GL03, JG04, CHK+05, ACP09, KV09, GK10, CDVW12, ABB+13, BBC+13, ABP15, JR15, BC16, ZY17, JGH+20], or using a RO [BMP00, Mac02, HL18, MRR20, AHH21][3]. Many interesting protocols [AP05, HS14, HR10, PW17] use *both* CRS and RO. We note that, there are PAKE protocols [GL01, NV04, BCL+05, GJO10, Goy12, CGJ15] in the *plain* model, where neither a trusted setup nor an idealized setup is used. Unfortunately these protocols are far from practical. In addition, as already proven by Canetti et al. [CHK+05], a UC-secure PAKE does *not* exist in the plain model, regardless of efficiency.

[1] In this paper, we focus on trusted setups and idealized setups. Researchers have also investigated "hardware setups" (e.g., [Kat07, BFSK11, PST17]), and other different types of setups.

[2] These RO-based SNARKs are sometimes called "SNARKs with a *transparent* setup".

[3] A different idealized setup, the *ideal cipher* (IC), has also been used [BPR00, ACCP08, BCJ+19].

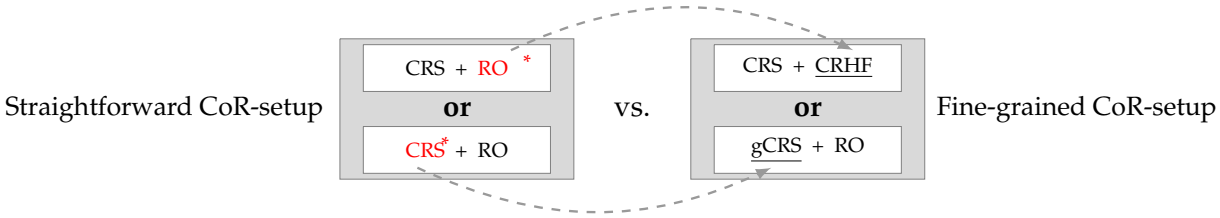
Straightforward CoR-setup for UC-secure PAKE fails. It is known that a trusted setup or an idealized setup is necessary for constructing UC-secure PAKE protocols. Suppose such setups may fail, then one may come up with a "straightforward" CRS-or-RO-setup (i.e., CoR-setup), by combining one CRS and one RO, expecting that either fails, the other can remain functional. Unfortunately, such straightforward CoR-setup cannot achieve UC-security. As mentioned above, Katz et al. [KKZZ14] have already demonstrated an impossibility result: UC-secure MPC is infeasible when the majority of the setups are actively corrupted (i.e., fully controlled by the adversary). That means, among the two setups, namely a CRS and a RO, we cannot obtain UC-secure MPC protocols when either setup is actively corrupted. This impossibility is true for UC-secure PAKE protocols.

Maximizing the utility of RO/CRS: Fine-grained CoR-setup. In the above straightforward CoR-setup, the worst-case scenario is assumed: one setup is considered functional, and the other setup is considered to be *completely controlled by the adversary*. We note that, the impossibility result by Katz et al [KKZZ14], holds if the majority of the setups are *completely controlled by the adversary*.

How can we maximize the utility of setups? In many real-world applications, a setup may still provide certain security guarantees even when it becomes non-functional. We refer to this as a fine-grained CoR-setup, where either the CRS or RO can be non-functional yet *does not have to be fully controlled* by the adversary. For instance, ROs are often instantiated with deterministic hash functions like SHA2 or SHA3. When an RO can no longer be treated as an idealized setup for security proofs, it can *fall back* to a collision-resistant hash function (CRHF). In contrast, a straightforward CoR-setup assumes the RO is fully controlled by the adversary, making it unsuitable for treating as a CRHF.

Similarly, a CRS is essentially a set of pre-generated public parameters. In situations where we neither rely on it for security proofs nor wish adversaries to exploit its advantages, the CRS can fall back to a global CRS (gCRS). We note that, a global CRS is fundamentally weaker than a regular CRS: A global CRS *cannot* be used as a setup for UC-secure MPC (cf. [CDPW07]) or for UC-secure PAKE (details given in Section 3). We stress that weakened variants of the CRS have previously been investigated: Canetti, Pass and shelat [CPs07] study the case that the reference string is taken from an adversarially specified distribution unknown to the players; Katz et al. [KKZZ14] studied the case that the CRS is passively corrupted. Very differently, these weakened versions of the CRS can still be used as a setup for achieving UC-secure MPC.

From the above discussions, it is easy to see that there is a fundamental difference between a straightforward CoR-setup and a fine-grained CoR-setup; See Fig. 1 for the comparison.



‡ The text in red with superscript "$*$" means that the corresponding setup is actively corruptible, and can be completely controlled by the adversary.

Figure 1: Straightforward CoR-setup vs. Fine-grained CoR-setup

It is worth emphasizing that our fine-grained CoR-setup focuses on the case that, among the two setups, one is functional, while the other falls back to a "useless" setup[4] (i.e., the setup is not fully controlled by the adversary, yet still insufficient to achieve UC-security).

Our pursuit of a fine-grained CoR-setup arises from a proactive approach to mitigating potential vulnerabilities that may emerge in protocols relying solely on one setup, i.e., "Don't put all your eggs in one basket". Consider a scenario where protocols based on a setup deemed satisfactory today become compromised or outdated in the future. The consequences could be severe, especially if these protocols have

[4]In contrast, in previous works (e.g., [CPs07, KKZZ14]), a functional setup falls back to a useful setup: the sunspot [CPs07] and the passively corrupted CRS [KKZZ14] could be used as a setup for achieving UC-secure MPC. Furthermore, in [CPs07, KKZZ14], only feasibility results are demonstrated; secure MPC protocols using techniques in [CPs07, KKZZ14] are far from being efficient for practical applications.

already been deployed in practical settings, potentially leading to significant damage. However, if a fine-grained CoR-setup were developed, the security of an already deployed protocol could be maintained as long as at least one of the two setups remains satisfied. This effectively means that the same protocol can be proven secure under two different setups simultaneously. We therefore can provide **two security interpretations** for the **same** (already deployed) protocol.

These lead to our main research question:

Main question. *Is it possible to develop UC-secure PAKEs with a fine-grained CoR-setup?*

1.2 Our results

We give an affirmative answer to the above question, and we have several *theoretical* contributions towards the goal.

Main result: A PAKE protocol with a fined-grained CoR-setup in the UC framework. We construct a UC-secure PAKE protocol that can be interpreted **either** in the $\mathcal{F}_{\text{CRS}}$-hybrid world with CRHFs—using a functional CRS setup $\mathcal{F}_{\text{CRS}}$ alongside a non-functional RO setup that relies on CRHF—**or** in the $\{\mathcal{G}_{\text{CRS}}, \mathcal{F}_{\text{RO}}\}$-hybrid world—employing a functional RO setup $\mathcal{F}_{\text{RO}}$ and a non-functional CRS setup that falls back to a global CRS setup $\mathcal{G}_{\text{CRS}}$. It is important to note that constructing a UC-secure PAKE protocol under a global setup $\mathcal{G}_{\text{CRS}}$ is *impossible* (see Sec. 3), as there is no known method to embed a trapdoor into the global CRS.

Towards this goal, novel design and analysis techniques are discovered and listed below.

A New primitive for PAKE constructions: Introducing a novel Non-Interactive Key Exchange (NIKE) [FHKP13] variant termed as homomorphic NIKE with associated functions.

We present this new primitive to encapsulate essential properties found in several existing RO-based PAKE protocols to establish their security, including the well-known group-based standards, PPK [Mac02] and SPAKE2 [AP05].

Informally, a NIKE scheme enables a party $\mathcal{P}$ to compute a shared key k with party $\mathcal{P}'$ from $\mathcal{P}$'s secret key dk and $\mathcal{P}'$'s public key ek', where $\mathsf{k} = \textsc{Key}(\mathsf{dk}, \mathsf{ek}')$. Using a NIKE as the foundation for PAKE protocols, both parties must incorporate their passwords in such a way that they can negotiate the same NIKE shared key only when they use identical passwords, allowing them to subsequently derive the same session key. To encapsulate this, we first define homomorphic property for NIKE scheme, which requires that:

$$\begin{cases} \textsc{Key}(\mathsf{dk}_0 + \mathsf{dk}_1, \mathsf{ek}') = \textsc{Key}(\mathsf{dk}_0, \mathsf{ek}') \cdot \textsc{Key}(\mathsf{dk}_1, \mathsf{ek}'); \\ \textsc{Key}(\mathsf{dk}, \mathsf{ek}'_0 \cdot \mathsf{ek}'_1) = \textsc{Key}(\mathsf{dk}, \mathsf{ek}'_0) \cdot \textsc{Key}(\mathsf{dk}, \mathsf{ek}'_1). \end{cases}$$

We then define two pairs of associated functions $(f_\sigma, \hat{f}_\sigma)_{\sigma \in \{0,1\}}$: If $\mathsf{pek} = f_\sigma(x, \mathsf{ek})$, then $\hat{f}_\sigma(x, \mathsf{pek}) = \mathsf{ek}$ and $\hat{f}_\sigma(x, \mathsf{pek} \cdot \mathsf{ek}') = \hat{f}_\sigma(x, \mathsf{pek}) \cdot \mathsf{ek}'$, where x is used to capture passwords in PAKE protocols. Furthermore, we define the following properties:

- *Perfect hiding.* Regarding $\mathsf{pek} = f_\sigma(x, \mathsf{ek})$, it can be viewed as generating a perfect hiding commitment of x.
- *Equivocability.* Regarding pek, with the knowledge of the corresponding secret key pdk, it can be opened as a commitment of any x.
- *Intractability.* This property encapsulates the security guarantee that the target NIKE shared key is difficult to compute. It applies to two types of PAKE protocols: those in which both parties must commit to their passwords and those where only one party is required to do so. It primarily captures two security aspects: (1) A passive adversary cannot carry out successful offline guessing attacks; (2) An active adversary can validate only a single password guess within a single session.

New analysis: Developing the "Brute Force" extraction strategy.

To complete a UC analysis, it is crucial to ensure that the simulator can extract the adversary's input, regardless of which setup is non-functional. In this work, a non-functional RO setup falls back to a CRHF (denoted as $\mathcal{H}$). It is well known that low-entropy inputs can make hash functions susceptible to exhaustive search attacks. This motivates our "Brute Force" extraction strategy: if the password pw is chosen from a

polynomial-sized space, the simulator only needs to extract $\mathcal{H}(\mathsf{pw}, *)$ and try each candidate password to recompute the hash values in order to match the target.

Password space extension: Extending PAKEs from a polynomial-sized password space to an arbitrary-sized password space.

To ensure the success of the "Brute Force" extraction strategy, we must assume a polynomial-sized password space, which may be a limitation in certain application scenarios. To address this, we propose a method that extends a PAKE protocol with a polynomial-sized password space to accommodate arbitrary-sized password spaces.

New modifications to PAKE functionalities: Parameterizing the functionalities with a password space; Preventing the functionality from offering responses related to the correctness of a password guess.

To be compatible with the above "Brute Force" extraction strategy and password space extension, we made two modifications to existing PAKE functionalities:

(1) The functionality is parameterized with a password space, and the functionality only accepts passwords belonging to the given password space.
(2) For the TestPwd interface provided to the ideal adversary to test its password guess, the functionality no longer directly responds with "correct guess" or "wrong guess." With this modification, the functionality will not provide feedback on the accuracy of a password guess.

We seek clarification on the rationale behind the second modification. We believe this is reasonable because our work focuses solely on implicit authentication[5]. In a PAKE protocol providing implicit authentication, the involved parties generate their session keys independently. If they use the same input password, they derive the same session key; otherwise, they produce random independent session keys. Crucially, neither party can distinguish between these outcomes until they compare their session keys.

Even if an adversary actively participates in the protocol execution, it cannot ascertain the correctness of its password guess without obtaining additional input or output from an honest party. Therefore, we propose modifying the functionality to limit the capabilities of the ideal adversary accordingly.

Additional result: PAKE with a weaker fine-grained CoR-setup in the BPR framework. We also present a BPR-secure PAKE protocol with a weaker fine-grained CoR-setup than we used in the UC framework: (1) The CRS setup is functional, while the RO setup falls back to CRHFs; or (2) The RO setup is functional, while the CRS setup falls back to a case that the adversary may obtain the trapdoor embedded in the CRS. This result is presented in Appendix E.

1.3 Our techniques

We address our main question by composing a RO-based sub-protocol, Π_{RO}, and a CRS-based sub-protocol, Π_{CRS}. At a very high level, our protocol generates two session keys using these sub-protocols and computes the XOR of the resulting keys. The primary challenge of this work lies in ensuring that the simulator operates correctly, regardless of which setup is non-functional during this composition.[6] In particular, we face the following two technical issues:

- How can the simulator extract the adversary's input?
- How can the simulator produce an indistinguishable view, when the adversary makes a correct password guess?

[5]PAKE protocols can have two types of authentication: implicit and explicit. Implicit authentication means that at the end of the protocol, if both parties used the same password, they share the same session key; otherwise, they end up with random independent session keys. Explicit authentication ensures that both parties are aware of their situation.

[6]A trivial solution involves executing Π_{RO} and Π_{CRS} *in parallel*, which proves ineffective. To successfully complete a UC analysis, we need to construct a simulator that must possess both CRS and RO capabilities simultaneously. Our goal is to achieve UC security with a fine-grained CoR-setup, where the simulator has only one capability in each case. When the CRS setup is functional, the simulator is restricted to embedding trapdoors in the CRS, causing it to falter when simulating the view of Π_{RO}. This issue similarly arises in the opponent case.

1.3.1 A non-functional RO setup admits a candidate "Brute Force" extraction strategy.

Recall that a non-functional RO setup falls back to a CRHF (denoted as $\mathcal{H}$). If the password pw is selected from a polynomial-sized space, then the structure $\mathcal{H}(\mathsf{pw}, *)$ allows for a "Brute Force" extraction strategy. Once the simulator extracts $\mathsf{ppw} = \mathcal{H}(\mathsf{pw}, *)$, it can recompute hash values for all candidate passwords to match ppw. Here, we use the notation ppw to indicate a "processed" password, distinguishing it from the original password pw.

We have Π_{RO} and Π_{CRS} executed, *sequentially*. That means we start with Π_{RO}, followed by Π_{CRS}. In particular, Π_{RO} takes pw as input and generates ppw, which serves as the input for Π_{CRS}. This sequential composition serves as the foundation of our protocol. To guarantee the success of the Brute-Force extraction strategy when the RO setup is non-functional, while also accommodating a successful extraction strategy when the RO setup is functional, we must meticulously design Π_{RO}. In particular, we take *homomorphic NIKE with associated functions* as our building block. The core of our design for Π_{RO} is shown in Fig. 2.

Client $\mathcal{P}_i$ (pw_i):		Server $\mathcal{P}_j$ (pw_j):
$(\mathsf{ek}_0, \mathsf{dk}_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$;		$(\mathsf{ek}'_0, \mathsf{dk}'_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$;
$(\mathsf{ek}_1, \mathsf{dk}_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$;		$(\mathsf{ek}'_1, \mathsf{dk}'_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$;
$\mathsf{pek}_0 = f_0(\mathsf{pw}_i, \mathsf{ek}_0)$;		$\mathsf{pek}'_1 = f_1(\mathsf{pw}_j, \mathsf{ek}'_1)$;
	$\xrightarrow{\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)}$	
	$\xleftarrow{\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)}$	
$\mathsf{k}_{i,0} = \textsc{Key}(\mathsf{dk}_0, \mathsf{ek}'_0)$;		$\mathsf{k}_{j,0} = \textsc{Key}(\mathsf{dk}'_0, \hat{f}_0(\mathsf{pw}_j, \mathsf{pek}_0))$;
$\mathsf{k}_{i,1} = \textsc{Key}(\mathsf{dk}_1, \hat{f}_1(\mathsf{pw}_i, \mathsf{pek}'_1))$;		$\mathsf{k}_{j,1} = \textsc{Key}(\mathsf{dk}'_1, \mathsf{ek}_1)$;
$\mathsf{trans} = i \vert j \vert \mathsf{msg} \vert \mathsf{msg}'$;		$\mathsf{trans} = i \vert j \vert \mathsf{msg} \vert \mathsf{msg}'$;
$\omega_{i,0} = \mathcal{H}_0(\mathsf{trans} \vert \mathsf{k}_{i,0} \vert \mathsf{pw}_i)$;		$\omega_{j,0} = \mathcal{H}_0(\mathsf{trans} \vert \mathsf{k}_{j,0} \vert \mathsf{pw}_j)$;
$\omega_{i,1} = \mathcal{H}_1(\mathsf{trans} \vert \mathsf{k}_{i,1} \vert \mathsf{pw}_i)$;		$\omega_{j,1} = \mathcal{H}_1(\mathsf{trans} \vert \mathsf{k}_{j,1} \vert \mathsf{pw}_j)$;
$\mu_i \vert \hat{\mu}_i \vert \theta_i = \overline{\mathcal{H}}(\mathsf{trans} \vert \omega_{i,0} \vert \omega_{i,1} \vert \mathsf{pw}_i)$;		$\mu_j \vert \hat{\mu}_j \vert \theta_j = \overline{\mathcal{H}}(\mathsf{trans} \vert \omega_{j,0} \vert \omega_{j,1} \vert \mathsf{pw}_j)$;
Output θ_i as the session key.		Output θ_j as the session key.

† $\textsc{Gen}$ is the key generation algorithm of the underlying NIKE scheme. $\mathcal{H}_0 : \{0,1\}^* \to \{0,1\}^{2\lambda}$, $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^{2\lambda}$ and $\overline{\mathcal{H}} : \{0,1\}^* \to \{0,1\}^{6\lambda}$ are three hash functions.

Figure 2: The core of our design for Π_{RO}

If we view the entire protocol of Π_{RO} as a "Full RO-PAKE", which can be viewed as concurrently running two instances of the same "Half RO-PAKE":

- The first one is Client-to-Server, where the client masks its NIKE contribution ek_0 with its input password pw_i, sending $\mathsf{pek}_0 = f_0(\mathsf{pw}_i, \mathsf{ek}_0)$ (that can be viewed as a perfect hiding commitment to pw_i), while the server only sends its NIKE contribution ek'_0.
- The second one is Sever-to-Client, where the server masks its NIKE contribution ek'_1 with its input password pw_j, sending $\mathsf{pek}'_1 = f_1(\mathsf{pw}_j, \mathsf{ek}'_1)$ (that can be viewed as a perfect hiding commitment to pw_j), while the client only sends its NIKE contribution ek_1.

If the two parties use the same password, they can compute a shared secret $\mathsf{k}_0 = \mathbf{Key}(\mathsf{ek}_0, \mathsf{ek}'_0)$ (resp., $\mathsf{k}_1 = \mathbf{Key}(\mathsf{ek}_1, \mathsf{ek}'_1)$) from the first (resp., second) instance, then process k_0 (resp., k_1) through $\mathcal{H}_0$ (resp., $\mathcal{H}_1$) into a shared output ω_0 (resp., ω_1). The shared outputs ω_0 and ω_1 of these two "Half RO-PAKEs" are finally combined via $\hat{\mathcal{H}}$ into values $(\mu, \hat{\mu}, \theta)$.

Plugging Π_{RO} shown in Fig. 2 into our Π_{RO}-then-Π_{CRS} design, we have that:

(1) *The "Brute-Force" extraction strategy is effective when the RO setup is non-functional.* When simulating an honest server (resp., client), the simulator can extract the adversary's input password from the 1st (resp., 2nd) instance of "Half RO-PAKE": The simulator knows dk'_0 (resp., dk_1), which allows it to try each candidate password to compute k_0 (resp., k_1) and match ω_0 (resp., ω_1).

(2) *When the RO setup is functional, its observability provides the simulator with a mean of extraction.* When simulating an honest server (resp., client), the simulator can extract the adversary's input password from the 1st (resp., 2nd) instance of "Half RO-PAKE": The simulator observes all queries the adversary made on $\mathcal{H}_0$ (resp., $\mathcal{H}_1$). The Type-II (ii) intractability (cf. Sec. 2.2.2) of the underlying homomorphic NIKE scheme

ensures that the adversary can only compute one valid k_0 (resp., k_1) value with its committed password through pek_0 (resp., pek'_1).

1.3.2 Twisting the two "Half RO-PAKEs" in a subtle manner ensures the simulator can always produce indistinguishable view.

Given Π_{RO} shown in Fig. 2, we now need to define ppw that will be used as the input for Π_{CRS}. In the next, we have three design attempts. We will explain why the first two attempts do not work, with the goal of understanding the rationale of the final successful attempt.

– ***The first attempt:*** Set $\mathsf{ppw} := \omega_0|\omega_1$.

By doing so, the simulator can effectively implement the "Brute Force" extraction strategy when the RO setup is non-functional. However, issues arise when the adversary guesses the correct password. In this scenario, the simulator must compute the same output session key as the environment, which requires the simultaneous recomputation of k_0 and k_1. Unfortunately, the simulator can only recompute one, corresponding to the instance of "Half RO-PAKE" where it does not commit to a input password.

Possible Solution: Let the server (resp., the client) "**send**" the value k_0 (resp., k_1) to its peer party.

– ***The second attempt:*** Set $\mathsf{ppw} := (\omega_0 \odot \mathsf{k}_1)|(\omega_1 \odot \mathsf{k}_0)$.

By doing so, the simulator can compute the exact value as the adversary does when the RO setup is non-functional. However, new issues appear when the RO setup is functional (meanwhile the CRS setup is not functional). A key observation is as follows:

Assume an adversary is playing the role of server. After receiving the message $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$, the simulator may be unable to extract the adversary's input at once.[7] Meanwhile, the simulator is expected to provide its input $\mathsf{ppw} := (\omega_0 \odot \mathsf{k}_1)|(\omega_1 \odot \mathsf{k}_0)$ for Π_{CRS}, necessitating an immediate assignment of ω_0. However, since the adversary knows dk'_0, such that it may have tried all candidate passwords in $\mathcal{H}_0$ queries. Without knowing the honest server's input password, the simulator fails with high probability when the adversary makes a correct password guess.

Possible Solution: Let ω_0 and ω_1 be XORed, such that the simulator can postpone assigning ω_0 until it detects that the adversary has queried $\mathcal{H}_1$ on the correct value. More concretely, when an input is expected to be provided for Π_{CRS}, the simulator replaces the value $\omega_0 \oplus \omega_1$ with a random value η^*, which is also recorded to program subsequent $\mathcal{H}_1$ queries. Since the adversary has previously sent pek'_1, the underlying password guess pw^* was already fixed. Given a tuple $(\mathsf{ek}_1, \mathsf{pek}'_1)$, the adversary can compute at most one valid k_1 value. When detecting the adversary queries $\mathcal{H}_1$ on a correct value, the simulator is also able to extract pw^*. (Remark that we are considering the case that the adversary makes a correct password guess.) After that, the simulator can use pw^* as index to search in the list that records all $\mathcal{H}_0$ queries, and use the corresponding record to assign ω_0. Finally, the simulator sets $\omega_1 = \eta^* \oplus \omega_0$ and uses ω_1 to answer the corresponding $\mathcal{H}_1$ query.

– ***The final successful attempt:*** Set $\mathsf{ppw} := (\omega_0 \oplus \omega_1)|(\mu \odot \mathsf{k}_0)|(\hat{\mu} \odot \mathsf{k}_1)$.

By doing so, when the RO setup is non-functional, the simulator can still employ the "Brute Force" extraction strategy. Since the sub-protocol Π_{CRS} is invoked in a black box manner, the simulator can use existing CRS-based simulation strategy to extract ppw (then employ the "Brute Force" extraction strategy to extract pw) and generate indistinguishable view for the adversary.

Conversely, when the RO setup is functional, the simulator can extract the adversary's password guess through observing queries it made on $\mathcal{H}_0$ or $\mathcal{H}_1$. In particular, the third RO $\overline{\mathcal{H}}$ is used to ensure that the adversary must make valid such queries before computing the correct values for μ, $\hat{\mu}$ and θ, where θ will be XORed with the output session key of Π_{CRS} to derive the final session key. The simulator can also apply the programmability of RO to generate indistinguishable view for the adversary.

More details can be found in Sec. 4 and Theorem 5.

[7] In the discussed case, the simulator can extract the adversary's input by observing all queries it made on $\mathcal{H}_1$. However, the adversary may not necessarily query $\mathcal{H}_1$ before sending out the message msg'.

1.3.3 A modified functionality meets password space extension.

Given a protocol $\mathrm{Prot}_{\mathrm{UNIT}}$ that UC-realizes PAKE for a polynomial-sized password space $\mathcal{D}$, we can construct another protocol that UC-realizes PAKE for an arbitrary-sized password space $\mathcal{D}^n = \mathcal{D} \times \cdots \times \mathcal{D}$ with $n \in \mathbb{N}$. For simplicity, a password $\mathsf{pw} \in \mathcal{D}^n$ can be expressed as the concatenation of n shorter passwords $\mathsf{pw}_1, \ldots, \mathsf{pw}_n \in \mathcal{D}$. Specifically, we have $\mathsf{pw} = \mathsf{pw}_1 | \ldots | \mathsf{pw}_n$. We begin by concurrently running n instances of $\mathrm{Prot}_{\mathrm{UNIT}}$; each instance uses pw_i as its input, where $i = 1, \ldots, n$. Let SK_i denote the output session key from each execution instance of $\mathrm{Prot}_{\mathrm{UNIT}}$. Finally, we set the overall session key $\mathsf{SK} := \bigoplus_{i=1}^{n} \mathsf{SK}_i$.

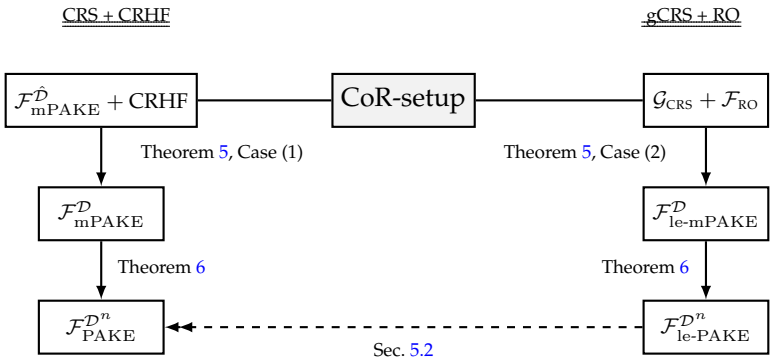
Intuitively, as long as $\mathrm{Prot}_{\mathrm{UNIT}}$ only provides implicit authentication, we have constructed a protocol that UC-realizes PAKE with $\mathcal{D}^n$. It is important to note that the honest party's secret intermediates $(\mathsf{SK}_1, \ldots, \mathsf{SK}_n)$ will not be revealed to the environment; instead, a summary value $\mathsf{SK} := \bigoplus_{i=1}^{n} \mathsf{SK}_i$ will be output. As a result, the adversary cannot verify whether its guess for each sub-password pw_i is correct. In contrast, the adversary's guesses for all sub-passwords $(\mathsf{pw}_1, \ldots, \mathsf{pw}_n)$ are treated as a whole.

What contradicts intuition is how existing PAKE functionalities define the adversary's ability to guess passwords online. In particular, they offer the adversary with a TestPwd interface. Upon receiving the adversary's password guess, the functionality first compares it with the honest party's password and then replies with either "correct guess" or "wrong guess" accordingly. If we use this type of functionality, the simulator will get stuck when it needs to reply to the adversary with "correct guess" or "wrong guess" for a sub-password guess. To avoid this, we define a modified functionality that removes TestPwd's reply of "correct guess" or "wrong guess".

More details can be found in Sec. 5.1 and Theorem 6.

1.3.4 The roadmap of our main results.

By applying the above techniques, we obtain our main results, summarized in Fig. 3, which incorporates several variants of PAKE functionality. Let $\mathcal{F}_{\mathrm{PAKE}}$ denote the original PAKE functionality [CHK+05], and $\mathcal{F}_{\text{le-PAKE}}$ denote its "lazy extraction" variant [ABB+20]. In this paper, we add a password space parameter to their top right corner, indicating the space of allowable passwords. We also use the presence or absence of "m" to indicate whether the functionality's TestPwd interface has been modified as we previously discussed.



† The notation "A→ B" indicates that we construct a protocol that UC-realizes B in the A-hybrid world. The dashed arrow indicates the possibility to upgrade the security.

‡ The notation $\mathcal{F}_{\mathrm{name}}^{\mathrm{ps}}$ denotes the functionality $\mathcal{F}_{\mathrm{name}}$ parameterized with a password space ps, where $\mathrm{name} \in \{\text{PAKE, le-PAKE, mPAKE, le-mPAKE}\}$ and $\mathrm{ps} \in \{\hat{\mathcal{D}}, \mathcal{D}, \mathcal{D}^n\}$. The prefixes "m" and "le-" denote different variants of the original PAKE functionality $\mathcal{F}_{\mathrm{PAKE}}$: "m" indicates that the TestPwd interface is modified; "le-" means the "lazy extraction". In addition, $\{0,1\}^{6\lambda} \subseteq \hat{\mathcal{D}}$, $|\mathcal{D}| = \mathsf{poly}(\lambda)$ and $\mathcal{D}^n = \mathcal{D} \times \cdots \times \mathcal{D}$.

Figure 3: The results of our UC-secure PAKE protocols with a CoR-setup

We construct a PAKE protocol based on homomorphic NIKE with associated functions, a sub-protocol $\mathrm{Prot}_{\mathrm{PAKE}}$ that UC-realizes $\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}$ in the $\mathcal{F}_{\mathrm{CRS}}$-hybrid world, and three hash functions (that will be treated as CRHFs or ROs). In Theorem 5, we show our PAKE protocol is UC-secure with a CoR-setup. In particular,

our protocol UC-realizes $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ when the CRS setup is functional but the RO setup falls back to CRHFs; it also UC-realizes $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$ when the RO setup is functional but the CRS setup falls back to gCRS.

No matter in which case, based on Theorem 6, the password space of the protocol can be extended from $\mathcal{D}$ to $\mathcal{D}^n$ for $n \in \mathbb{N}$, achieving $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}^n}$ and $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}^n}$, respectively. Now, the achieved functionality is no longer the variant with "m", i.e., TestPwd's reply of "correct guess" or "wrong guess" is kept.

Since our construction merely black-box invokes $\text{Prot}_{\text{PAKE}}$, we only achieve a "lazy extraction" PAKE functionality variant, $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}^n}$ when the RO setup is functional. We also discussed in Sec. 5.2, if we explore the internal structure of $\text{Prot}_{\text{PAKE}}$, the result can be upgraded to $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}^n}$. As a result, we achieve $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}^n}$ using a CoR-setup.

1.4 Paper organization

In Sec. 2, we define some useful ideal functionalities for PAKE, homomorphic NIKE with associated functions. In Sec. 3, we prove impossibility of UC-secure PAKE in the $\mathcal{G}_{\text{CRS}}$-hybrid world. In Sec. 4, we present our PAKE protocol with a fine-grained CoR-setup in the UC framework. In Sec. 5, we present three important extensions (including our password space extending protocol) and discuss the practicality of our UC-secure PAKE protocol.

For completeness, we provide supplemental preliminaries in Appendix A and supplemental materials for homomorphic NIKE with associated functions in Appendix B. In Appendix C, we present the security proof details for our protocol in the UC framework. In Appendix D, we present the security proof details for our password space extending protocol. In Appendix E, we present our further results in the BPR framework.

2 Preliminaries

Notations. We use $\lambda \in \mathbb{N}$ to denote a security parameter. We use $|\cdot|$ to denote the cardinality of a set or the bit length of a string. We use "$|$" to denote concatenation of strings. We use "$:=$" to denote "be defined as". We use "$\overset{c}{\approx}$" to denote computational indistinguishability. We use "$\overset{\$}{\leftarrow}$" to denote a randomized process, and "$=$" to denote a deterministic process. For a deterministic algorithm DALG, we use $y = \text{DALG}(x)$ to denote running it with x as input and obtain output y. For a probabilistic algorithm PALG, we use $y \overset{\$}{\leftarrow} \text{PALG}(x)$ to denote running it with x as input and obtain output y. A probabilistic algorithm will become deterministic once its internal randomness r is explicitly specified, which is denoted as $y = \text{PALG}(x; r)$. We use "$\odot$" to denote the extension of $\oplus$ to non-bit strings. For example, for $a \in \{0,1\}^\lambda$ and $b \in \mathbb{G}$, where $\mathbb{G}$ denotes a cyclic group. Assume there exist two coding algorithms, ENCODE that maps a group element into a bit string, and DECODE that inverts a bit string into a group element. Then $a \odot b := a \oplus \text{ENCODE}(b)$. For simplicity, the reverse process to compute $b = \text{DECODE}((a \odot b) \oplus a)$ is denoted as $b = a \odot b \odot a$. A positive function $\mathsf{negl}(\lambda)$ is called negligible, if for all positive polynomial $p(\cdot)$, there exists a constant $\lambda_0 > 0$ such that for all $\lambda > \lambda_0$, it holds that $\mathsf{negl}(\lambda) < 1/p(\lambda)$.

2.1 Security definitions for PAKE

In this paper, we define the security for PAKE both in the UC framework [Can01, Can00] (Appendix A.1), and in the BPR framework [BPR00] (Appendix A.2). The differences between the two definition frameworks are discussed in Appendix A.3. Here, we only describe four functionalities (as shown in Fig. 4) for PAKE following the formulations in [ABB+20], which are further based on that in [CHK+05]; and adopting the fix recommended by [AHH21] (i.e., deleting "either $\mathcal{P}_i$ or $\mathcal{P}_j$ corrupt" case in the NewKey interface).

In particular, we introduce two new modifications:

(1) Parameterizing the functionalities with a password space $\mathcal{D}$. This was never done in the existing PAKE functionalities [CHK+05, GK10, ABB+20];
(2) Preventing the functionality from offering responses related to the correctness of a password guess (i.e., deleting 'reply with "correct guess"' and 'reply with "wrong guess"' in the TestPwd interface).

Functionalities for PAKE

The functionalities $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$, $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}}$, $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ and $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$ are parameterized by a security parameter λ and a password space $\mathcal{D}$ for passwords.

Session initialization:
On a query $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, \mathsf{role})$ from $\mathcal{P}_i$, ignore this query if $\mathsf{pw} \notin \mathcal{D}$ or record $\langle \mathsf{sid}, \mathcal{P}_i, \cdot, \cdot, \cdot \rangle$ already exists. Otherwise, record $\langle \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, \mathsf{role} \rangle$ marked fresh and send $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role})$ to $\mathcal{S}$.

Password guessing:
On a query $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ from $\mathcal{S}$, if $\mathsf{pw}^* \in \mathcal{D}$, and there exists a record $\langle \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, \mathsf{role} \rangle$ marked fresh:

- If $\mathsf{pw}^* = \mathsf{pw}$, mark it compromised and reply with "correct guess";
- If $\mathsf{pw}^* \neq \mathsf{pw}$, mark it interrupted and reply with "wrong guess".

On a query $(\mathsf{RegisterTest}, \mathsf{sid}, \mathcal{P}_i)$ from $\mathcal{S}$, if there exists a fresh record $\langle \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, \mathsf{role} \rangle$, then mark it interrupted and flag it tested.

On a query $(\mathsf{LateTestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ from $\mathcal{S}$, if there exists a completed record $\langle \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, *, \mathsf{SK} \rangle$ with flag tested, then remove its flag and do:

- If $\mathsf{pw}^* = \mathsf{pw}$, reply with SK;
- If $\mathsf{pw}^* \neq \mathsf{pw}$, reply with $\mathsf{SK} \xleftarrow{\$} \{0,1\}^\lambda$.

Key generation:
- On a query $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ from $\mathcal{S}$, if $\exists$ a record $\langle \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, \mathsf{role} \rangle$ that is not marked completed:
  (1) If this record is compromised, set $\mathsf{SK}_i \leftarrow \mathsf{SK}^*$;
  (2) Else if this record is fresh and $\exists$ a completed record $\langle \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}, \mathsf{role}', \mathsf{SK}_j \rangle$ with $\mathsf{role} \neq \mathsf{role}'$, set $\mathsf{SK}_i \leftarrow \mathsf{SK}_j$.
  (3) Otherwise, sample $\mathsf{SK} \xleftarrow{\$} \{0,1\}^\lambda$.
  Send SK_i to $\mathcal{P}_i$, mark $\langle \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, \mathsf{role} \rangle$ with completed, and append SK_i to the record.

Figure 4: UC PAKE variants. The original PAKE $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$: Includes the boxed text but excludes the gray text. The modified-PAKE $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$: Excludes both the boxed text and gray text. The lazy-extraction PAKE $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}}$: Includes both the boxed text and gray text. The lazy-extraction modified-PAKE $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$: Includes the gray text but excludes the boxed text.

Let $\mathcal{F}_{\text{PAKE}}$ denote the original PAKE functionality [CHK+05], and $\mathcal{F}_{\text{le-PAKE}}$ denote its "lazy extraction" variant proposed in [ABB+20]. In this paper, the password space parameter is labeled as a superscript. We also use the presence or absence of "m" to indicate whether the functionality's TestPwd interface has been modified as we previously discussed.

The reasonability of the Modification (1). For the first modification, we note that all PAKE protocols proven w.r.t any known PAKE functionality can still be considered secure, where all passwords are in fact chosen from an implicit password space. It is helpful for us to explicitly define the parameter $\mathcal{D}$, because our protocol in the UC framework deals with passwords with low entropy. In practice, this modification is also reasonable, where the passwords are pre-restricted with selection rules (that may include the type of characters, the shortest and longest length of passwords). For example, the passwords used to unlock most mobile phones are 6-digit numbers; WiFi passwords in the WPA/WPA2-PSK require 8-16 digits combination of numbers and letters; Bank card passwords are generally 6-8 digits numbers. We also remark one can replace $\mathcal{D}$ with an explicitly defined password length parameter ℓ which defines the length of all passwords.

However, we should point out that, in these functionalities, the password space (or equivalently, the password length) must be a priori fixed and known to the ideal adversary. This functionality is clearly weaker than the standard one.

The reasonability of the the Modification (2). For the second modification, it is also reasonable as we discussed in Sec. 1.2. We note that if a protocol UC-realizes $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$, it can also be shown to UC-realize $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$. In a UC analysis for $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$, there is a particular case to consider, where the simulator must send

messages before receiving any from the adversary. This means that the simulator must "commit to" a password before extracting the adversary's password guess, without knowing the honest party's password in advance; however, if the adversary makes a correct password guess, the simulator should be capable of computing the same output session key as the adversary.

Now, compare the following two different ways of the simulator querying NewKey to make $\mathcal{F}_{\mathrm{PAKE}}^{\mathcal{D}}$ to send a session key to the honest party:

- If the adversary's password guess is correct, the simulator computes the adversary's output session key and sends it to $\mathcal{F}_{\mathrm{PAKE}}^{\mathcal{D}}$. Otherwise, the simulator sends a random session key to $\mathcal{F}_{\mathrm{PAKE}}^{\mathcal{D}}$.
- No matter the adversary's password guess is correct or wrong, the simulator always computes the adversary's output session key and sends it to $\mathcal{F}_{\mathrm{PAKE}}^{\mathcal{D}}$.

In both cases, the result is essentially the same: if the adversary's guess is correct, it determines the honest party's output session key; If the guess is incorrect, the honest party's output session key is independently chosen at random. This implies that even when the functionality is switched from $\mathcal{F}_{\mathrm{PAKE}}^{\mathcal{D}}$ to $\mathcal{F}_{\mathrm{mPAKE}}^{\mathcal{D}}$, such that the simulator no longer knows whether the adversary's guess is correct, it can still function correctly.

2.2 Homomorphic Non-Interactive Key Exchange with associated functions

We recall Non-Interactive Key Exchange (NIKE) with one-wayness [FHKP13], before we introduce a novel NIKE variant, called *homomorphic NIKE with associated functions*. We use this new primitive to encapsulate a few necessary properties to design and prove security of many RO-based PAKE protocols.

2.2.1 NIKE with one-wayness

A NIKE scheme consists of three polynomial-time algorithms:

- $\textsc{Setup}(1^\lambda)$: On input a security parameter 1^λ, this probabilistic algorithm outputs a public parameter pp.
- $\textsc{Gen}(\mathsf{pp})$: On input a public parameter pp, this probabilistic algorithm outputs a key pair $(\mathsf{ek}, \mathsf{dk}) \in \mathcal{EK} \times \mathcal{DK}$.
- $\textsc{Key}(\mathsf{dk}, \mathsf{ek}')$: On input a secret key $\mathsf{dk} \in \mathcal{DK}$ and a public key $\mathsf{ek}' \in \mathcal{EK}$, this deterministic algorithm outputs a session key $\mathsf{k} \in \mathcal{K}$.

Here, $\mathcal{EK}$, $\mathcal{DK}$ and $\mathcal{K}$ are the sets of public keys, secret keys and session keys.

Correctness. For all $\mathsf{pp} \xleftarrow{\$} \textsc{Setup}(1^\lambda)$, $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $(\mathsf{ek}', \mathsf{dk}') \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, it holds that $\textsc{Key}(\mathsf{dk}, \mathsf{ek}') = \textsc{Key}(\mathsf{dk}', \mathsf{ek})$.

One-wayness (OW). A NIKE scheme $\Pi = (\textsc{Setup}, \textsc{Gen}, \textsc{Key})$ is called *OW-secure*, if for any PPT algorithm $\mathcal{A}$, there exists a negligible function negl, s.t.,

$$\Pr\left[\mathsf{k} = \mathsf{k}^* \;\middle|\; \begin{array}{c} \mathsf{pp} \xleftarrow{\$} \textsc{Setup}(1^\lambda); (\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp}); \\ (\mathsf{ek}', \mathsf{dk}') \xleftarrow{\$} \textsc{Gen}(\mathsf{pp}); \mathsf{k}^* = \textsc{Key}(\mathsf{dk}, \mathsf{ek}'); \\ \mathsf{k} \xleftarrow{\$} \mathcal{A}(1^\lambda, \mathsf{ek}, \mathsf{ek}'). \end{array}\right] \leq \mathsf{negl}(\lambda).$$

For simplicity, in clear contexts, we sometimes use $\mathbf{Key}(\cdot, \cdot)$ to denote computing a session key for a pair of public keys.

2.2.2 Homomorphic NIKE with associated functionalities

Using a NIKE as a building-block for PAKE protocols, both parties must incorporate their passwords in a way such that they can negotiate the same NIKE shared key only when they use an identical password. To implement this idea, we define homomorphic NIKE with associated functionalities.

Homomorphic property. Consider a NIKE scheme $\Pi = (\text{SETUP}, \text{GEN}, \text{KEY})$, whose secret key space $\mathcal{DK}$ are associated with group operation "$+$" and public key space $\mathcal{EK}$ (resp., session key space $\mathcal{K}$) with group operation "$\cdot$". For any $\mathsf{ek}'_0, \mathsf{ek}'_1, \mathsf{ek}' \in \mathcal{EK}$ and $\mathsf{dk}_0, \mathsf{dk}_1, \mathsf{dk} \in \mathcal{DK}$, the following holds:

$$\begin{cases} \text{KEY}(\mathsf{dk}_0 + \mathsf{dk}_1, \mathsf{ek}') = \text{KEY}(\mathsf{dk}_0, \mathsf{ek}') \cdot \text{KEY}(\mathsf{dk}_1, \mathsf{ek}'); \\ \text{KEY}(\mathsf{dk}, \mathsf{ek}'_0 \cdot \mathsf{ek}'_1) = \text{KEY}(\mathsf{dk}, \mathsf{ek}'_0) \cdot \text{KEY}(\mathsf{dk}, \mathsf{ek}'_1). \end{cases}$$

The associated functions. To have a better view of passwords in a PAKE protocol, we introduce a pair of functions $(f_\sigma, \hat{f}_\sigma)$ with space $\mathcal{X}$ to which passwords are mapped. Let $f_\sigma : \mathcal{X} \times \mathcal{EK} \to \mathcal{EK}$ and $\hat{f}_\sigma : \mathcal{X} \times \mathcal{EK} \to \mathcal{EK}$ for $\sigma \in \{0, 1\}$. For any $x \in \mathcal{X}$, $\mathsf{ek}, \mathsf{ek}' \in \mathcal{EK}$ and $\mathsf{pek} = f_\sigma(x, \mathsf{ek})$, the following holds:

$$\begin{cases} \hat{f}_\sigma(x, \mathsf{pek}) = \mathsf{ek}; \\ \hat{f}_\sigma(x, \mathsf{pek} \cdot \mathsf{ek}') = \hat{f}_\sigma(x, \mathsf{pek}) \cdot \mathsf{ek}'. \end{cases}$$

Perfect hiding. For a homomorphic NIKE scheme $\Pi = (\text{SETUP}, \text{GEN}, \text{KEY})$ with associated functions $\{f_\sigma, \hat{f}_\sigma\}_{\sigma \in \{0,1\}}$, we say the *perfect hiding* property holds, if the following distributions are identical for any $x_0, x_1 \in \mathcal{X}$:

$$\left\{ \mathsf{pek} \;\middle|\; \begin{array}{l} (\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \text{GEN}(\mathsf{pp}); \\ \mathsf{pek} = f_\sigma(x_0, \mathsf{ek}). \end{array} \right\} \quad , \quad \left\{ \mathsf{pek} \;\middle|\; \begin{array}{l} (\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \text{GEN}(\mathsf{pp}); \\ \mathsf{pek} = f_\sigma(x_1, \mathsf{ek}). \end{array} \right\}$$

Applying the functions f_0 and f_1 can be viewed as generating commitments for $x \in \mathcal{X}$, which *perfectly hide* the committed values. However, applying the functions $\hat{f}_0$ and $\hat{f}_1$ can only be viewed as partially opening the corresponding commitments, because they only output public keys in $\mathcal{EK}$, leaving certain inherent hard-to-solve problems unresolved (i.e., computing the secret keys).

Equivocability. For a homomorphic NIKE scheme $\Pi = (\text{SETUP}, \text{GEN}, \text{KEY})$ with associated functions $\{f_\sigma, \hat{f}_\sigma\}_{\sigma \in \{0,1\}}$, we say the *equivocability* holds, if there exists a PPT algorithm EXT, s.t., for any $x \in \mathcal{X}$ the following distributions are indistinguishable:

$$\left\{ (x, \mathsf{pek}, \mathsf{ek}, \mathsf{dk}) \;\middle|\; \begin{array}{l} (\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \text{GEN}(\mathsf{pp}); \\ \mathsf{pek} = f_\sigma(x, \mathsf{ek}). \end{array} \right\} \overset{c}{\approx} \left\{ (x, \mathsf{pek}, \mathsf{ek}, \mathsf{dk}) \;\middle|\; \begin{array}{l} (\mathsf{pek}, \mathsf{pdk}) \xleftarrow{\$} \text{GEN}(\mathsf{pp}); \\ (\mathsf{ek}, \mathsf{dk}) = \text{EXT}(x, \mathsf{pek}, \mathsf{pdk}, \hat{f}_\sigma). \end{array} \right\}$$

Equivocability bears resemblance to the *equivocability* of commitment scheme [Bea96]. Regarding pek, with the knowledge of the corresponding secret key pdk, invoking the algorithm EXT can open it to any $x \in \mathcal{X}$.

Intractability. For a homomorphic NIKE scheme $\Pi = (\text{SETUP}, \text{GEN}, \text{KEY})$ with associated functions $\{f_\sigma, \hat{f}_\sigma\}_{\sigma \in \{0,1\}}$, we say the Type-I/II (w.r.t. i/ii) intractability holds, if for any $\mathsf{pp} \xleftarrow{\$} \text{SETUP}(1^\lambda)$, $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \text{GEN}(\mathsf{pp})$, $(\mathsf{pek}, \mathsf{pdk}) \xleftarrow{\$} \text{GEN}(\mathsf{pp})$, $(\mathsf{pek}', \mathsf{pdk}') \xleftarrow{\$} \text{GEN}(\mathsf{pp})$ and PPT algorithm $\mathcal{A}$ defined in Table 1, the success probability of $\mathcal{A}$ is negligible.

Table 1: The two types of Intractability

		$\mathcal{A}$'s Input	$\mathcal{A}$'s Output	$\mathcal{A}$'s Success Condition
Type-I	i	$(\mathsf{pek}, \mathsf{pek}')$	(x, k)	$\mathsf{k} = \mathbf{Key}(\hat{f}_0(x, \mathsf{pek}), \hat{f}_1(x, \mathsf{pek}'))$
	ii	pek	$(\mathsf{pek}', (x_0, \mathsf{k}_0), (x_1, \mathsf{k}_1))$	$x_0 \neq x_1$; $\mathsf{k}_0 = \mathbf{Key}(\hat{f}_\sigma(x_0, \mathsf{pek}), \hat{f}_{1-\sigma}(x_0, \mathsf{pek}'))$; $\mathsf{k}_1 = \mathbf{Key}(\hat{f}_\sigma(x_1, \mathsf{pek}), \hat{f}_{1-\sigma}(x_1, \mathsf{pek}'))$.
Type-II	i	$(\mathsf{ek}, \mathsf{pek})$	(x, k)	$\mathsf{k} = \mathbf{Key}(\mathsf{ek}, \hat{f}_\sigma(x, \mathsf{pek}))$
	ii	ek	$(\mathsf{pek}, (x_0, \mathsf{k}_0), (x_1, \mathsf{k}_1))$	$x_0 \neq x_1$ $\mathsf{k}_0 = \mathbf{Key}(\mathsf{ek}, \hat{f}_\sigma(x_0, \mathsf{pek}))$; $\mathsf{k}_1 = \mathbf{Key}(\mathsf{ek}, \hat{f}_\sigma(x_1, \mathsf{pek}))$.

Note that if using a NIKE as the foundation for PAKE protocols, a basic security principle to be followed is that: Only when the both parties use the same password, they can negotiate the same NIKE shared key. The

security property *Intractability* guarantees that the target shared NIKE key is difficult to compute in several typical application scenarios. In particular, Type-I Intractability deals with the type of PAKE protocols where both parties need to commit their password, while Type-II Intractability deals with the type of PAKE protocols where only one party needs to commit its password. Furthermore, Type-I/II(i) Intractability is employed to address the case that a passive adversary cannot launch successful offline-guessing attacks. Type-I/II(ii) Intractability is utilized to address the case that an active adversary can only validate a single password guess within one session.

2.3 Instantiations of homomorphic NIKE with associated functions

PPK [Mac02] and SPAKE2 [AP05] share the same underlying homomorphic NIKE yet with different associated functions as shown in Fig. 5.

The underlying homomorphic NIKE scheme

$\underline{\text{SETUP}(1^\lambda)}$:	$\underline{\text{GEN}(\mathsf{pp})}$: $//\mathcal{EK} := \mathbb{G}$ and $\mathcal{DK} := \mathbb{Z}_q$	$\underline{\text{KEY}(\mathsf{dk}, \mathsf{ek}')}$:
1. $(\mathbb{G}, q, g) \xleftarrow{\$} \mathcal{G}en(1^\lambda)$;	1. $a \xleftarrow{\$} \mathbb{Z}_q$;	1. Phrase $\mathsf{dk} := a$;
2. **Output** $\mathsf{pp} := (\mathbb{G}, q, g)$.	2. **Output** $(\mathsf{ek}, \mathsf{dk}) := (g^a, a)$.	2. **Output** $\mathsf{k} := (\mathsf{ek}')^a$.

The associated functions derived from PPK [Mac02]

$(\mathcal{H}_0 : \mathcal{X} \to \mathcal{EK}$ and $\mathcal{H}_1 : \mathcal{X} \to \mathcal{EK})$

$$\begin{cases} f_0(x, \mathsf{ek}) := \mathsf{ek} \cdot \mathcal{H}_0(x); & \hat{f}_0(x, \mathsf{pek}) := \mathsf{pek}/\mathcal{H}_0(x); \\ f_1(x, \mathsf{ek}) := \mathsf{ek} \cdot \mathcal{H}_1(x); & \hat{f}_1(x, \mathsf{pek}) := \mathsf{pek}/\mathcal{H}_1(x). \end{cases}$$

The associated functions derived from SPAKE2 [AP05]

$(M, N \in \mathcal{EK})$

$$\begin{cases} f_0(x, \mathsf{ek}) := \mathsf{ek} \cdot M^x; & \hat{f}_0(x, \mathsf{pek}) := \mathsf{pek}/M^x; \\ f_1(x, \mathsf{ek}) := \mathsf{ek} \cdot N^x; & \hat{f}_1(x, \mathsf{pek}) := \mathsf{pek}/N^x. \end{cases}$$

Figure 5: The homomorphic NIKE with asscoatied functions derived from PPK [Mac02] and SPAKE2 [AP05]

Theorem 1. *If the computational Diffie-Hellman (CDH) assumption holds, the NIKE scheme in Fig. 5 is OW-secure and homomorphic.*

Theorem 2. *If $\mathcal{H}_0$ and $\mathcal{H}_1$ are modeled as ROs, the perfect hiding, equivocability and intractability properties hold for the homomorphic NIKE scheme with associated functions derived from PPK in Fig. 5, simultaneously.*

Theorem 3. *If M and N are treated as CRS, The perfect hiding, equivocability and intractability properties hold for the homomorphic NIKE scheme with associated functions derived from SPAKE2 in Fig. 5, simultaneously.*

We postpone the proofs to the above theorems to Appendix B.

3 Impossibility Result regarding Global CRS

In this section, we will show that global CRS is *not* sufficient to realize $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$, $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}}$, $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ or $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$. The result refers to non-trivial protocols. A protocol is non-trivial, if it is guaranteed that two honest parties agree on matching session keys at the conclusion of a protocol execution (except perhaps with negligible probability), provided that *(i)* both parties use the same password, and *(ii)* the adversary forwards all messages between them without modifying or inserting any messages.

Theorem 4. *There exists no non-trivial protocol that UC-realizes the functionality $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$, $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}}$, $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ or $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$ in the $\mathcal{G}_{\text{CRS}}$-hybrid world.*

The proof extends Canetti et al.'s impossibility proof in [CHK+05]. There, Canetti et al. prove that it is impossible to construct a non-trivial protocol for UC-realizing the PAKE functionality in the plain model; but here, we will show that even certain setup such as a global CRS are given, it is still impossible to construct a non-trivial UC-secure protocol for the PAKE functionality. More details can be found in Appendix C.1.

4 Our construction in the UC framework

In this section, we first present a formal description of our PAKE protocol. Then we prove that it is UC-secure with a fine-grained CoR-setup.

4.1 Formal description of the construction

Our construction will use the following ingredients:

- A homomorphic NIKE scheme $\Pi = (\text{SETUP}, \text{GEN}, \text{KEY})$, where $\mathcal{EK}$, $\mathcal{DK}$ and $\mathcal{K}$ are the sets of public keys, secret keys and session keys. In addition, this scheme is equipped with associated functions $f_\sigma : \mathcal{X} \times \mathcal{EK} \to \mathcal{EK}$ and $\hat{f}_\sigma : \mathcal{X} \times \mathcal{EK} \to \mathcal{EK}$ for $\sigma \in \{0,1\}$.
- A sub-protocol $\text{Prot}_{\text{PAKE}}$ that UC-realizes $\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}$ in the $\mathcal{F}_{\text{CRS}}$-hybrid world.
- Three hash functions: $\mathcal{H}_1 : \{0,1\}^* \to \{0,1\}^{2\lambda}$, $\mathcal{H}_2 : \{0,1\}^* \to \{0,1\}^{2\lambda}$ and $\mathcal{H}_3 : \{0,1\}^* \to \{0,1\}^{6\lambda}$.

Public parameters. A setup phase is required to assign the security parameter λ, the password space $\mathcal{D} \subset \mathcal{X}$, $\mathsf{pp} \xleftarrow{\$} \text{SETUP}(1^\lambda)$ and INHERITED PARAMETER that consists of parameters inherited from $\text{Prot}_{\text{PAKE}}$. Jumping ahead, in our analysis, INHERITED PARAMETER will be treated as the following different cases:

(1) INHERITED PARAMETER is treated as a local parameter when $(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$ are assumed as collision resistant hash functions;
(2) INHERITED PARAMETER is treated as a global parameter when $(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$ are modeled as ROs.

Remark 1. *In Case-(1), all the three hash functions should be collision-resistant. In particular, for any input $x \in \{0,1\}^*$, the output $y_1|y_2|y_3$ computed from $\mathcal{H}_3(x)$ will be used as three individual values, where $|y_i| = \lambda$ for $i \in \{1,2,3\}$. For security, we necessitate a more special collision-resistance property, i.e., for any $x' \neq x$ and $y_1'|y_2'|y_3' = \mathcal{H}_3(x')$, it should hold that $y_i \neq y_i'$ for $i \in \{1,2,3\}$. Note that this property is not hard to achieve: Simply choose a collision-resistant hash function $\tilde{\mathcal{H}} : \{0,1\}^* \to \{0,1\}^{2\lambda}$, and set $\mathcal{H}_3(x) = \tilde{\mathcal{H}}(1|x)|\tilde{\mathcal{H}}(2|x)|\tilde{\mathcal{H}}(3|x)$.*

In Case-(2), $\mathcal{H}_3$ should be modeled as a single RO. The same property can also be achieved. Let the simulator internally use three different ROs. For a new query x, treat it as three queries on these internal ROs with the same input x.

Protocol execution. Assume two parties, $\mathcal{P}_i$ and $\mathcal{P}_j$ initially holding their inputs, $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{client})$ and $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}_j, \mathsf{server})$ received from the environment, respectively. They execute as shown in Fig. 6 to negotiate a session key. In particular, during the course, they need to jointly execute the sub-protocol $\text{Prot}_{\text{PAKE}}$, where the party $\mathcal{P}_j$ first prepares its input, thus playing the role of client.

If the protocol is analyzed in the $\mathcal{F}_{\text{RO}}$-hybrid world, all local hash computations should be replaced by querying the functionality $\mathcal{F}_{\text{RO}} = \{\mathcal{F}_{\text{RO}i}\}_{i\in[3]}$. In particular, *(i)* $\mathcal{H}_1$ (resp., $\mathcal{H}_2$) computations are replaced by querying $\mathcal{F}_{\text{RO}_1}$ (resp., $\mathcal{F}_{\text{RO}_2}$) parameterized with domain $\{0,1\}^*$ and range $\{0,1\}^\lambda$; *(ii)* $\mathcal{H}_3$ computations are replaced by querying $\mathcal{F}_{\text{RO}_3}$ parameterized with domain $\{0,1\}^*$ and range $\{0,1\}^{3\lambda}$. Abusing notations, in Fig. 6, we still use $\mathcal{H}_i(x)$ to denote the answer of querying $\mathcal{F}_{\text{RO}i}$ on x.

4.2 The security analysis

The protocol shown in Fig. 6 is UC-secure PAKE with a fine-grained CoR-setup. In particular, we have the following theorem.

Theorem 5. *Suppose $|\mathcal{D}| = \mathsf{poly}(\lambda)$. Given the following building blocks:*

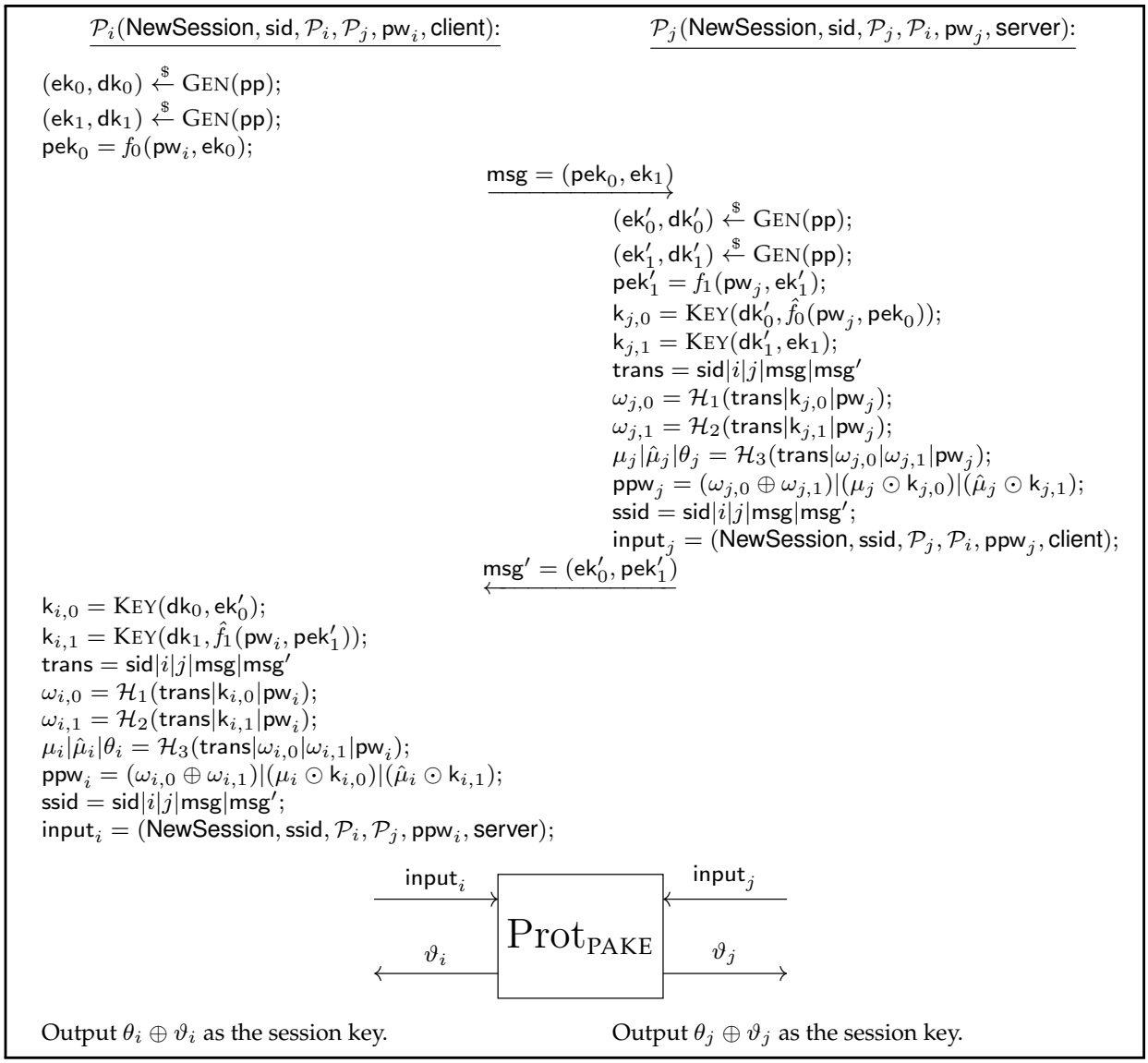


Figure 6: Our PAKE protocol in the UC framework

- *A homomorphic NIKE scheme* $\Pi = (\textsc{Setup}, \textsc{Gen}, \textsc{Key})$ *with associated functions* $\{f_\sigma, \hat{f}_\sigma\}_{\sigma\in\{0,1\}}$*, where the prefect hiding property holds; In addition, the scheme satisfies the Equivocability and Type-II Intractability in the RO model.*
- *A sub-protocol* $\mathrm{Prot}_{\mathrm{PAKE}}$ *that UC-realizes* $\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}$ *in the* $\mathcal{F}_{\mathrm{CRS}}$*-hybrid world, where* $\{0,1\}^{6\lambda} \subseteq \hat{\mathcal{D}}$*; In addition, the sub-protocol* $\mathrm{Prot}_{\mathrm{PAKE}}$ *satisfies the correctness property in the* $\mathcal{G}_{\mathrm{CRS}}$*-hybrid world.*
- *Three hash functions* $\mathcal{H}_1$*,* $\mathcal{H}_2$ *and* $\mathcal{H}_3$*.*

The protocol in Fig. 6 is secure in the UC framework, in particular:

(1) It UC-realizes $\mathcal{F}_{\mathrm{mPAKE}}^{\mathcal{D}}$ *in the* $\mathcal{F}_{\mathrm{CRS}}$*-hybrid world when* $\mathcal{H}_1$*,* $\mathcal{H}_2$ *and* $\mathcal{H}_3$ *are sampled from collision-resistant hash families;*

(2) It UC-realizes $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$[8] *in the* $\{\mathcal{G}_{\mathrm{CRS}}, \mathcal{F}_{\mathrm{RO}}\}$*-hybrid world by modeling* $\mathcal{H}_1$*,* $\mathcal{H}_2$ *and* $\mathcal{H}_3$ *as ROs.*

Proof. In Case-(1) (resp.,Case-(2)), we will construct a simulator $\mathcal{S}$, which interacts with the functionality $\mathcal{F}_{\mathrm{mPAKE}}^{\mathcal{D}}$ (resp., $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$) and an environment $\mathcal{Z}$, and we show that for any efficient environment $\mathcal{Z}$ and

[8]Jumping ahead, in the next section (Sec. 5.2), we will show that $\mathcal{F}_{\mathrm{mPAKE}}^{\mathcal{D}}$ is also achieveable in Case-(2).

any real-world adversary $\mathcal{A}$, $\mathcal{Z}$ cannot distinguish the ideal-world execution, created by $\mathcal{S}$ in interaction with $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ (resp., $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$), from the real-world execution, created by $\mathcal{A}$ in interaction with real-world parties executing protocol in Fig. 6. For simplicity, assume the adversary $\mathcal{A}$is a "dummy" adversary, solely relaying all messages and computations to $\mathcal{Z}$.

Proof sketch in Case-(1). Since we assume that sub-protocol $\text{Prot}_{\text{PAKE}}$ realizes functionality $\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}$, the real-world adversary can be without loss of generality replaced by a hybrid-world adversary, where the sub-protocol $\text{Prot}_{\text{PAKE}}$ executed by the real-world parties within the protocol in Fig. 6 is replaced by the ideal functionality $\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}$. In our proof, we assume that $\mathcal{A}$ operates within this $\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}$-hybrid world, albeit for terminological convenience, we persist in referring to $\mathcal{A}$ as a "real-world" adversary. We implicitly assume that $\perp$ will be accepted but treated as a wrong password guess by the functionalities.

When $\mathcal{A}$ corrupts the party $\mathcal{P}_j$ that plays the role of server:

The simulator $\mathcal{S}$ generates the message $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ using a specified value $\overline{\mathsf{pw}} \in \mathcal{X} \setminus \mathcal{D}$. The prefect hiding property of the underlying homomorphic NIKE with associated functions ensures that $\mathcal{A}$ cannot observe the change. Upon receiving the message $\mathsf{msg}' = (\mathsf{ek}'_1, \mathsf{pek}'_1)$ and $(\mathsf{TestPwd}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{ppw}^* = \eta_1|\eta_2|\eta_3)$ from $\mathcal{A}$, $\mathcal{S}$ can extract $\mathcal{A}$'s password guess by trying all candidate passwords in $\mathcal{D}$ to find pw^* that makes the following equations hold simultaneously:

$$\begin{cases} \mathsf{k}^*_{i,1} & = \text{KEY}(\mathsf{dk}_1, \hat{f}_1(\mathsf{pw}^*, \mathsf{pek}'_1)); \\ \omega^*_{i,1} & = \mathcal{H}_2(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}^*_{i,1}|\mathsf{pw}^*); \\ \omega^*_{i,0} & = \eta_1 \oplus \omega^*_{i,1}; \\ \mu^*_i|\hat{\mu}^*_i|\theta^*_i & = \mathcal{H}_3(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega^*_{i,0}|\omega^*_{i,1}|\mathsf{pw}^*); \\ \hat{\mu}^*_i & = \eta_3 \odot \mathsf{k}^*_{i,1}; \\ \mathsf{k}^*_{i,0} & = \eta_2 \odot \mu^*_i; \\ \omega^*_{i,0} & = \mathcal{H}_1(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}^*_{i,0}|\mathsf{pw}^*). \end{cases} \tag{1}$$

Since all computations included in Equations 1 are deterministic and $\mathcal{H}_1, \mathcal{H}_2$ and $\mathcal{H}_3$ are collision-resistant, at most one valid value pw^* can be found. If not found such value, then set $\mathsf{pw}^* = \perp$. Then, $\mathcal{S}$ can send $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ to $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$. Upon receiving $(\mathsf{NewKey}, \mathsf{ssid}, \mathcal{P}_i, \vartheta^*)$ from $\mathcal{A}$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$, where $\mathsf{SK}^* = \vartheta^* \oplus \theta^*_i$ if $\mathsf{pw}^* \neq \perp$, and SK^* is randomly chosen otherwise. It can be verified that if pw^* is a correct guess, $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ will make the honest party $\mathcal{P}_i$ outputs the session key determined by $\mathcal{A}$; otherwise, a random session key is output.

When $\mathcal{A}$ corrupts the party $\mathcal{P}_i$ that plays the role of client:

Similar proof idea is also applied here. After receiving the message $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$, $\mathcal{S}$ generates $\mathsf{msg}' = (\mathsf{ek}'_1, \mathsf{pek}'_1)$ using the specified value $\overline{\mathsf{pw}}$. Upon receiving $(\mathsf{TestPwd}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{ppw}^* = \eta_1|\eta_2|\eta_3)$ from $\mathcal{A}$, try all candidate passwords in $\mathcal{D}$ to find pw^* that makes the following equations hold simultaneously; otherwise set $\mathsf{pw}^* = \perp$.

$$\begin{cases} \mathsf{k}^*_{j,0} & = \text{KEY}(\mathsf{dk}'_0, \hat{f}_0(\mathsf{pw}^*, \mathsf{pek}_0)); \\ \omega^*_{j,0} & = \mathcal{H}_1(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}^*_{j,0}|\mathsf{pw}^*); \\ \omega^*_{j,1} & = \eta_1 \oplus \omega^*_{j,0}; \\ \mu^*_j|\hat{\mu}^*_j|\theta^*_j & = \mathcal{H}_3(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega^*_{j,0}|\omega^*_{j,1}|\mathsf{pw}^*); \\ \hat{\mu}^*_j & = \eta_3 \odot k^*_{j,0}; \\ \mathsf{k}^*_{j,1} & = \eta_2 \odot \mu^*_j; \\ \omega^*_{j,1} & = \mathcal{H}_2(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}^*_{j,1}|\mathsf{pw}^*). \end{cases} \tag{2}$$

Then, $\mathcal{S}$ can send $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_j, \mathsf{pw}^*)$ to $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$. Upon receiving $(\mathsf{NewKey}, \mathsf{ssid}, \mathcal{P}_j, \vartheta^*)$ from $\mathcal{A}$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$, where $\mathsf{SK}^* = \vartheta^* \oplus \theta^*_j$ if $\mathsf{pw}^* \neq \perp$, and SK^* is randomly chosen otherwise.

The detailed proof will be found in Appendix C.2.

Proof sketch in Case-(2). For notational simplicity, in the next paragraph, we will employ the following two terminologies, which correspond to the important computations to detect whether $\mathcal{A}$ provides a valid input in its $\mathcal{F}_{\text{RO}}$ queries:

- k $\triangle$-*satisfies* $(\mathsf{pw}, \mathsf{dk}, \mathsf{pek}, f_\sigma)$: $\mathsf{k} = \textsc{Key}(\mathsf{dk}, \hat{f}_\sigma(\mathsf{pw}, \mathsf{pek}))$;
- k $\triangledown$-*satisfies* $(\mathsf{pw}, \mathsf{pdk}, \mathsf{ek}, f_\sigma)$: $\mathsf{k} = \textsc{Key}(\mathsf{dk}^*, \mathsf{ek})$ with $\mathsf{dk}^* = \textsc{Ext}(\mathsf{pw}, \mathsf{pdk}, \hat{f}_\sigma)$.

where the algorithm $\textsc{Ext}$ can be invoked because the Equivocability of the underlying NIKE with associated functions holds in the RO model.

When $\mathcal{A}$ corrupts the party $\mathcal{P}_j$ that plays the role of server:

$\mathcal{S}$ first generates $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_0)$ with $(\mathsf{pek}_0, \mathsf{pdk}_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $(\mathsf{ek}_1, \mathsf{dk}_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$. After receiving the message $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$, $\mathcal{S}$ distinguishes the following cases:

- In the case that $\mathcal{A}$ has queried $\mathcal{F}_{\text{RO}_2}$ on "$\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}$" such that k_1 $\triangle$-*satisfies* $(\mathsf{pw}, \mathsf{dk}_1, \mathsf{pek}'_1, \hat{f}_1)$. Given pek'_1, the Type-II(ii) Intractability ensures that $\mathcal{A}$ can only makes one such valid query. In this case, $\mathcal{S}$ sends $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw})$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$. Due to the equivocability of underlying homomorphic NIKE with associated functions, $\mathcal{S}$ can compute the secret key of $\hat{f}_0(\mathsf{pw}, \mathsf{pek}_0)$ using pdk_0. Since $\mathcal{S}$ also knows dk_1 and pw, it can proceed as if pw was the input. Upon completing the protocol and obtaining a session key SK^*, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$.
- Otherwise, $\mathcal{S}$ executes the sub-protocol $\text{Prot}_{\text{PAKE}}$ using a random value as input, which is recorded to program all subsequent $\mathcal{F}_{\text{RO}}$ queries. If, before completing the execution, $\mathcal{A}$ has made a valid query to $\mathcal{F}_{\text{RO}_2}$ with a password guess pw, $\mathcal{S}$ sends $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw})$ followed by $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$, where SK^* is computed using pw as input based on the protocol description. Otherwise, $\mathcal{S}$ sends $(\mathsf{RegisterTest}, \mathsf{sid}, \mathcal{P}_i)$ followed by $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$, where SK^* is randomly chosen. In particular, for the latter case, when $\mathcal{A}$ later makes a valid query to $\mathcal{F}_{\text{RO}_3}$ (that requires a valid query to $\mathcal{F}_{\text{RO}_2}$ in advance) with a password guess pw, $\mathcal{S}$ sends $(\mathsf{LateTestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw})$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$ and uses the replied session key to program the answer for the corresponding query to $\mathcal{F}_{\text{RO}_3}$.

It can be verified that if $\mathcal{A}$ makes correct password guess before the protocol completes with a session key, $\mathcal{F}^{\mathcal{D}}_{\text{le-PAKE}}$ will make the honest party $\mathcal{P}_i$ outputs the session key determined by $\mathcal{S}$; otherwise, a random session key is output. For the case that $\mathcal{A}$ makes a correct late password guess, it can compute to the same session key as the honest party.

When $\mathcal{A}$ corrupts the party $\mathcal{P}_i$ that plays the role of client:

Similar proof idea is also applied here. After receiving the message $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$, $\mathcal{S}$ generates $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ with $(\mathsf{ek}'_0, \mathsf{dk}'_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $(\mathsf{pek}'_1, \mathsf{pdk}'_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and executes the sub-protocol $\text{Prot}_{\text{PAKE}}$ using a random value as input, which is recorded to program all subsequent $\mathcal{F}_{\text{RO}}$ queries.

- If, before completing the execution of $\text{Prot}_{\text{PAKE}}$, $\mathcal{A}$ has queried $\mathcal{F}_{\text{RO}_1}$ on "$\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}$" such that k_0 $\triangle$-*satisfies* $(\mathsf{pw}, \mathsf{dk}'_0, \mathsf{pek}_0, \hat{f}_0)$. Similarly, given pek_0, the Type-II(ii) Intractability ensures that $\mathcal{A}$ can only makes one such valid query. In this case, $\mathcal{S}$ sends $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_j, \mathsf{pw})$ followed by $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$, where SK^* is computed using pw as input based on the protocol description.
- Otherwise, $\mathcal{S}$ sends $(\mathsf{RegisterTest}, \mathsf{sid}, \mathcal{P}_i)$ followed by $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$ in order, where SK^* is randomly chosen. In particular, for the latter case, when $\mathcal{A}$ later makes a valid query to $\mathcal{F}_{\text{RO}_3}$ (that requires a valid query to $\mathcal{F}_{\text{RO}_1}$ in advance) with a password guess pw, $\mathcal{S}$ sends $(\mathsf{LateTestPwd}, \mathsf{sid}, \mathcal{P}_j, \mathsf{pw})$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$ and uses the replied session key to program the answer for the corresponding query to $\mathcal{F}_{\text{RO}_3}$.

The detailed proof is given in Appendix C.3. □

5 Extensions and discussions

In this section, we will present three important extensions and discuss the practicality of our main results presented in Sec. 4.

5.1 Extending the password space

Now, we show how to extend the password space. Given a sub-protocol $\mathrm{Prot}_{\mathrm{UNIT}}$ that UC-realizes $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$, we can obtain a protocol that UC-realizes $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}^n}$ for $n \in \mathbb{N}$. We achieve this goal by: First, splitting a long password in $\mathcal{D}^n$ into n shorter passwords in $\mathcal{D}$; Next, invoking n instances of the sub-protocol $\mathrm{Prot}_{\mathrm{UNIT}}$ with each shorter password as input; Finally, XORing all output session keys.

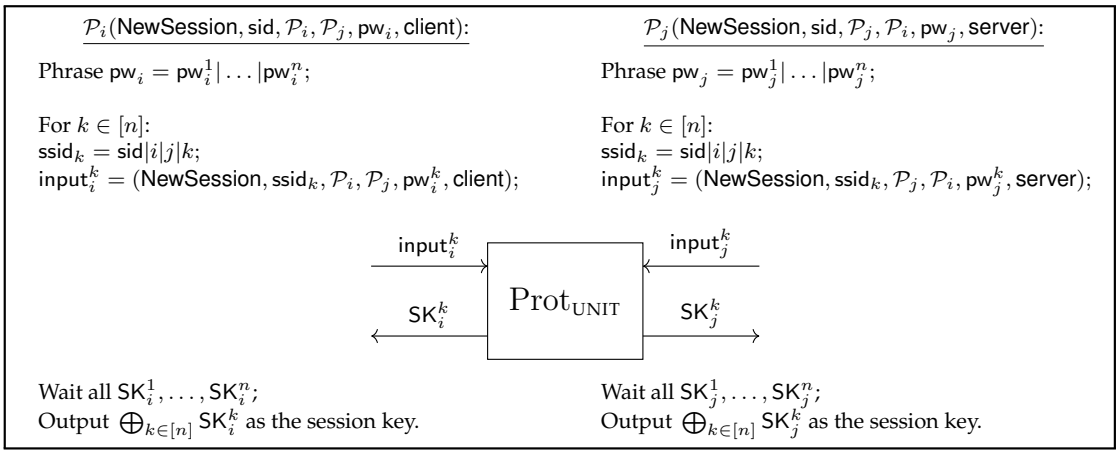


Figure 7: Our password space extending protocol

Protocol execution. Assume two parties, $\mathcal{P}_i$ and $\mathcal{P}_j$ initially holding their inputs, $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{client})$ and $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}_j, \mathsf{server})$ received from the environment, respectively. In particular, $\mathsf{pw}_i, \mathsf{pw}_j \in \mathcal{D}^n$. They execute as shown in Fig. 7 to negotiate a session key.

Theorem 6. *Let $n \in \mathbb{N}$. If the sub-protocol $\mathrm{Prot}_{\mathrm{UNIT}}$ UC-realizes $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$, then the protocol in Fig. 7 UC-realizes $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}^n}$. Meanwhile, if the sub-protocol $\mathrm{Prot}_{\mathrm{UNIT}}$ UC-realizes $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$, then the protocol UC-realizes $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}^n}$.*

Proof. We construct a simulator $\mathcal{S}$ that interacts with the functionality $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}^n}$ and an environment $\mathcal{Z}$. Similarly, since we assume that the sub-protocol $\mathrm{Prot}_{\mathrm{UNIT}}$ UC-realizes the functionality $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$, in our proof, we assume that $\mathcal{A}$ operates within the $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$-hybrid world. For simplicity, we persist in referring to $\mathcal{A}$ as a "real-world" adversary and denote functionality $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}^n}$ throughout the proof as simply $\mathcal{F}$, assume the adversary $\mathcal{A}$ is a "dummy" adversary, solely relaying all messages and computations to $\mathcal{Z}$. Since there are no interactions between parties in the real world, the simulator $\mathcal{S}$ only needs to simulate the interactions between $\mathcal{A}$ and the functionality $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$. In particular, the simulator $\mathcal{S}$ is constructed as shown in Fig. 8. Proof of indistinguishability between the real world and ideal world can be found in Appendix D.

The proof idea outlined above can similarly be applied to show that if the sub-protocol $\mathrm{Prot}_{\mathrm{UNIT}}$ UC-realizes $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$, then the protocol in Fig. 8 UC-realizes $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}^n}$. The necessary modifications involve removing the parts that correspond to RegisterTest and LateTestPwd that are not included in the functionalities $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$ and $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}^n}$.

□

5.2 Upgrading to the standard UC security

The protocol shown in Fig. 6 only UC-realizes the functionality $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$, when the RO setup is functional but the CRS setup falls back to gCRS. If combining with the password space extending protocol shown in Fig. 8, it only achieves $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}^n}$. One might think our results are limited. Actually, we can easily achieve the standard UC-security (i.e., $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}^n}$) by investigating the internal structure of $\mathrm{Prot}_{\mathrm{PAKE}}$ and slightly modifying the simulator.

Recall the main difference between $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$ and $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ is that the former allows the ideal-world adversary to postpone its password guess until a session is complete, i.e., a session key is computed. To

On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role})$ from $\mathcal{F}$:
For $k \in [n]$, set $\mathsf{ssid}_k := \mathsf{sid}|i|j|k$, store a record $\langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \bot, \bot \rangle$ marked fresh and send $(\mathsf{NewSession}, \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role})$ to $\mathcal{A}$.

On $(\mathsf{TestPwd}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{pw}}_i^k)$ from $\mathcal{A}$ on behalf of a corrupted party $\mathcal{P}_j$:
If $\exists \langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \bot, \bot \rangle$ that is marked with fresh, update it as $\langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \widehat{\mathsf{pw}}_i^k, \bot \rangle$.

On $(\mathsf{RegisterTest}, \mathsf{ssid}_k, \mathcal{P}_i)$ from $\mathcal{A}$ on behalf of a corrupted party $\mathcal{P}_j$:
If $\exists \langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \bot, \bot \rangle$ that is marked with fresh, update it as $\langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \top, \bot \rangle$.

On $(\mathsf{NewKey}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{SK}}_i^k)$ from $\mathcal{A}$ on behalf of a corrupted party $\mathcal{P}_j$:
If $\exists \langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \mathsf{pw}_i^k, \bot \rangle$ that is not marked completed:

- If $\mathsf{pw}_i^k \neq \bot \wedge \mathsf{pw}_i^k \neq \top$, update it as $\langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \mathsf{pw}_i^k, \widehat{\mathsf{SK}}_i^k \rangle$.
- Else if $\mathsf{pw}_i^k = \top$, update it as $\langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \top, \top \rangle$.
- Otherwise, update it as $\langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \bot, \mathsf{SK}_i^k \rangle$, where $\mathsf{SK}_i^k \xleftarrow{\$} \{0,1\}^\lambda$.

In all cases, mark this record with completed.

On $(\mathsf{LateTestPwd}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{pw}}_i^k)$ from $\mathcal{A}$ on behalf of a corrupted party $\mathcal{P}_j$:
If $\exists \langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \top, \top \rangle$, do as follows:

- If for all $k' \in [n]$ with $k' \neq k$, $\mathsf{pw}_i^{k'} \neq \top \wedge \mathsf{pw}_i^{k'} \neq \bot$ holds, then set $\mathsf{pw}^* := \mathsf{pw}_i^1 | \ldots | \mathsf{pw}_i^n$, where $\mathsf{pw}_i^k := \widehat{\mathsf{pw}}_i^k$. In addition, send $(\mathsf{LateTestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ to $\mathcal{F}$. After receving a reply SK, set $\mathsf{SK}_i^k := \mathsf{SK} \bigoplus_{k' \in [n] \setminus \{k\}} \mathsf{SK}_i^{k'}$.
- Otherwise, randomly choose $\mathsf{SK}_i^k \xleftarrow{\$} \{0,1\}^\lambda$.

In both cases, send SK_i^k to $\mathcal{A}$ and update this record as $\langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \mathsf{pw}^*, \mathsf{SK}_i^k \rangle$.

Extracting $\mathcal{A}$'s password guess:
If $\exists \langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \mathsf{pw}_i^k \neq \bot, * \rangle$ for all $k \in [n]$, where $\mathsf{ssid}_k = \mathsf{sid}|i|j|k$:

- If $\mathsf{pw}_i^k \neq \top$ for all $k \in [n]$, set $\mathsf{pw}^* := \mathsf{pw}_i^1 | \ldots | \mathsf{pw}_i^n$ and send $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ to $\mathcal{F}$.
- Otherwise, send $(\mathsf{RegisterTest}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{F}$.

Computing the session key:
If $\exists \langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, *, \mathsf{SK}_i^k \neq \bot \rangle$ for all $k \in [n]$, where $\mathsf{ssid}^k = \mathsf{sid}|i|j|k$:

- If $\mathsf{SK}_i^k \neq \top$ for all $k \in [n]$, set $\mathsf{SK}^* := \bigoplus_{k \in [n]} \mathsf{SK}_i^k$.
- Otherwise, randomly choose $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^\lambda$.

In both cases, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}$.

Figure 8: The simulator for password space extending protocol

have the standard $\mathcal{F}_{\mathrm{mPAKE}}^{\mathcal{D}}$ security, the simulator should always be able to extract the adversary's input before the session key is determined. We notice that both parties should contribute randomness to the output in the sub-protocol $\mathrm{Prot}_{\mathrm{PAKE}}$, however, the current black-box calls to $\mathrm{Prot}_{\mathrm{PAKE}}$ hinder to use its nice features. But these do help!

At a high level, CRS-based PAKEs [KOY01, GL03, JG04, CHK+05, ACP09, KV09, GK10, BBC+13, ABP15], are based on PKE with associated Smooth Projective Hash Function (SPHF, Appendix A.6) with the following design:

- Each party sends a ciphertext encapsulating the party's password to and receives a projection key from its peer party.
- Given a specific ciphertext and projection key, if the two parties share the same input password, they can use different ways to compute the same SPHF hash, using either the witness (i.e., the randomness) used to generate the ciphertext, or the secret hashing key. The resulted SPHF value is used as a piece of material to derive the session key.

Here, the smoothness of the underlying SPHF is essential: For an adversarially generated ciphertext, the corresponding SPHF hash value is statistically indistinguishable from a random one. Consequently, the

resulting session key is also statistically indistinguishable from a random session key.

Applying the aforementioned idea into the construction of the simulator in Fig. 14, we immediately have that: If an adversary has never queried $\mathcal{F}_{\text{RO}}$ on the correct value before the execution of the sub-protocol $\text{Prot}_{\text{PAKE}}$, the ciphertext generated by the adversary during the execution of $\text{Prot}_{\text{PAKE}}$ would not contain the value ppw used by the honest party, except for negligible probability. In this case, the simulator can safely send $(\mathsf{TestPwd}, *, *, \perp)$ and $(\mathsf{NewKey}, *, *, \mathsf{SK}^*)$ to $\mathcal{F}$, which yields a random session key for the honest party. By doing so, the simulator avoids to use the interfaces RegisterTest and LateTestPwd. We can then obtain a proof for achieving $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ instead of $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$.

5.3 Achieving post-quantum security

If we instantiate the sub-protocol $\text{Prot}_{\text{PAKE}}$ using a post-quantum secure one, our proposed protocol in Fig. 6 is immediately post-quantum secure when the CRS setup is functional. This assurance remains valid regardless of the underlying assumptions used to instantiate the underlying homomorphic NIKE scheme. For an honest party acting as the client, the only message sent before executing $\text{Prot}_{\text{PAKE}}$ is $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$. Importantly, this message reveals no information about the input password, even against an adversary with unlimited computational power. In particular, pek_0 hides the input password perfectly, while ek_1 is independently from the input password. The same applies to the server side. These facts indicate that even if the homomorphic NIKE scheme is not post-quantum secure, an adversary would still need to guess the password to compromise the post-quantum secure sub-protocol $\text{Prot}_{\text{PAKE}}$. Therefore, we can conclude that the entire protocol is post-quantum secure.

However, achieving post-quantum security when the RO setup is functional seems not easy, as it requires a post-quantum secure homomorphic NIKE scheme. Isogeny-based NIKE schemes [DKS18,DHK+22, Ler22] show promise in this area. However, we currently do not know how to instantiate such a scheme with lattice-based techniques due to challenges in handling errors. Nonetheless, our design is fairly generic, and we believe it can be adapted to the lattice-based setting — an interesting open question for future work.

5.4 Practicality Considerations

We now discuss the practicality of our protocol. While it incurs some overhead, it is less efficient than existing PAKE protocols based on number-theoretic assumptions but still more efficient than those in the plain model.

Let's examine the internal structure of the two sub-protocols. As mentioned in the introduction, sub-protocol Π_{RO} can be seen as a combination of two Half RO-PAKEs. The term "Half" does not imply reduced overhead; thus, the overhead of Π_{RO} is roughly double that of existing RO-based PAKEs. Sub-protocol Π_{CRS}, on the other hand, must handle a password space of at least $\{0,1\}^{6\lambda}$. In existing CRS-based PAKEs, the password space is linked to the message space of the underlying encryption scheme. Therefore, selecting appropriate parameters is crucial, as it can lead to parameter expansion in the underlying schemes.

If we instantiate the sub-protocol $\text{Prot}_{\text{PAKE}}$ using the DDH-based one-round PAKE protocol [KV11] and assume $\lambda = 128$, our design would require an elliptic curve of prime order 768 bits. In this case, the ciphertext size and computation cost for the CRS-based sub-protocol expand by about 3 and 9 times, respectively. However, this remains practical and can be further optimized using generalized Mersenne numbers (GMN) with low Hamming weight. More importantly, this is the first PAKE to achieve UC security with fine-grained CoR-setup. We leave further performance improvements as future work.

Acknowledgements

The third author extends his gratitude to Jonathan Katz and Adam Groce for numerous valuable discussions during his time as a postdoctoral researcher at the University of Maryland *over a decade ago*.

References

[ABB+13] Michel Abdalla, Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, and David Pointcheval. SPHF-friendly non-interactive commitments. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part I*, volume 8269 of *LNCS*, pages 214–234. Springer, Heidelberg, December 2013.

[ABB+20] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 278–307. Springer, Heidelberg, August 2020.

[ABP15] Michel Abdalla, Fabrice Benhamouda, and David Pointcheval. Public-key encryption indistinguishable under plaintext-checkable attacks. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 332–352. Springer, Heidelberg, March / April 2015.

[ACCP08] Michel Abdalla, Dario Catalano, Céline Chevalier, and David Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In Tal Malkin, editor, *CT-RSA 2008*, volume 4964 of *LNCS*, pages 335–351. Springer, Heidelberg, April 2008.

[ACP09] Michel Abdalla, Céline Chevalier, and David Pointcheval. Smooth projective hashing for conditionally extractable commitments. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 671–689. Springer, Heidelberg, August 2009.

[AFP05] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, Heidelberg, January 2005.

[AHH21] Michel Abdalla, Björn Haase, and Julia Hesse. Security analysis of CPace. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part IV*, volume 13093 of *LNCS*, pages 711–741. Springer, Heidelberg, December 2021.

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Heidelberg, February 2005.

[BBC+13] Fabrice Benhamouda, Olivier Blazy, Céline Chevalier, David Pointcheval, and Damien Vergnaud. New techniques for SPHFs and efficient one-round PAKE protocols. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 449–475. Springer, Heidelberg, August 2013.

[BBP04] Mihir Bellare, Alexandra Boldyreva, and Adriana Palacio. An uninstantiable random-oracle-model scheme for a hybrid-encryption problem. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 171–188. Springer, Heidelberg, May 2004.

[BC16] Olivier Blazy and Céline Chevalier. Structure-preserving smooth projective hashing. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 339–369. Springer, Heidelberg, December 2016.

[BCG+14] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy*, pages 459–474. IEEE Computer Society Press, May 2014.

[BCH+20] Christian Badertscher, Ran Canetti, Julia Hesse, Björn Tackmann, and Vassilis Zikas. Universal composition with global subroutines: Capturing global setup within plain UC. Cryptology ePrint Archive, Report 2020/1209, 2020. https://eprint.iacr.org/2020/1209.

[BCJ+19] Tatiana Bradley, Jan Camenisch, Stanislaw Jarecki, Anja Lehmann, Gregory Neven, and Jiayu Xu. Password-authenticated public-key encryption. In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 442–462. Springer, Heidelberg, June 2019.

[BCL+05] Boaz Barak, Ran Canetti, Yehuda Lindell, Rafael Pass, and Tal Rabin. Secure computation without authentication. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, Heidelberg, August 2005.

[Bea96] Donald Beaver. Adaptive zero knowledge and computational equivocation (extended abstract). In *28th ACM STOC*, pages 629–638. ACM Press, May 1996.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *20th ACM STOC*, pages 103–112. ACM Press, May 1988.

[BFSK11] Christina Brzuska, Marc Fischlin, Heike Schröder, and Stefan Katzenbeisser. Physically uncloneable functions in the universal composition framework. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 51–70. Springer, Heidelberg, August 2011.

[BMP00] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, Heidelberg, May 2000.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In Dorothy E. Denning, Raymond Pyle, Ravi Ganesan, Ravi S. Sandhu, and Victoria Ashby, editors, *ACM CCS 93*, pages 62–73. ACM Press, November 1993.

[BSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[Can00] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. https://eprint.iacr.org/2000/067.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Heidelberg, February 2007.

[CDVW12] Ran Canetti, Dana Dachman-Soled, Vinod Vaikuntanathan, and Hoeteck Wee. Efficient password authenticated key exchange via oblivious transfer. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 449–466. Springer, Heidelberg, May 2012.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[CGH98] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited (preliminary version). In *30th ACM STOC*, pages 209–218. ACM Press, May 1998.

[CGJ15] Ran Canetti, Vipul Goyal, and Abhishek Jain. Concurrent secure computation with optimal query complexity. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 43–62. Springer, Heidelberg, August 2015.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.

[CPs07] Ran Canetti, Rafael Pass, and abhi shelat. Cryptography from sunspots: How to use an imperfect reference string. In *48th FOCS*, pages 249–259. IEEE Computer Society Press, October 2007.

[DHK+22] Julien Duman, Dominik Hartmann, Eike Kiltz, Sabrina Kunzweiler, Jonas Lehmann, and Doreen Riepel. Group action key encapsulation and non-interactive key exchange in the QROM. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 36–66. Springer, Heidelberg, December 2022.

[DKS18] Luca De Feo, Jean Kieffer, and Benjamin Smith. Towards practical key exchange from ordinary isogeny graphs. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 365–394. Springer, Heidelberg, December 2018.

[DOP05] Yevgeniy Dodis, Roberto Oliveira, and Krzysztof Pietrzak. On the generic insecurity of the full domain hash. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 449–466. Springer, Heidelberg, August 2005.

[FHKP13] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. Non-interactive key exchange. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 254–271. Springer, Heidelberg, February / March 2013.

[GGJS11] Sanjam Garg, Vipul Goyal, Abhishek Jain, and Amit Sahai. Bringing people of different beliefs together to do UC. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 311–328. Springer, Heidelberg, March 2011.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Heidelberg, May 2013.

[GJO10] Vipul Goyal, Abhishek Jain, and Rafail Ostrovsky. Password-authenticated session-key generation on the internet in the plain model. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 277–294. Springer, Heidelberg, August 2010.

[GK03] Shafi Goldwasser and Yael Tauman Kalai. On the (in)security of the Fiat-Shamir paradigm. In *44th FOCS*, pages 102–115. IEEE Computer Society Press, October 2003.

[GK08] Vipul Goyal and Jonathan Katz. Universally composable multi-party computation with an unreliable common reference string. In Ran Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 142–154. Springer, Heidelberg, March 2008.

[GK10] Adam Groce and Jonathan Katz. A new framework for efficient password-based authenticated key exchange. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 516–525. ACM Press, October 2010.

[GL01] Oded Goldreich and Yehuda Lindell. Session-key generation using human passwords only. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 408–432. Springer, Heidelberg, August 2001.

[GL03] Rosario Gennaro and Yehuda Lindell. A framework for password-based authenticated key exchange. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, Heidelberg, May 2003. https://eprint.iacr.org/2003/032.ps.gz.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 323–341. Springer, Heidelberg, August 2007.

[Goy12] Vipul Goyal. Positive results for concurrently secure computation in the plain model. In *53rd FOCS*, pages 41–50. IEEE Computer Society Press, October 2012.

[Gro10] Jens Groth. Short pairing-based non-interactive zero-knowledge arguments. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 321–340. Springer, Heidelberg, December 2010.

[HL18] Björn Haase and Benoît Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, Report 2018/286, 2018. https://eprint.iacr.org/2018/286.

[HM04] Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 58–76. Springer, Heidelberg, February 2004.

[HR10] Feng Hao and Peter Ryan. J-pake: Authenticated key exchange without pki. *Transactions on Computational Science*, 11:192–206, 2010.

[HS14] Feng Hao and Siamak F. Shahandashti. The SPEKE protocol revisited. Cryptology ePrint Archive, Report 2014/585, 2014. https://eprint.iacr.org/2014/585.

[JG04] Shaoquan Jiang and Guang Gong. Password based key exchange with mutual authentication. In Helena Handschuh and Anwar Hasan, editors, *SAC 2004*, volume 3357 of *LNCS*, pages 267–279. Springer, Heidelberg, August 2004.

[JGH+20] Shaoquan Jiang, Guang Gong, Jingnan He, Khoa Nguyen, and Huaxiong Wang. PAKEs: New framework, new techniques and more efficient lattice-based constructions in the standard model. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 396–427. Springer, Heidelberg, May 2020.

[JR15] Charanjit S. Jutla and Arnab Roy. Dual-system simulation-soundness with applications to UC-PAKE and more. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 630–655. Springer, Heidelberg, November / December 2015.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 115–128. Springer, Heidelberg, May 2007.

[KKZZ14] Jonathan Katz, Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Distributing the setup in universally composable multi-party computation. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 20–29. ACM, July 2014.

[KOY01] Jonathan Katz, Rafail Ostrovsky, and Moti Yung. Efficient password-authenticated key exchange using human-memorable passwords. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 475–494. Springer, Heidelberg, May 2001.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In Mitsuru Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 636–652. Springer, Heidelberg, December 2009.

[KV11] Jonathan Katz and Vinod Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, Heidelberg, March 2011.

[Ler22] Antonin Leroux. A new isogeny representation and applications to cryptography. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part II*, volume 13792 of *LNCS*, pages 3–35. Springer, Heidelberg, December 2022.

[Mac02] Philip MacKenzie. The PAK suite: Protocols for Password-Authenticated Key Exchange. In *IEEE P1363.2*, 2002.

[MRH04] Ueli M. Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, impossibility results on reductions, and applications to the random oracle methodology. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 21–39. Springer, Heidelberg, February 2004.

[MRR20] Ian McQuoid, Mike Rosulek, and Lawrence Roy. Minimal symmetric PAKE and 1-out-of-N OT from programmable-once public functions. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 425–442. ACM Press, November 2020.

[NV04] Minh-Huyen Nguyen and Salil P. Vadhan. Simpler session-key generation from short random passwords. In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 428–445. Springer, Heidelberg, February 2004.

[PST17] Rafael Pass, Elaine Shi, and Florian Tramèr. Formal abstractions for attested execution secure processors. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 260–289. Springer, Heidelberg, April / May 2017.

[PW17] David Pointcheval and Guilin Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In Ramesh Karri, Ozgur Sinanoglu, Ahmad-Reza Sadeghi, and Xun Yi, editors, *ASIACCS 17*, pages 301–312. ACM Press, April 2017.

[Tha22] Justin Thaler. Proofs, arguments, and zero-knowledge. *Found. Trends Priv. Secur.*, 4(2-4):117–660, 2022. https://people.cs.georgetown.edu/jthaler/ProofsArgsAndZK.html.

[Yon14] Kazuki Yoneyama. Password-based authenticated key exchange without centralized trusted setup. In Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay, editors, *ACNS 14*, volume 8479 of *LNCS*, pages 19–36. Springer, Heidelberg, June 2014.

[Zca] The Zcash blockchain project. https://z.cash.

[ZY17] Jiang Zhang and Yu Yu. Two-round PAKE from approximate SPH and instantiations from lattices. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 37–67. Springer, Heidelberg, December 2017.

A Supplemental Preliminaries

A.1 The UC framework

The UC framework of Canetti [Can01, Can00] is based on the real/ideal simulation paradigm, i.e., the security is defined based on indistinghuishability between what an adversary can do in the real execution of the protocol and what it can do in an ideal world that is secure by definition.

The real world. An execution of a protocol π in the real world consists of $n \in \mathbb{N}$ interactive Turing machines (ITMs) $\mathcal{P}_1, \ldots, \mathcal{P}_n$ representing the parties, along with two additional ITMs, an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$, representing the external environment in which the protocol executes: $\mathcal{Z}$ gives inputs to the honest parties, receives their outputs, and can communicate with $\mathcal{A}$ at any point during the execution; $\mathcal{A}$ controls the corrupted parties and the delivery of messages between the parties. π completes once $\mathcal{Z}$ stops activating other parties and outputs a single bit. Let $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote $\mathcal{Z}$'s output in this execution.

The ideal world. A computation in the ideal world involves n dummy parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, a simulator $\mathcal{S}$, an environment $\mathcal{Z}$, and a functionality $\mathcal{F}$. The environment $\mathcal{Z}$ gives inputs to the honest (dummy) parties and receives their outputs; it also communicates with $\mathcal{S}$ at any point during the execution. As before, the computation completes once $\mathcal{Z}$ stops activating other parties and outputs a single bit. Let $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ denote $\mathcal{Z}$'s output in this execution.

Definition 1. *We say a protocol π UC-realizes a functionality $\mathcal{F}$, if for any PPT real world adversary $\mathcal{A}$, there exists a PPT ideal world adversary $\mathcal{S}$, such that for any PPT environment $\mathcal{Z}$, the following holds.*

$$\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}.$$

The $\mathcal{G}$-hybrid world. The execution of π proceeds as in the real world, however, the parties have access to a functionality $\mathcal{G}$. The communication of the parties with the functionality $\mathcal{G}$ is performed as in the ideal world. Let $\mathsf{REAL}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}}$ denote $\mathcal{Z}$'s output in this case.

Definition 2. *We say a protocol π UC-realizes a functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid world, if for any PPT real world adversary $\mathcal{A}$, there exists a PPT ideal world adversary $\mathcal{S}$, such that for any PPT environment $\mathcal{Z}$, the following holds.*

$$\mathsf{REAL}^{\mathcal{G}}_{\pi,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}.$$

If π is a protocol that UC-realizes the functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid world, and ρ is another protocol that UC-realizes the functionality $\mathcal{G}$. Then the composition theorem guarantees that π composed with ρ (i.e., replacing $\mathcal{G}$ by ρ) also UC-realizes the functionality $\mathcal{F}$.

Remark 2. *In this paper, we consider a malicious $\mathcal{A}$, i.e., it may instruct the corrupted parties to deviate from the protocol arbitrarily. We only consider static corruptions, which means the corrupted parties are fixed before the protocol starts, and is known to $\mathcal{Z}$, $\mathcal{S}$ and $\mathcal{F}$. In addition, $\mathcal{A}$ fully controls the communication between the parties, i.e., it can omit, change or inject messages.*

A.1.1 UC with global subroutines

Note that the UC theorem requires that both the functionality $\mathcal{F}$ and the protocol π are subroutine respecting. A protocol is subroutine respecting if, within any extended session of the protocol, the only machines that take input from or provide output to machines outside this session are the main machines of a subroutine. The first attempt [CDPW07] to handle scenarios with multiple sessions of protocols within the basic model of execution involved extending the UC framework, which increased complexity and caused incompatibility with the basic UC framework. In the subsequent work [BCH+20], it has been shown that the following formalism suffices for capturing universal composition with global subroutines within the basic UC framework.

Say, a protocol π *is said to UC-realize a functionality $\mathcal{F}$ in the presence of a global subroutine $\mathcal{G}$* if there exists an efficient simulator $\mathcal{S}$ such that no environment can distinguish whether it is interacting with π and $\mathcal{G}$, or with $\mathcal{F}$, $\mathcal{G}$, and $\mathcal{S}$. Here, $\mathcal{G}$ can be a single machine or an entire protocol instance, acting as a subroutine of π or $\mathcal{F}$, while also taking inputs directly from and providing outputs directly to the environment.

Therefore, without loss of generality, we consider our UC experiments in the hybrid world (as defined in Definition 2) for functionality $\mathcal{G}$, which can be either local or global.

A.1.2 The functionalities $\mathcal{F}_{\mathbf{CRS}}$, $\mathcal{G}_{\mathbf{CRS}}$ and $\mathcal{F}_{\mathbf{RO}}$

Here, we recall several functionalities for CRS and RO.

CRS and global CRS. The functionality $\mathcal{F}_{\mathrm{CRS}}$ as shown in Fig. 9 was defined by Cannetti and Fischlin in [CF01]. The functionality $\mathcal{G}_{\mathrm{CRS}}$ as shown in Fig. 10 defined by Canetti et al. [CDPW07] is a shared functionality, which differs from $\mathcal{F}_{\mathrm{CRS}}$. $\mathcal{F}_{\mathrm{CRS}}$ provides the reference string only to the parties that take part in the actual protocol execution. In particular, the environment does not have direct access to the reference string, while $\mathcal{G}_{\mathrm{CRS}}$ provides the same string to all parties and all protocol executions.

Functionality $\mathcal{F}_{\mathbf{CRS}}$

The functionality $\mathcal{F}_{\mathrm{CRS}}$ is parameterized by a distribution $\mathcal{D}_{\mathrm{CRS}}$. When activated by any party with input sid, proceeds in the following way:

- Find the recorded value r. If no value r has been previously recorded, then sample r in $\mathcal{D}_{\mathrm{CRS}}$ and record (sid, r) .
- Send (sid, r) to the activating party.

Figure 9: The functionality $\mathcal{F}_{\mathrm{CRS}}$

Random oracle. The functionality $\mathcal{F}_{\mathrm{RO}}$ as shown in Fig. 11 for a RO was defined by Hofheinz and Müller-Quade in [HM04], which captures an "idealisation" of a hash function. Given a query, it returns a random

Functionality $\mathcal{G}_{\text{CRS}}$

The functionality $\mathcal{G}_{\text{CRS}}$ is parameterized by a distribution $\mathcal{D}_{\text{CRS}}$. When activated by any party, proceeds in the following way:

- If no value r has been previously recorded, then sample r in $\mathcal{D}_{\text{CRS}}$ and records the value r.
- Sends the value r to the activating party.

Figure 10: The functionality $\mathcal{G}_{\text{CRS}}$

Functionality $\mathcal{F}_{\text{RO}}$

The functionality $\mathcal{F}_{\text{RO}}$ is parameterized by a domain D and a range R. It performs in the following way:

- Initializes a list $L_{\mathsf{sid}} = \emptyset$;
- Upon receiving (sid, x) (with $x \in D$) from any party:
  - If there is a pair (x, h^*) for some $h^* \in R$ in the L_{sid}, set $h = h^*$.
  - Otherwise, randomly choose $h \xleftarrow{\$} R$ and store (x, h) in L_{sid}.

  Once h is set, reply with (sid, h).

Figure 11: The functionality $\mathcal{F}_{\text{RO}}$

value. It also updates a local list L_{sid} in order to return the same value to similar queries. In particular, we consider ROs with respect to domain D and range R.

A.2 The BPR-security definition for PAKE

Here, we present the definition for PAKE in the BPR framework [BPR00]. The following description and definition is based on [ABP15], which in turn follows [BPR00, AFP05].

Let *Client* and *Server* denote the two sets of client and server users, respectively. Let *User* := *Client*∪*Server* denote the set of all users. Each client user $\mathtt{C} \in$ *Client* holds a password $\mathsf{pw}_{\mathtt{C}}$ that is uniformly chosen from a dictionary $\mathcal{PW}$, while each sever $\mathtt{S} \in$ *Server* holds a password $\mathsf{pw}_{\mathtt{S},\mathtt{C}}$ for each client user $\mathtt{C} \in$ *Client*. Assume each client password is chosen independently.

Execution of the protocol. Each user $\mathtt{U} \in$ *User* can execute the protocol multiple times with different partners, and each execution instance is called a session. We use $\Pi^i_{\mathtt{U}}$ to denote the i-th session of $\mathtt{U}$. Each session can be used only once associated with the following variables (that will be updated during the course of the experiment):

- $\mathsf{sid}^i_{\mathtt{U}}$, denotes the *session id* to keep track of different sessions. It is set as the (ordered) concatenation of all messages sent and received by $\Pi^i_{\mathtt{U}}$.
- $\mathsf{pid}^i_{\mathtt{U}}$, denotes the *partner id*. It is set as the identity of the user with whom $\Pi^i_{\mathtt{U}}$ believes it is interacting.
- $\mathsf{acc}^i_{\mathtt{U}}$ and $\mathsf{term}^i_{\mathtt{U}}$, denote whether $\Pi^i_{\mathtt{U}}$ has accepted or terminated.
- $\mathsf{SK}^i_{\mathtt{U}}$, denotes the *session key* of $\Pi^i_{\mathtt{U}}$.

For every distinct users $\mathtt{C} \in$ *Client* and $\mathtt{S} \in$ *Server*, the two sessions $\Pi^i_{\mathtt{C}}$ and $\Pi^j_{\mathtt{S}}$ are called *partnered* if: (1) $\mathsf{sid}^i_{\mathtt{C}} = \mathsf{sid}^j_{\mathtt{S}} \neq \mathtt{NULL}$; and (2) $\mathsf{pid}^i_{\mathtt{C}} = \mathtt{S}$, $\mathsf{pid}^j_{\mathtt{S}} = \mathtt{C}$.

Security. The security is defined via an experiment played between a challenger and an adversary $\mathcal{A}$. In particular, the challenger should initialize the system and allow $\mathcal{A}$ adaptively query the following *oracles*:

- $\mathrm{Execute}(\mathtt{C}, i, \mathtt{S}, j)$: For $\mathtt{C} \in$ *Client* and $\mathtt{S} \in$ *Server*, if both $\Pi^i_{\mathtt{C}}$ and $\Pi^j_{\mathtt{S}}$ have not been used, this oracle executes as the protocol specification to activate them on $\mathtt{C}$ and $\mathtt{S}$, respectively. The corresponding transcript is returned. This models passive attacks.
- $\mathrm{Send}(\mathtt{U}, i, \mathsf{in\_msg})$: For $\mathtt{U} \in$ *User*, this oracle sends a message $\mathsf{in\_msg}$ to the session $\Pi^i_{\mathtt{U}}$, which computes what the protocol says and returns a response message $\mathsf{out\_msg}$. This oracle models active attacks.
- $\mathrm{Corrupt}(\mathtt{C})$: For $\mathtt{C} \in$ *Client*, this oracle returns $\mathsf{pw}_{\mathtt{C}}$. This oracle models client corruptions.

- $\mathrm{Corrupt}(\mathtt{S}, \mathtt{C}, \mathsf{pw})$: For $\mathtt{S} \in$ *Server* and $\mathtt{C} \in$ *Client*, if $\mathsf{pw} = \perp$, this oracle returns the stored password $\mathsf{pw}_{\mathtt{S},\mathtt{C}}$, otherwise the stored password $\mathsf{pw}_{\mathtt{S},\mathtt{C}}$ is then changed to pw. This oracle models server corruptions.
- $\mathrm{Reveal}(\mathtt{U}, i)$: For $\mathtt{U} \in$ *User*, if $\Pi^i_{\mathtt{U}}$ has accepted and terminated with a session key $\mathsf{SK}^i_{\mathtt{U}}$ derived, this oracle returns it. This oracle models session key leakages.
- $\mathrm{Test}(\mathtt{U}, i)$: This oracle is allowed to be queried for only once and the tested session $\Pi^i_{\mathtt{U}}$ must be terminated and accepted. To complete this query, a random coin b is flipped: if $b = 1$, the real session key $\mathsf{SK}^i_{\mathtt{U}}$ is returned; otherwise, a random session key is returned.

The adversary $\mathcal{A}$ finally outputs a bit b'. Let Succ denote the event that $\mathcal{A}$ succeeds, i,e., the tested session $\Pi^i_{\mathtt{U}}$ is *fresh* and $b' = b$. For any $\mathtt{U} \in$ *User*, a session $\Pi^i_{\mathtt{U}}$ is called *fresh* unless one of the following is true at the conclusion of the experiment: (1) the session key has never been determined; or (2) $\mathcal{A}$ queried Reveal on $\Pi^i_{\mathtt{U}}$ or its partnered session; or (3) $\mathcal{A}$ corrupted password of $\mathtt{C}$ either via a query $\mathrm{Corrupt}(\mathtt{C})$ or via a query $\mathrm{Corrupt}(\cdot, \mathtt{C}, \cdot)$ before $\Pi^i_{\mathtt{U}}$ determining its session key, where $\mathtt{C} = \mathtt{U}$ or the partnered session of $\Pi^i_{\mathtt{U}}$ is a session of $\mathtt{C}$.

Definition 3. *We say a protocol π is BPR-secure if:*

- *(Correctness) For any partnered sessions $\Pi^i_{\mathtt{C}}$ and $\Pi^j_{\mathtt{S}}$, it holds that $\mathsf{acc}^i_{\mathtt{C}} = \mathsf{acc}^j_{\mathtt{S}} = \mathsf{true}$ and $\mathsf{SK}^i_{\mathtt{C}} = \mathsf{SK}^j_{\mathtt{S}}$, i.e., both sessions accept with the same session key;*
- *(Security) For any non-empty dictionary $\mathcal{PW}$ and PPT adversary $\mathcal{A}$ who makes at most n_{Send} online password-guessing attacks via the* Send *oracle, it holds that*

$$\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},\pi} := |\Pr[\mathsf{Succ}] - \frac{1}{2}| \leq n_{\mathrm{Send}}/|\mathcal{PW}| + \mathsf{negl}(\lambda).$$

A.3 BPR-security vs UC-security

In [CHK+05], Canetti et al. have shown that UC-security implies BPR-security. Here, we provide further clarification of the differences between the two definitional frameworks. Consider the case that the adversary corrupts one party and interacts with another honest party. Let x^* denote the adversary's input, and x denote the honest party's input.

- In the BPR framework, to complete the security analysis, we need to construct a challenger to bound the adversary's advantage. Typically, in the thought experiment, the best attack for the adversary is to launch online-guessing attack. Recall that, in the thought experiment in the BPR framework, the challenger is already aware of the honest party' input x. A direct extraction is not always a must; Often the challenger could play an **indirect extraction** strategy.

  More explicitly, consider a function f. (This f will be a hash function as we use in the next sections.) Instead of extracting the adversary's input x^*, now the challenger could be able to obtain/extract the image of x^*. If that is the case, i.e., the challenger obtains a value $y = f(x^*)$, then the challenger can test if the value y is equal to $f(x)$. This single-bit information will be sufficient for the challenger to decide the adversary's advantage. To the best of our knowledge, known instantiations of $f(\cdot)$ in existing PAKEs (including the GK-design [GK10] we use in the next sections) are identity functions. In our understanding, that is not necessary. As long as f is injective, i.e., $\forall\, x \neq x'$, $f(x) \neq f(x')$, equality test on $x \stackrel{?}{=} x^*$ is equivalent to $f(x) \stackrel{?}{=} f(x^*)$.
- Very differently, in the UC framework, to complete the security analysis, we need to construct a simulator who should be able to **directly extract** the adversary's input value x^*.

  Recall that, in the UC framework, the input x of the honest party is provided by the environment, and the simulator is *not* aware of the input x. (This is fundamentally different from the challenger in the BPR framework, where the challenger *is* aware of the honest party's input x.) More concretely, the simulator first extracts the adversary's input x^*. The simulator then invokes the PAKE functionality to learn whether the adversary makes a correct password-guess.

Consider the case that $x = x^*$ holds (i.e., the adversary makes a correct password-guess). The challenger in the BPR framework can halt the thought experiment and declare that the adversary wins. Because, in this case, the adversary can compute the real session key by itself and trivially win the experiment. However, the simulator in the UC framework still needs to continue simulating (the adversary's view).

A.4 Collision-resistant hash function family

Let $\hat{\mathcal{H}}$ be a family of hash functions from domain $\mathcal{X}$ to range $\mathcal{Y}$, i.e., $\hat{\mathcal{H}} := \{\mathcal{H} : \mathcal{X} \to \mathcal{Y}\}$. A *collision-resistant hash function family* is defined as follows: For any distinct $x_1, x_2 \in \mathcal{X}$, if $\mathcal{H}$ is chosen uniformly at random from $\hat{\mathcal{H}}$, then the probability of finding $\mathcal{H}(x_1) = H(x_2)$ in polynomial time is negligible. Formally, we define it as follows:

Definition 4. *We say $\hat{\mathcal{H}}$ is collision-resistant if for any PPT algorithm $\mathcal{A}$, there exists a negligible function* negl,

$$\Pr\left[\begin{array}{c} \mathcal{H} \xleftarrow{\$} \hat{\mathcal{H}} \\ (x_1, x_2) \xleftarrow{\$} \mathcal{A}(\mathcal{H}) \end{array} : x_1 \neq x_2 \text{ and } \mathcal{H}(x_1) = \mathcal{H}(x_2)\right] \leq \mathsf{negl}(\lambda).$$

A.5 Public key encryption

A Public Key Encryption (PKE) scheme consists of the following polynomial time algorithms:

- $\mathrm{KG}(1^\lambda)$: on input a security parameter 1^λ, this probabilistic algorithm outputs a pair of public-secret keys $(\mathsf{pk}, \mathsf{sk}) \in \mathcal{PK} \times \mathcal{SK}$.
- $\textsc{Enc}(\mathsf{pk}, \mathsf{m})$: on input public key $\mathsf{pk} \in \mathcal{PK}$, a message $\mathsf{m} \in \mathcal{M}$, this probabilistic algorithm outputs a ciphertext $\mathsf{CT} \in \mathcal{CT}$. In particular, it uses an internal randomness $\mathsf{r} \in \mathcal{R}$.
- $\textsc{Dec}(\mathsf{sk}, \mathsf{CT})$: on input secret key $\mathsf{sk} \in \mathcal{SK}$ and a ciphertext $\mathsf{CT} \in \mathcal{C}$, this deterministic algorithm outputs a message $\mathsf{m} \in \mathcal{M}$ or a special symbol $\perp$ indicating the ciphertext is invalid.

where $\mathcal{PK}$, $\mathcal{SK}$, $\mathcal{M}$, $\mathcal{R}$ and $\mathcal{C}$ are the sets of public keys, secret keys, messages, randomness and ciphertexts.

A labeled PKE scheme is defined in a similar way but adapted to support the inclusion of labels (as inputs) when encrypting and decrypting. We require normal correctness.

CPA-secure PKE. For any PPT algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, there exists a negligible function negl such that:

$$\Pr\left|\left[\begin{array}{l|l} |\mathsf{m}_0| = |\mathsf{m}_1|; b' = b. & \begin{array}{l} b \xleftarrow{\$} \{0,1\}; \\ (\mathsf{pk}, \mathsf{sk}) = \mathrm{KG}(1^\lambda); \\ (\mathsf{m}_0, \mathsf{m}_1) \xleftarrow{\$} \mathcal{A}_0(1^\lambda, \mathsf{pk}); \\ \mathsf{CT}^* = \textsc{Enc}(\mathsf{pk}, \mathsf{m}_b); \\ b' \xleftarrow{\$} \mathcal{A}_1(1^\lambda, \mathsf{pk}, \mathsf{CT}^*). \end{array} \end{array}\right] - 1/2\right| \leq \mathsf{negl}(\lambda).$$

CCA-secure labeled PKE. For any PPT algorithm $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, there exists a negligible function negl such that:

$$\Pr\left|\left[\begin{array}{l|l} |\mathsf{m}_0| = |\mathsf{m}_1|; b' = b. & \begin{array}{l} b \xleftarrow{\$} \{0,1\}; \\ \Pr[(\mathsf{pk}, \mathsf{sk}) = \mathrm{KG}(1^\lambda); \\ (\ell^*, \mathsf{m}_0, \mathsf{m}_1) \xleftarrow{\$} \mathcal{A}_0^{\textsc{Dec}(\mathsf{sk}, \cdot, \cdot)}(1^\lambda, \mathsf{pk}); \\ \mathsf{CT}^* = \textsc{Enc}(\mathsf{pk}, \ell^*, \mathsf{m}_b); \\ b' \xleftarrow{\$} \mathcal{A}_1^{\textsc{Dec}(\mathsf{sk}, \cdot, \cdot)}(1^\lambda, \mathsf{pk}). \end{array} \end{array}\right] - 1/2\right| \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}$'s queries are forbade on the input (ℓ^*, CT^*).

A.6 Smooth projective hash function

Now, we define Smooth Projective Hash Function (SPHF). The descriptions and definitions are based on [BBC+13]. For a clearer formulation, we give the definitions in a manner specific to PKE. Fix a PKE scheme $(\mathrm{KG}, \textsc{Enc}, \textsc{Dec})$ (cf. Appendix A.5). Let $\mathcal{CT}$ and $\mathcal{M}$ denote the ciphertext and message spaces with respect to a fixed public key pk, where $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathrm{KG}(1^\lambda)$. Define a set $\mathcal{X} := \{(\mathsf{CT} \in \mathcal{CT}, \mathsf{m} \in \mathcal{M})\}$. For $\mathsf{m} \in \mathcal{M}$, let $\mathcal{L}_\mathsf{m} := \{(\mathsf{CT}, \mathsf{m}) | \ \textsc{Dec}(\mathsf{sk}, \mathsf{CT}) = \mathsf{m}\} \subset \mathcal{X}$ and $\mathcal{L} := \bigcup_{\mathsf{m} \in \mathcal{M}} \mathcal{L}_\mathsf{m}$. For each $\mathsf{CT} \in \mathcal{CT}$, there is at most one $\mathsf{m} \in \mathcal{M}$ for which $(\mathsf{CT}, \mathsf{m}) \in \mathcal{L}$. An associated SPHF scheme consists of the following polynomial time algorithms:

- $\mathrm{PG}(1^\lambda)$: this probabilistic algorithm generates a public parameter Γ. Note that Γ determines a set $\mathcal{X}$ and a language $\mathcal{L}$ as defined above, and a set $\mathcal{V}$ of computed hash values. A public key pk is implicitly fixed by Γ.

- $\text{HKG}(\Gamma)$: this probabilistic algorithm generates a hashing key hk for Γ.
- $\text{PKG}(\Gamma, \mathsf{hk}, \mathsf{CT})$: this deterministic algorithm derives the projective key hp of hk, depending on CT.
- $\text{HASH}(\Gamma, \mathsf{hk}, (\mathsf{CT}, \mathsf{m}))$: this deterministic algorithm derives the hash value from the hashing key hk, for the tuple $(\mathsf{CT}, \mathsf{m}) \in \mathcal{X}$.
- $\text{PHASH}(\Gamma, \mathsf{hp}, (\mathsf{CT}, \mathsf{m}), w)$: this deterministic algorithm derives the hash value from the projection key hp, and the witness w for an element $(\mathsf{CT}, \mathsf{m}) \in \mathcal{L}$. In particular, w equals the randomness used when generating the ciphertext CT that encrypts m.

Correctness. For all $(\mathsf{CT}, \mathsf{m}) \in \mathcal{L}$ with a witness w of this fact, it holds that $\text{HASH}(\Gamma, \mathsf{hk}, (\mathsf{CT}, \mathsf{m})) = \text{PHASH}(\Gamma, \mathsf{hp}, (\mathsf{CT}, \mathsf{m}), w)$.

Smoothness for SPHF. For any $\Gamma \xleftarrow{\$} \text{PG}(1^\lambda)$ and $(\mathsf{CT}, \mathsf{m}) \notin \mathcal{L}$, the following two distributions are statistical close:

$$\left\{ (\mathsf{CT}, \mathsf{m}, \mathsf{hp}, \mathsf{k}) \;\middle|\; \begin{array}{l} \mathsf{hk} \xleftarrow{\$} \text{HKG}(\Gamma); \\ \mathsf{hp} = \text{PKG}(\Gamma, \mathsf{hk}, \mathsf{CT}); \\ \mathsf{k} = \text{HASH}(\Gamma, \mathsf{hk}, (\mathsf{CT}, \mathsf{m})). \end{array} \right\} \quad , \quad \left\{ (\mathsf{CT}, \mathsf{m}, \mathsf{hp}, \mathsf{k}) \;\middle|\; \begin{array}{l} \mathsf{hk} \xleftarrow{\$} \text{HKG}(\Gamma); \\ \mathsf{hp} = \text{PKG}(\Gamma, \mathsf{hk}, \mathsf{CT}); \\ \mathsf{k} \xleftarrow{\$} \mathcal{V}. \end{array} \right\}$$

B Supplemental Material for Homomorphic NIKE with Associated Functions

The proof of Theorem 1 is quite easy so we omit it here. Then we prove Theorem 2 and Theorem 3 in the next subsections, respectively.

B.1 Proof of Theorem 2

Recall: Given two hash functions $\mathcal{H}_0 : \mathcal{X} \to \mathcal{EK}$ and $\mathcal{H}_1 : \mathcal{X} \to \mathcal{EK}$, for any $\mathsf{ek}, \mathsf{pek} \in \mathcal{EK}$ and $x \in \mathcal{X}$, the associated functions (for $\sigma \in \{0, 1\}$) derived from PPK are defined as follows:

$$\begin{cases} f_\sigma(x, \mathsf{ek}) := \mathsf{ek} \cdot \mathcal{H}_\sigma(x); \\ \hat{f}_\sigma(x, \mathsf{pek}) := \mathsf{pek}/\mathcal{H}_\sigma(x). \end{cases}$$

It is obvious that for $\mathsf{pek} \leftarrow f_\sigma(x, \mathsf{ek})$, $\mathsf{ek} = \hat{f}_\sigma(x, \mathsf{pek})$ holds. In addition, for any $\mathsf{ek}' \in \mathcal{EK}$, $\hat{f}_\sigma(x, \mathsf{pek} \cdot \mathsf{ek}') = \hat{f}_\sigma(x, \mathsf{pek}) \cdot \mathsf{ek}'$ holds.

Perfect hiding. For any $x \in \mathcal{X}$ we have that

$$\begin{aligned} \Pr[\text{PEK} = \mathsf{pek} | \text{X} = \text{x}] &= \Pr[\mathsf{pek} = \text{EK} \cdot \mathcal{H}_\sigma(\text{x}) | \text{X} = \text{x}] \\ &= \Pr[\text{EK} = \mathsf{pek}/\mathcal{H}_\sigma(\text{x})] = \frac{1}{|\mathcal{X}|}, \end{aligned}$$

where PEK, EK and X denote three different variables. In particular, EK is assigned by invoking the probabilistic algorithm GEN. Due to this, we have that

$$\begin{aligned} \Pr[\text{X} = \text{x} | \text{PEK} = \mathsf{pek}] &= \frac{\Pr[\text{X} = \text{x} \wedge \text{PEK} = \mathsf{pek}]}{\Pr[\text{PEK} = \mathsf{pek}]} \\ &= \frac{\Pr[\text{PEK} = \mathsf{pek} | \text{X} = \text{x}] \cdot \Pr[\text{X} = \text{x}]}{\Sigma_{x \in \mathcal{X}} \Pr[\text{PEK} = \mathsf{pek} | \text{X} = \text{x}] \cdot \Pr[\text{X} = \text{x}]} \\ &= \frac{\frac{1}{q} \cdot \Pr[\text{X} = \text{x}]}{\Sigma_{x \in \mathcal{X}} \frac{1}{|\mathcal{X}|} \cdot \Pr[\text{X} = \text{x}]} \\ &= \frac{\Pr[\text{X} = \text{x}]}{\Sigma_{x \in \mathcal{X}} \Pr[\text{X} = \text{x}]} \\ &= \Pr[\text{X} = \text{x}]. \end{aligned}$$

Equivocability. When $\mathcal{H}_\sigma$ is modeled as RO. Let the algorithm EXT has the ability of programming RO such that it can compute the discrete-log of $\mathcal{H}_\sigma(x)$ easily, then compute the secret key of $\hat{f}_\sigma(x, \mathsf{pek})$ with pdk.

Type-I(i) Intractability. Assume there exists an algorithm $\mathcal{A}$ that breaks the Type-I(i) intractability, we can construct an algorithm $\mathcal{B}$ to break the one-wayness of the underlying homomorphic NIKE scheme as follows:

- Input a challenge tuple $(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$.
- Initialize two empty lists L_0 and L_1.
- For a fresh $\mathcal{H}_\sigma(x)$ query:
  - If $(x, \mathsf{ek}, \mathsf{dk}) \in L_\sigma$, return ek;
  - Otherwise, generate $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \text{GEN}(\mathsf{pp})$, return ek and append $(x, \mathsf{ek}, \mathsf{dk})$ into the list L_σ.
- Set $(\mathsf{pek}, \mathsf{pek}') = (\mathsf{ek}_0^*, \mathsf{ek}_1^*)$ and send $(\mathsf{pek}, \mathsf{pek}')$ as the challenge tuple to $\mathcal{A}$.
- After receiving the answer (x, k) from $\mathcal{A}$, if there exist $(x, \mathsf{ek}, \mathsf{dk}) \in L_0$ and $(x, \mathsf{ek}', \mathsf{dk}') \in L_1$. Output $\mathsf{k} \cdot \text{KEY}(\mathsf{dk}, \mathsf{ek}_1^*/\mathsf{ek}') \cdot \text{KEY}(\mathsf{dk}', \mathsf{ek}^*)$ as the solution; otherwise, output $\perp$.

If $\mathcal{A}$ has never query both $\mathcal{H}_0$ and $\mathcal{H}_1$ on x, then its answer (x, k) cannot be true. In the case that $\mathcal{A}$ has queried, if $\mathcal{A}$ outputs the correct answer, the following equation holds:

$$\mathsf{k} = \mathbf{Key}(\mathsf{pek}/\mathcal{H}_0(x), \mathsf{pek}'/\mathcal{H}_1(x)) = \mathbf{Key}(\mathsf{ek}_0^*/\mathsf{ek}, \mathsf{ek}_1^*/\mathsf{ek}').$$

We can adjust the equation to derive the value of $\mathbf{Key}(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$. Therefore, $\mathcal{B}$ computes the correct answer.

Type-I(ii) intractability. Here, we fix $\sigma = 0$. The analysis for $\sigma = 1$ is similar. Assume there exists an algorithm $\mathcal{A}$ that breaks the Type-I(ii) intractability, we can construct an algorithm $\mathcal{B}$ to break the one-wayness of the underlying homomorphic NIKE scheme. Given a challenge tuple $(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$, $\mathcal{B}$ works as follows:

- Initialize two empty lists L_0 and L_1.
- For a fresh $\mathcal{H}_0(x)$ query:
  - If $(x, \mathsf{ek}, \mathsf{dk}) \in L_0$, return ek;
  - Otherwise, generate $(\mathsf{ek}, \mathsf{dk}) \xleftarrow{\$} \text{GEN}(\mathsf{pp})$, return ek and append $(x, \mathsf{ek}, \mathsf{dk})$ into the list L_0.
- For a fresh $\mathcal{H}_1(x)$ query:
  - If $(x, \delta, \mathsf{ek}', \mathsf{dk}') \in L_1$, return ek';
  - Otherwise, choose a random bit δ, generate $(\mathsf{ek}', \mathsf{dk}') \xleftarrow{\$} \text{GEN}(\mathsf{pp})$, and append $(x, \delta, \mathsf{ek}', \mathsf{dk}')$ into the list L_1. If $\delta = 0$, return ek'; else if $\delta = 1$, return $\mathsf{ek}' \cdot \mathsf{ek}_1^*$.
- Set $\mathsf{pek} = \mathsf{ek}_0^*$ and send pek as the challenge to $\mathcal{A}$.
- After receiving the answer $(\mathsf{pek}', (x_0, \mathsf{k}_0), (x_1, \mathsf{k}_1))$ from $\mathcal{A}$, if there exist $(x_0, \mathsf{ek}_0, \mathsf{dk}_0) \in L_0$, $(x_1, \mathsf{ek}_1, \mathsf{dk}_1) \in L_0$, $(x_0, \delta_0, \mathsf{ek}_0', \mathsf{dk}_0') \in L_1$ and $(x_1, \delta_1, \mathsf{ek}_1', \mathsf{dk}_1') \in L_1$, do as follows:
  - Compute the following values:
    - $\bar{\mathsf{k}}_1 = \text{KEY}(\mathsf{dk}_0, \mathsf{ek}_0^* \cdot \mathsf{pek}')$;
    - $\bar{\mathsf{k}}_2 = \text{KEY}(\mathsf{dk}_1, \mathsf{ek}_0^* \cdot \mathsf{pek}')$;
    - $\bar{\mathsf{k}}_3 = \text{KEY}(\mathsf{dk}_0, \mathsf{ek}_0')$;
    - $\bar{\mathsf{k}}_4 = \text{KEY}(\mathsf{dk}_1, \mathsf{ek}_1')$;
    - $\bar{\mathsf{k}}_5 = \text{KEY}(\mathsf{dk}_0, \mathsf{ek}_1^*)$;
    - $\bar{\mathsf{k}}_6 = \text{KEY}(\mathsf{dk}_1, \mathsf{ek}_1^*)$.
  - If $\delta_1 = 1$ and $\delta_0 = 0$, output $\mathsf{k}^* = (\mathsf{k}_0 \cdot \bar{\mathsf{k}}_1 \cdot \bar{\mathsf{k}}_4 \cdot \bar{\mathsf{k}}_6)/(\mathsf{k}_1 \cdot \bar{\mathsf{k}}_2 \cdot \bar{\mathsf{k}}_3)$.
  - Else if $\delta_1 = 0$ and $\delta_0 = 1$, output $\mathsf{k}^* = (\mathsf{k}_1 \cdot \bar{\mathsf{k}}_2 \cdot \bar{\mathsf{k}}_3 \cdot \bar{\mathsf{k}}_5)/(\mathsf{k}_0 \cdot \bar{\mathsf{k}}_1 \cdot \bar{\mathsf{k}}_4)$.
  - Otherwise, output $\perp$.

If $\mathcal{A}$ has never query both $\mathcal{H}_0$ and $\mathcal{H}_1$ on x_0 and x_1, then its answer (x, k) cannot be true. In the case that $\mathcal{A}$ has queried, if $\mathcal{A}$ outputs the correct answer, then the following equations hold simultaneously:

$$\begin{cases} \mathsf{k}_0 = \mathbf{Key}(\mathsf{pek}/\mathcal{H}_0(x_0), \mathsf{pek}'/\mathcal{H}_1(x_0)) = \mathbf{Key}(\mathsf{ek}_0^*/\mathsf{ek}_0, \mathsf{pek}'/(\mathsf{ek}_0' \cdot (\mathsf{ek}_1^*)^{\delta_0})); \\ \mathsf{k}_1 = \mathbf{Key}(\mathsf{pek}/\mathcal{H}_0(x_1), \mathsf{pek}'/\mathcal{H}_1(x_1)) = \mathbf{Key}(\mathsf{ek}_0^*/\mathsf{ek}_1, \mathsf{pek}'/(\mathsf{ek}_1' \cdot (\mathsf{ek}_1^*)^{\delta_1})). \end{cases}$$

Note that $\delta_0 \neq \delta_1$ holds with probability $1/2$. In this case, we can compute $\mathsf{k}_0/\mathsf{k}_1$ and adjust the resulted equation to derive the value of $\mathbf{Key}(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$. Therefore, $\mathcal{B}$ computes the correct answer too.

Type-II(i) intractability. Here, we fix $\sigma = 0$. The analysis for $\sigma = 1$ is similar. Assume there exists an algorithm $\mathcal{A}$ that breaks the Type-II(i) intractability, we can construct an algorithm $\mathcal{B}$ to break the one-wayness of the underlying homomorphic NIKE scheme. Given a challenge tuple $(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$, $\mathcal{B}$ works as follows:

- Initialize an empty list L_0.
- For a fresh $\mathcal{H}_0(x)$ query:
  - If $(x, \mathsf{ek}', \mathsf{dk}') \in L_0$, return ek;
  - Otherwise, generate $(\mathsf{ek}', \mathsf{dk}') \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, return ek and append $(x, \mathsf{ek}', \mathsf{dk}')$ into the list L_0.
- Set $(\mathsf{ek}, \mathsf{pek}) = (\mathsf{ek}_0^*, \mathsf{ek}_1^*)$ and send $(\mathsf{ek}, \mathsf{pek})$ as the challenge tuple to $\mathcal{A}$.
- After receiving the answer (x, k) from $\mathcal{A}$, if there exist $(x, \mathsf{ek}', \mathsf{dk}') \in L_0$. Output $\mathsf{k} \cdot \textsc{Key}(\mathsf{dk}', \mathsf{ek}_0^*)$ as the solution; otherwise, output $\perp$.

If $\mathcal{A}$ has never query both $\mathcal{H}_0$ and $\mathcal{H}_1$ on x, then its answer (x, k) cannot be true. In the case that $\mathcal{A}$ has queried, if $\mathcal{A}$ outputs the correct answer, the following equation holds:

$$\mathsf{k} = \mathbf{Key}(\mathsf{ek}, \mathsf{pek}/\mathcal{H}_0(x)) = \mathbf{Key}(\mathsf{ek}_0^*, \mathsf{ek}_1^*/\mathsf{ek}').$$

We can adjust the equation to derive the value of $\mathbf{Key}(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$. Therefore, $\mathcal{B}$ computes the correct answer.

Type-II(ii) Intractability. Here, we fix $\sigma = 0$. The analysis for $\sigma = 1$ is similar. Assume there exists an algorithm $\mathcal{A}$ that breaks the Type-II(ii) intractability of the underlying homomorphic NIKE scheme, we can construct an algorithm $\mathcal{B}$ to break the one-wayness of the underlying homomorphic NIKE scheme. Given a challenge $(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$, $\mathcal{B}$ works as follows:

- Initialize two empty lists L_0.
- For a fresh $\mathcal{H}_0(x)$ query:
  - If $(x, \delta, \mathsf{ek}', \mathsf{dk}') \in L_1$, return ek';
  - Otherwise, choose a random bit δ, generate $(\mathsf{ek}', \mathsf{dk}') \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, and append $(x, \delta, \mathsf{ek}', \mathsf{dk}')$ into the list L_1. If $\delta = 0$, return ek'; else if $\delta = 1$, return $\mathsf{ek}' \cdot \mathsf{ek}_1^*$.
- Set $\mathsf{ek} = \mathsf{ek}_0^*$ and send ek as the challenge to $\mathcal{A}$.
- After receiving the answer $(\mathsf{pek}, (x_0, \mathsf{k}_0), (x_1, \mathsf{k}_1))$ from $\mathcal{A}$, if there exist $(x_0, \delta_0, \mathsf{ek}_0', \mathsf{dk}_0') \in L_0$ and $(x_1, \delta_1, \mathsf{ek}_1', \mathsf{dk}_1') \in L_1$, do as follows:
  - Compute $\bar{\mathsf{k}}_1 = \textsc{Key}(\mathsf{dk}_0', \mathsf{ek}_0^*)$ and $\bar{\mathsf{k}}_2 = \textsc{Key}(\mathsf{dk}_1', \mathsf{ek}_0^*)$;
  - If $\delta_1 = 1$ and $\delta_0 = 0$, output $\mathsf{k}^* = (\mathsf{k}_0 \cdot \bar{\mathsf{k}}_1)/(\mathsf{k}_1 \cdot \bar{\mathsf{k}}_2)$.
  - Else if $\delta_1 = 0$ and $\delta_0 = 1$, output $\mathsf{k}^* = (\mathsf{k}_1 \cdot \bar{\mathsf{k}}_2)/(\mathsf{k}_0 \cdot \bar{\mathsf{k}}_1)$.
  - Otherwise, output $\perp$.

If $\mathcal{A}$ has never query $\mathcal{H}_0$ on x_0 and x_1, then its answer (x, k) cannot be true. In the case that $\mathcal{A}$ has queried, if $\mathcal{A}$ outputs the correct answer, then the following equations hold simultaneously:

$$\begin{cases} \mathsf{k}_0 = \mathbf{Key}(\mathsf{ek}, \mathsf{pek}/\mathcal{H}_0(x_0)) = \mathbf{Key}(\mathsf{ek}_0^*, \mathsf{pek}/(\mathsf{ek}_0' \cdot (\mathsf{ek}_1^*)^{\delta_0})); \\ \mathsf{k}_1 = \mathbf{Key}(\mathsf{pek}/\mathcal{H}_0(x_1), \mathsf{ek}) = \mathbf{Key}(\mathsf{ek}_0^*, \mathsf{pek}/(\mathsf{ek}_1' \cdot (\mathsf{ek}_1^*)^{\delta_1})). \end{cases}$$

Note that $\delta_0 \neq \delta_1$ holds with probability $1/2$. In this case, we can compute $\mathsf{k}_0/\mathsf{k}_1$ and adjust the resulted equation to derive the value of $\mathbf{Key}(\mathsf{ek}_1^*, \mathsf{ek}_1^*)$. Therefore, $\mathcal{B}$ computes the correct answer too.

B.2 Proof of Theorem 3

Recall: Given two different generators $M, N \in \mathcal{EK}$ (from public parameters), for any $\mathsf{ek}, \mathsf{pek} \in \mathcal{EK}$ and $x \in \mathcal{X}$, the associated functions derived from SPAKE2 are defined as follows:

$$\begin{cases} f_0(x, \mathsf{ek}) := \mathsf{ek} \cdot M^x; & \hat{f}_0(x, \mathsf{pek}) := \mathsf{pek}/M^x; \\ f_1(x, \mathsf{ek}) := \mathsf{ek} \cdot N^x; & \hat{f}_1(x, \mathsf{pek}) := \mathsf{pek}/N^x. \end{cases}$$

For $\sigma \in \{0, 1\}$, it is obvious that for $\mathsf{pek} \leftarrow f_\sigma(x, \mathsf{ek})$, $\mathsf{ek} = \hat{f}_\sigma(x, \mathsf{pek})$ holds. In addition, for any $\mathsf{ek}' \in \mathcal{EK}$, $\hat{f}_\sigma(x, \mathsf{pek} \cdot \mathsf{ek}') = \hat{f}_\sigma(x, \mathsf{pek}) \cdot \mathsf{ek}'$ holds.

Perfect hiding. This can be proved by using the same proof idea in the previous subsection.

Equivocability. Since (M, N) are treated as CRS, this can be easily proved.

Type-I(i) intractability. Assume there exists an algorithm $\mathcal{A}$ that breaks the Type-I(i) intractability, we can construct an algorithm $\mathcal{B}$ to break the one-wayness of the homomorphic NIKE scheme. Given a challenge tuple $(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$, $\mathcal{B}$ works as follows:

- Generate $(M, \mathsf{dk}_M) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $(N, \mathsf{dk}_N) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, and open (M, N) as the public parameters.
- Set $(\mathsf{pek}, \mathsf{pek}') = (\mathsf{ek}_0^*, \mathsf{ek}_1^*)$ and send $(\mathsf{pek}, \mathsf{pek}')$ as the challenge tuple to $\mathcal{A}$.
- After receiving the answer (x, k) from $\mathcal{A}$, output $\mathsf{k} \cdot \textsc{Key}(M^x, \mathsf{dk}_M^x, \mathsf{pek}'/N^x) \cdot \textsc{Key}(N^x, \mathsf{dk}_N^x, \mathsf{ek}_0^*)$ as the solution.

If $\mathcal{A}$ outputs the correct answer, the following equation holds:

$$\mathsf{k} = \mathbf{Key}(\mathsf{pek}/M^x, \mathsf{pek}'/N^x) = \mathbf{Key}(\mathsf{ek}_0^*/M^x, \mathsf{ek}_1^*/N^x).$$

We can adjust the equation to derive the value of $\mathbf{Key}(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$. Therefore, $\mathcal{B}$ computes the correct answer too.

Type-I(ii) intractability. Here, we fix $\sigma = 0$. The analysis for $\sigma = 1$ is similar. Assume there exists an algorithm $\mathcal{A}$ that breaks Type-I(ii) intractability, we can construct an algorithm $\mathcal{B}$ to break the one-wayness of the underlying homomorphic NIKE scheme. Given a challenge tuple $(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$, $\mathcal{B}$ works as follows:

- Generate $(M, \mathsf{dk}_M) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $N = \mathsf{ek}_1^*$, and open (M, N) as the public parameters.
- Set $\mathsf{pek} = \mathsf{ek}_0^*$ and send pek as the challenge tuple to $\mathcal{A}$.
- After receiving the answer $(\mathsf{pek}', (x_0, \mathsf{k}_0), (x_1, \mathsf{k}_1))$ from $\mathcal{A}$, if $x_0 \neq x_1$:
  - Compute $\bar{\mathsf{k}}_1 = \textsc{Key}(M^{x_0}, \mathsf{dk}_M^{x_0}, \mathsf{pek}'/N^{x_0})$ and $\bar{\mathsf{k}}_2 = \textsc{Key}(M^{x_1}, \mathsf{dk}_M^{x_1}, \mathsf{pek}'/N^{x_1})$.
  - Output $\mathsf{k}^* = ((\mathsf{k}_0 \cdot \bar{\mathsf{k}}_1)/(\mathsf{k}_1, \bar{\mathsf{k}}_2))^{1/(x_1 - x_0)}$.

If $\mathcal{A}$ outputs the correct answer, then the following equations hold simultaneously:

$$\begin{cases} \mathsf{k}_0 = \mathbf{Key}(\mathsf{pek}/M^{x_0}, \mathsf{pek}'/N^{x_0}) = \mathbf{Key}(\mathsf{ek}_0^*/M^{x_0}, \mathsf{pek}'/(\mathsf{ek}_1^*)^{x_0}); \\ \mathsf{k}_1 = \mathbf{Key}(\mathsf{pek}/M^{x_1}, \mathsf{pek}'/N^{x_1}) = \mathbf{Key}(\mathsf{ek}_0^*/M^{x_1}, \mathsf{pek}'/(\mathsf{ek}_1^*)^{x_1}). \end{cases}$$

In the case that $x_0 \neq x_1$, we can compute $\mathsf{k}_0/\mathsf{k}_1$ and adjust the resulted equation to derive the value of $\mathbf{Key}(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$. Therefore, $\mathcal{B}$ computes the correct answer.

Type-II(i) intractability. Here, we fix $\sigma = 0$. The analysis for $\sigma = 1$ is similar. Assume there exists an algorithm $\mathcal{A}$ that breaks Type-II(i) intractability, we can construct an algorithm $\mathcal{B}$ to break the one-wayness of the underlying homomorphic NIKE scheme. Given a challenge tuple $(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$, $\mathcal{B}$ works as follows:

- Replace the generation of the public parameter M as $(M, \mathsf{dk}_M) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$.
- Set $(\mathsf{ek}, \mathsf{pek}) = (\mathsf{ek}_0^*, \mathsf{ek}_1^*)$ and send $(\mathsf{ek}, \mathsf{pek})$ as the challenge tuple to $\mathcal{A}$.
- After receiving the answer (x, k) from $\mathcal{A}$, output $\mathsf{k} \cdot \textsc{Key}(M^x, \mathsf{dk}_M^x, \mathsf{ek}_0^*)$ as the solution.

If $\mathcal{A}$ outputs the correct answer, the following equation holds:

$$\mathsf{k} = \mathbf{Key}(\mathsf{ek}, \mathsf{pek}/M^x) = \mathbf{Key}(\mathsf{ek}_0^*, \mathsf{ek}_1^*/M^x).$$

We can adjust the equation to derive the value of $\mathbf{Key}(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$. Therefore, $\mathcal{B}$ computes the correct answer.

Type-II(ii) intractability. Here, we fix $\sigma = 0$. The analysis for $\sigma = 1$ is similar. Assume there exists an algorithm $\mathcal{A}$ that breaks Type-II(ii) intractability, we can construct an algorithm $\mathcal{B}$ to break the one-wayness of the underlying homomorphic NIKE scheme. Given a challenge tuple $(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$, $\mathcal{B}$ works as follows:

- Replace the generation of the public parameter M as $M = \mathsf{ek}_0^*$.
- Set $\mathsf{ek} = \mathsf{ek}_1^*$ and send ek as the challenge to $\mathcal{A}$.
- After receiving the answer $(\mathsf{pek}, (x_0, \mathsf{k}_0), (x_1, \mathsf{k}_1))$ from $\mathcal{A}$, output

$$\mathsf{k}^* = (\mathsf{k}_0/\mathsf{k}_1)^{1/(x_1 - x_0)}.$$

If $\mathcal{A}$ outputs the correct answer, then the following equations hold simultaneously:

$$\begin{cases} \mathsf{k}_0 = \mathbf{Key}(\mathsf{pek}/M^{x_0}, \mathsf{ek}) = \mathbf{Key}(\mathsf{pek}/(\mathsf{ek}_0^*)^{x_0}, \mathsf{ek}_1^*); \\ \mathsf{k}_1 = \mathbf{Key}(\mathsf{pek}/M^{x_1}, \mathsf{ek}) = \mathbf{Key}(\mathsf{pek}/(\mathsf{ek}_0^*)^{x_1}, \mathsf{ek}_1^*). \end{cases}$$

In the case that $x_0 \neq x_1$, we can compute $\mathsf{k}_0/\mathsf{k}_1$ and adjust the resulted equation to derive the value of $\mathbf{Key}(\mathsf{ek}_0^*, \mathsf{ek}_1^*)$. $\mathcal{B}$ computes the correct answer too.

C Supplementary Proofs in the UC Framework

Here, we first prove that a global CRS setup is useless to UC-realize $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$. Then we present the detailed security proofs of our PAKE protocol in the UC framework. In particular, we will complete the UC-security analysis.

C.1 Proof of Theorem 4

Intuitively, a global CRS is also obtainable by the environment, which prevents the simulator from choosing the CRS on its own (in order to set up a trapdoor). Therefore, the simulator gains no more advantage. In order to deduce contradictions, we assume there exists a protocol π that UC-realizes $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$ in the $\mathcal{G}_{\text{CRS}}$-hybrid world.

First, consider an environment $\mathcal{Z}$ who plays the role of an honest party $\mathcal{P}_j$, and starts an execution instance of the protocol π with another honest part $\mathcal{P}_i$ and a dummy adversary $\mathcal{A}$ (who corrupts no parties and simply forwards messages between $\mathcal{Z}$ and the two parties as instructed). In addition, all parties (including $\mathcal{Z}$) access the shared functionality $\mathcal{G}_{\text{CRS}}$. The environment $\mathcal{Z}$ randomly chooses a password pw from $\{0,1\}^\ell$, and provides $\mathcal{P}_i$ with $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, \mathsf{server})$ as input. The environment $\mathcal{Z}$ also internally executes the code of $\mathcal{P}_j$ with $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}, \mathsf{client})$ as input, and let $\mathcal{A}$ to forward all messages between itself and $\mathcal{P}_i$. When the protocol terminates, $\mathcal{Z}$ compares the session-key SK_j that it obtained by playing $\mathcal{P}_j$ to the session-key SK_i that $\mathcal{P}_i$ outputs. If $\mathsf{SK}_i = \mathsf{SK}_j$ and $\mathsf{SK}_i \neq \perp$, $\mathcal{Z}$ outputs 1; otherwise, $\mathcal{Z}$ outputs 0. Remark that the protocol π runs in an *unauthenticated channel*, $\mathcal{P}_i$ cannot distinguish the messages sent by the real honest party $\mathcal{P}_j$ and the messages sent by $\mathcal{Z}$ in the name of $\mathcal{P}_j$. By the assumption that π is non-trivial, we have that, in a real execution, $\mathcal{Z}$ outputs 1 except with negligible probability.

Next, consider the ideal-world simulator $\mathcal{S}$ that is guaranteed to exist by the security of π. Simulator $\mathcal{S}$ interacts with the $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$ functionality, the $\mathcal{G}_{\text{CRS}}$ functionality and with $\mathcal{Z}$ (who follows the same strategy as above). Due to the security of π, it holds that except with negligible probability, $\mathsf{SK}_i = \mathsf{SK}_j$ and $\mathsf{SK}_i \neq \perp$ in this ideal execution. Therefore, in an ideal execution, $\mathcal{Z}$ outputs 1 except with negligible probability.

Then, consider a different environment $\mathcal{Z}'$ in the real world, who plays the role of the honest party $\mathcal{P}_i$ in a specific way, and starts an execution instance of the protocol π with the honest party $\mathcal{P}_j$ and a dummy adversary $\mathcal{A}'$. In particular, the environment $\mathcal{Z}'$ internally plays the simulator $\mathcal{S}$, a copy of the functionality $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$, and an ideal-mode honest party $\mathcal{P}_i$. In addition, $\mathcal{Z}'$ will run the algorithm for $\mathcal{S}$ using the same CRS as obtained from $\mathcal{G}_{\text{CRS}}$ to respond to all of $\mathcal{S}$'s $\mathcal{G}_{\text{CRS}}$ queries. Next, $\mathcal{Z}'$ hands $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}, \mathsf{server})$ to its internal honest party $\mathcal{P}_i$ and $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}, \mathsf{client}))$ to the external honest party $\mathcal{P}_j$ as inputs, respectively. Furthermore, $\mathcal{Z}'$ instructs $\mathcal{A}'$ to forward all messages between itself and $\mathcal{P}_j$. When the protocol terminates, $\mathcal{Z}'$ compares the session key SK_i that it obtained from the internal honest party $\mathcal{P}_i$ and the session-key SK_j that $\mathcal{P}_j$ outputs. If $\mathsf{SK}_i = \mathsf{SK}_j$ and $\mathsf{SK}_i \neq \perp$, $\mathcal{Z}$ outputs 1; otherwise,

$\mathcal{Z}$ outputs 0. The main observation is that the real execution of $\mathcal{Z}'$ with $\mathcal{A}'$ and the real honest $\mathcal{P}_j$ is identical to the ideal execution of $\mathcal{Z}$ above with $\mathcal{S}$ and $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$. Therefore, we have that, in this real execution, $\mathcal{Z}'$ outputs 1 except with negligible probability.

Last, consider $\mathcal{Z}'$ in the ideal world, it internally runs the simulator $\mathcal{S}$ above, and externally interacts with a new simulator $\mathcal{S}'$ to emulate $\mathcal{A}'$. Note that, all messages that $\mathcal{S}'$ received are simulated by $\mathcal{S}$. Neither $\mathcal{S}$ nor $\mathcal{S}'$ are given the input handed to the internal honest party $\mathcal{P}_i$ and the external honest party $\mathcal{P}_j$. Therefore, as long as neither simulator guesses a password to its copy of $\mathcal{F}_{\text{PAKE}}^{\mathcal{D}}$, their execution is independent of the password pw. To make $\mathcal{Z}'$ outputs 1, $\mathcal{S}$ and $\mathcal{S}'$ should both guess the correct password pw. Since pw is randomly chosen from $\mathcal{D}$, thus the correct password can be guessed with probability at most $1/|\mathcal{D}|$. Therefore, the probability $1 - 1/|\mathcal{D}|$ that $\mathcal{Z}'$ outputs 0 would not be negligible. This contradicts the assumed security of π and concludes the proof.

The above proof idea also applies to the case that $\mathcal{F}_{\text{le-PAKE}}^{\mathcal{D}}$, $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ or $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$ is considered.

C.2 Proof of Theorem 5 in Case-(1)

Now, we complete the proof of Theorem 5 in Case-(1). In particular, we investigate our protocol shown in Fig. 6. In particular, the protocol runs in the $\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}$-hybrid model. We then prove this protocol UC-realizes $\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}}$ in the $\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}$-hybrid world.

For a simulator $\mathcal{S}$ and an adversary $\mathcal{A}$, let $\mathsf{IDEAL}_{\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}},\mathcal{S},\mathcal{Z}}$ denote the output of the environment $\mathcal{Z}$ in the ideal world when interacting with $\mathcal{S}$, and let $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}}$ denote the output of $\mathcal{Z}$ in the real world (i.e., a $\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}$-hybrid world) when the protocol π in Fig. 6 is being run. We should prove that these distributions are computationally indistinguishable, i.e., $\mathsf{IDEAL}_{\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}},\mathcal{S},\mathcal{Z}} \stackrel{\text{c}}{\approx} \mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}}$.

In the next, we will consider a sequence of hybrid executions $\mathbf{Exec}_i$ for $0 \leq i \leq 9$ and let $\mathsf{EXEC}_{i,\mathcal{A},\mathcal{Z}}$ denote the view of $\mathcal{Z}$ in $\mathbf{Exec}_i$, where $\mathbf{Exec}_0$ corresponds to the real-world execution. We will show that $\mathsf{EXEC}_{i,\mathcal{A},\mathcal{Z}} \stackrel{\text{c}}{\approx} \mathsf{EXEC}_{i-1,\mathcal{A},\mathcal{Z}}$ for all i, and argue that the final view is identical to $\mathsf{IDEAL}_{\mathcal{F}_{\text{mPAKE}}^{\mathcal{D}},\mathcal{S},\mathcal{Z}}$.

Execution $\mathbf{Exec}_0$: Real
This is the real execution of Π where the environment $\mathcal{Z}$ runs the protocol (Fig. 6) with parties $\mathcal{P}_i$ and $\mathcal{P}_j$, both having access to $\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}$, and a dummy adversary $\mathcal{A}$.

Claim 1. $\mathsf{EXEC}_{0,\mathcal{A},\mathcal{Z}} = \mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\text{mPAKE}}^{\hat{\mathcal{D}}}}$.

Execution $\mathbf{Exec}_1$: Adding Ideal Layout
This is same as the real execution, but adding two dummy parties and a dummy functionality $\mathcal{F}$, and all previously existing nodes (except $\mathcal{Z}$) grouped into one machine called the simulator $\mathcal{S}$.

Claim 2. $\mathsf{EXEC}_{1,\mathcal{A},\mathcal{Z}} = \mathsf{EXEC}_{0,\mathcal{A},\mathcal{Z}}$

Proof. This is an internal modification, such that the claim holds. □

Execution $\mathbf{Exec}_2$: Adding Book-Keeping and NewKey Interface
Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to do all record-keeping and label all instances described in Fig. 4. $\mathcal{F}$ still forwards NewSession queries from the dummy parties in their enitemizey (including the password) to $\mathcal{S}$.
Modifications to $\mathcal{S}$: $\mathcal{S}$ creates NewKey queries for $\mathcal{F}$ from whatever output the simulated parties produce. $\mathcal{S}$ also locally labels for all simulated instances with an initialized label fresh.

Claim 3. $\mathsf{EXEC}_{2,\mathcal{A},\mathcal{Z}} = \mathsf{EXEC}_{1,\mathcal{A},\mathcal{Z}}$

Proof. These modifications are internal modifications, such that the claim holds. □

Execution $\mathbf{Exec}_3$: Allowing $\mathcal{F}$ to Set Session Keys for Two Honest Parties
Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions in Fig. 4 to set session keys when $\mathcal{P}_i$ and $\mathcal{P}_j$ are both honest and no matter $\mathsf{pw}_i = \mathsf{pw}_j$ or $\mathsf{pw}_i \neq \mathsf{pw}_j$.
Modifications to $\mathcal{S}$: When simulating two honest instances with the adversary $\mathcal{A}$ who just forwards all messages sent between the two parties:

- On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{client})$ from $\mathcal{F}$ to $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$:
  - Generate $(\mathsf{ek}_0, \mathsf{dk}_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, $\mathsf{pek}_0 = f_0(\overline{\mathsf{pw}}, \mathsf{ek}_0)$ and $(\mathsf{ek}_1, \mathsf{dk}_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$. Send $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_i$.
- On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}_j, \mathsf{server})$ from $\mathcal{F}$ to $\Pi^{\mathsf{sid}}_{\mathcal{P}_j}$:
  - If it previously received $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ from $\mathcal{A}$, continue.
  - Generate $(\mathsf{ek}'_0, \mathsf{dk}'_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, $(\mathsf{ek}'_1, \mathsf{dk}'_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $\mathsf{pek}'_1 = f_1(\overline{\mathsf{pw}}, \mathsf{ek}'_1)$. Send $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_j$.
  - Send $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{client})$ to $\mathcal{A}$, where the sub-session identifier $\mathsf{ssid} = \mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$ and store $\langle \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{client}\rangle$ marked fresh.
- On $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ from $\mathcal{A}$ to $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$:
  - If it previously sent $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ to $\mathcal{A}$, wait $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{ppw}_j, \mathsf{client})$. Until it comes, send $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{server})$ to $\mathcal{A}$ and store $\langle \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{server}\rangle$ marked fresh.
- On $(\mathsf{NewKey}, \mathsf{ssid}, \mathcal{P}_i, *)$ from $\mathcal{A}$ to $\mathcal{F}^{\hat{\mathcal{D}}}_{\mathrm{mPAKE}}$.
  - If there exists a record $\langle \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}\rangle$ not marked $\mathsf{completed}$, then do:
    - If the record is fresh and there exists a $\mathsf{completed}$ record $\langle \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{role}', \vartheta_j\rangle$ with $\mathsf{role} \neq \mathsf{role}'$ that was fresh when $\mathcal{P}_j$ output $(\mathsf{ssid}, \vartheta_j)$, then phrase ssid into $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$. If $\Pi^{\mathsf{sid}}_{\mathcal{P}_j}$ and $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$ use the same password, set $\vartheta_i = \vartheta_j$.
    - In other cases, set $\vartheta_i = \{0,1\}^\lambda$.

    Finally, append ϑ_i to record $\langle \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}\rangle$, mark it $\mathsf{completed}$, output $(\mathsf{ssid}, \vartheta_i)$ to $\mathcal{P}_i$.
  - Send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}$ with $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^\lambda$.

Claim 4. $\mathsf{EXEC}_{3,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{2,\mathcal{A},\mathcal{Z}}$.

Proof. The prefect hiding property of the underlying homomorphic NIKE with associated functions guarantes that the modifications for the way generating pek_0 and pek'_1 does not introduce any observable difference. □

Execution $\mathbf{Exec}_4$: Simulating An Honest Client Instance with Corrupt Partner
Modifications to $\mathcal{S}$: When simulating as honest client instance with corrupt partner, do as follows:

- On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{client})$ from $\mathcal{F}$:
  - Generate $(\mathsf{ek}_0, \mathsf{dk}_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, $\mathsf{pek}_0 = f_0(\overline{\mathsf{pw}}, \mathsf{ek}_0)$ and $(\mathsf{ek}_1, \mathsf{dk}_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$. Send $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_i$.
  - Wait $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$. Until it comes, send $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{server})$ to $\mathcal{A}$, where the sub-session identifier $\mathsf{ssid} = \mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$. In addition, store $\langle \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{server}\rangle$ marked fresh.
- On $(\mathsf{TestPwd}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{ppw}^*)$ from $\mathcal{A}$ to $\mathcal{F}^{\hat{\mathcal{D}}}_{\mathrm{mPAKE}}$:
  - Phrase ssid into $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$.
  - Phrase ppw^* into $\eta_1|\eta_2|\eta_3$, and try all possible passwords to find pw^* that makes Equations 1 hold simultaneously, otherwise set $\mathsf{pw}^* = \perp$.
    - If $\mathsf{pw}^* = \mathsf{pw}_j$, store a record $\langle \mathsf{sid}, \mathcal{P}_i, \theta_i\rangle$ with $\theta_i = \theta^*_i$.
    - Otherwise, store a record $\langle \mathsf{sid}, \mathcal{P}_i, \theta_i\rangle$ with $\theta_i \xleftarrow{\$} \{0,1\}^\lambda$.

- On $(\mathsf{NewKey}, \mathsf{ssid}, \mathcal{P}_i, \vartheta^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}$:
  - Send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}$, where $\mathsf{SK}^* = \theta_i \oplus \vartheta_i$ with θ_i retrieved from the record $\langle \mathsf{sid}, \mathcal{P}_i, * \rangle$.

Claim 5. $\mathsf{EXEC}_{4,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{3,\mathcal{A},\mathcal{Z}}$.

Proof. First, the modification of the way generating pek_0 does not introduce any observable difference. Second, since $(\mathcal{H}_1, \mathcal{H}_2, \mathcal{H}_3)$ are all collision resistant, and the invoked algorithms in Equations 1 are all deterministic, if any suitable password can be found, that should be unique except with negligible probability. The found password pw^* is identical to the corrupted party $\mathcal{P}_j$'s input. If both parties use different passwords, they also use different passwords to invoke the functionality $\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}$. In this case, the $\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}$ will send independent randomly chosen sub-session keys to both parties, which yields both parties output independent randomly chosen final session keys.

Summarizing all, this claim holds.

□

Execution $\mathbf{Exec}_5$: Simulating An Honest Server Instance with Corrupt Partner

Modifications to $\mathcal{S}$: When simulating an honest server instance with corrupt partner, do as follows:

- On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}_j, \mathsf{server})$ from $\mathcal{F}$:
  - Wait $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_0)$ from $\mathcal{A}$. Until it comes, generate $(\mathsf{ek}_0', \mathsf{dk}_0') \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, $(\mathsf{ek}_1', \mathsf{dk}_1') \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $\mathsf{pek}_1' = f_1(\overline{\mathsf{pw}}, \mathsf{ek}_1')$. Send $\mathsf{msg}' = (\mathsf{ek}_0', \mathsf{pek}_1')$ and $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{client})$ to $\mathcal{A}$, where the sub-session identifier $\mathsf{ssid} = \mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$. In addition, store $\langle \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{client} \rangle$ marked fresh.
- On $(\mathsf{TestPwd}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{ppw}^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}$:
  - Phrase ssid into $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$.
  - Phrase ppw^* into $\eta_1|\eta_2|\eta_3$, and try all possible passwords to find pw^* that makes Equations 2 hold simultaneously, otherwise set $\mathsf{pw}^* = \perp$.
    - If $\mathsf{pw}^* \neq \mathsf{pw}_i$, store a record $\langle \mathsf{sid}, \mathcal{P}_j, \theta_j \rangle$ with $\theta_j = \theta_j^*$.
    - Otherwise, store a record $\langle \mathsf{sid}, \mathcal{P}_j, \theta_j \rangle$ with $\theta_j \xleftarrow{\$} \{0,1\}^\lambda$.
- On $(\mathsf{NewKey}, \mathsf{ssid}, \mathcal{P}_j, \vartheta^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}$:
  - Send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}$, where $\mathsf{SK}^* = \theta_j \oplus \vartheta_j$ with θ_j retrieved from the record $\langle \mathsf{sid}, \mathcal{P}_j, * \rangle$.

Claim 6. $\mathsf{EXEC}_{5,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{4,\mathcal{A},\mathcal{Z}}$.

Proof. The proof idea is similar to the proof of Claim C.2.

□

Execution $\mathbf{Exec}_7$: Allowing $\mathcal{F}$ to Set Session Keys for An Honest Instance with Corrupt Partner

Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions Fig. 4 to set session keys when only one of $\mathcal{P}_i$ and $\mathcal{P}_j$ is honest.

Modifications to $\mathcal{S}$: When simulation an honest instance with corrupt partner: if an honest instance concludes with a session key SK^*, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ or $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}$.

Claim 7. $\mathsf{EXEC}_{7,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{6,\mathcal{A},\mathcal{Z}}$.

Proof. From $\mathbf{Exec}_5$ and $\mathbf{Exec}_6$, all variables needed to compute a session key that is same as the adversary $\mathcal{A}$ have been set for honest server and client instances labeled with $\mathsf{compromised}$, respectively. If $\Pi_{\mathcal{P}_j}^{\mathsf{sid}}$ in question is labeled with $\mathsf{interrupted}$, a random chosen session key is send to $\mathcal{F}$. All these are consistent with allowing $\mathcal{F}$ to set session keys for honest party with corrupt partner.

□

Execution $\mathbf{Exec}_8$: Adding TestPwd Interface And Removing Password Forwarding
Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions Fig. 4 to posses TestPwd queries. $\mathcal{F}$ still forwards NewSession queries from the dummy parties but not in their enitemizey to $\mathcal{S}$, as the password is removed.
Modifications to $\mathcal{S}$: When simulating an honest client (or server) instance, the local password equality tests about whether pw^* equals pw_i (or pw_j) are replaced by sending $\mathsf{TestPwd}(\mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ (or $\mathsf{TestPwd}(\mathsf{sid}, \mathcal{P}_j, \mathsf{pw}^*)$) to $\mathcal{F}$. Futhermore, the way to answer TestPwd query is modified as follows:

- On $(\mathsf{TestPwd}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{ppw}^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}$:
  - Phrase ssid into $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$.
  - Phrase ppw^* into $\eta_1|\eta_2|\eta_3$, and try all possible passwords to find pw^* that makes Equations 1 hold simultaneously, otherwise set $\mathsf{pw}^* = \perp$.
    - If $\mathsf{pw}^* \neq \perp$, store a record $\langle \mathsf{sid}, \mathcal{P}_i, \theta_i \rangle$ with $\theta_i = \theta_i^*$.
    - Otherwise, store a record $\langle \mathsf{sid}, \mathcal{P}_i, \theta_i \rangle$ with $\theta_i \xleftarrow{\$} \{0,1\}^\lambda$.
- On $(\mathsf{TestPwd}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{ppw}^*)$ from $\mathcal{A}$ to $\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}$:
  - Phrase ssid into $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$.
  - Phrase ppw^* into $\eta_1|\eta_2|\eta_3$, and try all possible passwords to find pw^* that makes Equations 2 hold simultaneously, otherwise set $\mathsf{pw}^* = \perp$.
    - If $\mathsf{pw}^* \neq \perp$, store a record $\langle \mathsf{sid}, \mathcal{P}_j, \theta_j \rangle$ with $\theta_j = \theta_j^*$.
    - Otherwise, store a record $\langle \mathsf{sid}, \mathcal{P}_j, \theta_j \rangle$ with $\theta_j \xleftarrow{\$} \{0,1\}^\lambda$.

Claim 8. $\mathsf{EXEC}_{8,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{7,\mathcal{A},\mathcal{Z}}$.

Proof. From $\mathbf{Exec}_6$, $\mathcal{S}$ is unnecessary to use pw_i or pw_j to simulate the messages expected to be sent to $\mathcal{A}$. Therefore, the internal modifications of invoking the TestPwd interface to replace local password equality tests do not introduce any observable difference. In the case that $\mathsf{pw}^* \neq \mathsf{pw}_j$ (resp., $\mathsf{pw}^* \neq \mathsf{pw}_i$), no matter the simulator sends what session key to $\mathcal{F}$, it will set $\Pi_{\mathcal{P}_i}^{\mathsf{sid}}$ (resp., $\Pi_{\mathcal{P}_j}^{\mathsf{sid}}$) in question with a random session key.
Summarize the above, this claim holds. □

Execution $\mathbf{Exec}_9$: Disallowing $\mathcal{S}$ Labels for Simulated Instances
Stop letting $\mathcal{S}$ to label for all simulated instances and let the job done internally by $\mathcal{F}$.

Claim 9. $\mathsf{EXEC}_{9,\mathcal{A},\mathcal{Z}} = \mathsf{EXEC}_{8,\mathcal{A},\mathcal{Z}}$

Proof. These modifications are only internal modifications. Note that the way of $\mathcal{S}$ labeling for all simulated instances are consistent with the way $\mathcal{F}$ do for dummy parties. Therefore, this modification do not introduce any observable difference such that the claim holds. □

In the final execution, $\mathcal{F}$ is equivalent to $\mathcal{F}_{\mathrm{mPAKE}}^{\mathcal{D}}$. The simulator $\mathcal{S}$ is finally constructed as shown in Fig. 12, which is no need to use the passwords and perfectly simulates the ideal world.
Combine all above discussions together, we conclude that

$$\mathsf{IDEAL}_{\mathcal{F}_{\mathrm{mPAKE}}^{\mathcal{D}},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}^{\mathcal{F}_{\mathrm{mPAKE}}^{\hat{\mathcal{D}}}}.$$

C.3 Proof of Theorem 5 in Case-(2)

Now, we complete the proof of Theorem 5 in Case-(2). In particular, we investigate our UC-secure protocol interpreted in Case-(2) as shown in Fig. 6, where the INHERITED PARAMETER is obtained by querying the functionality $\mathcal{G}_{\mathrm{CRS}}$ and all local hash computations are replaced by querying the functionality $\mathcal{F}_{\mathrm{RO}} = \{\mathcal{F}_{\mathrm{RO}i}\}_{i\in[3]}$. In particular, *(i)* $\mathcal{H}_1$ (resp., $\mathcal{H}_2$) computations are replaced by querying $\mathcal{F}_{\mathrm{RO}_1}$ (resp., $\mathcal{F}_{\mathrm{RO}_2}$) parameterized with domain $\{0,1\}^*$ and range $\{0,1\}^\lambda$; *(ii)* $\mathcal{H}_3$ computations are replaced by querying $\mathcal{F}_{\mathrm{RO}_3}$

Simulating an honest client instance $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{client})$ from $\mathcal{F}^{\mathcal{D}}_{\mathrm{mPAKE}}$:

Generate $(\mathsf{ek}_0, \mathsf{dk}_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, $\mathsf{pek}_0 = f_0(\overline{\mathsf{pw}}, \mathsf{ek}_0)$ and $(\mathsf{ek}_1, \mathsf{dk}_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$. Send $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_i$. Store $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_i} := \langle \mathsf{client}, \mathcal{P}_j, \mathsf{msg}, \mathsf{pdk}_0, \mathsf{dk}_1 \rangle$.

On $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ from $\mathcal{A}$:

If $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_i} \neq \emptyset$, then update $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_i} := \langle \mathsf{client}, \mathcal{P}_j, \mathsf{msg}, \mathsf{pdk}_0, \mathsf{dk}_1, \mathsf{msg}' \rangle$, send $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{server})$ to $\mathcal{A}$ and store $\langle \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{server} \rangle$, where $\mathsf{ssid} := \mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$.

On $(\mathsf{TestPwd}, \mathsf{ssid}, \mathcal{P}_i, \mathsf{ppw}^*)$ from $\mathcal{A}$:

1. If there is a record $\langle \mathsf{ssid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{server} \rangle$, then phrase ssid into $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$. Futhermore, if $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_i} = \langle \mathsf{client}, \mathcal{P}_j, \mathsf{msg}, *, *, \mathsf{msg}' \rangle$:
   - Retrieve dk_1 from $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_i}$ and phrase ppw^* into $\eta_1|\eta_2|\eta_3$.
   - Try all possible passwords in $\mathcal{D}$ to determine pw^* that makes Eequations 1 hold simultaneously, otherwise set $\mathsf{pw}^* = \perp$.
   - Send $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\mathrm{mPAKE}}$.
   - If $\mathsf{pw}^* \neq \perp$, set $\theta = \theta^*_i$; otherwise, $\theta \xleftarrow{\$} \{0,1\}^\lambda$.
   - Update $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_i} = \langle \mathsf{client}, \mathcal{P}_j, \mathsf{msg}, \mathsf{pdk}_0, \mathsf{dk}_1, \mathsf{msg}', \theta \rangle$
2. Otherwise, ignore this query.

On $(\mathsf{NewKey}, \mathsf{ssid}, \mathcal{P}_i, \vartheta^*)$ from $\mathcal{A}$:

Set $\mathsf{SK}^* = \theta \oplus \vartheta^*$ and send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\mathrm{mPAKE}}$.

Simulating an honest server instance $\Pi^{\mathsf{sid}}_{\mathcal{P}_j}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{server})$ from $\mathcal{F}^{\mathcal{D}}_{\mathrm{mPAKE}}$:

1. Wait $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ from $\mathcal{A}$. When it comes, generate $(\mathsf{ek}'_0, \mathsf{dk}'_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, $(\mathsf{ek}'_1, \mathsf{dk}'_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $\mathsf{pek}'_1 = f_1(\overline{\mathsf{pw}}, \mathsf{ek}'_1)$, send $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_j$.
2. Store $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_j} := \langle \mathsf{server}, \mathcal{P}_i, \mathsf{msg}, \mathsf{msg}', \mathsf{dk}'_0, \mathsf{pdk}'_1 \rangle$.
3. Send $(\mathsf{NewSession}, \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{client})$ to $\mathcal{A}$ and store $\langle \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{client} \rangle$, where $\mathsf{ssid} := \mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$

On $(\mathsf{TestPwd}, \mathsf{ssid}, \mathcal{P}_j, \mathsf{ppw}^*)$ from $\mathcal{A}$:

1. If there is a record $\langle \mathsf{ssid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{client} \rangle$, then phrase ssid into $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'$. Futhermore, if $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_j} = \langle \mathsf{server}, \mathcal{P}_i, \mathsf{msg}, \mathsf{msg}', *, * \rangle$:
   - Retrieve dk'_0 from $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_j}$ and phrase ppw^* into $\eta_1|\eta_2|\eta_3$.
   - Try all possible passwords in $\mathcal{D}$ to determine pw^* that makes the Equations 2 hold simultaneously, otherwise set $\mathsf{pw}^* = \perp$.
   - Send $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\mathrm{mPAKE}}$.
   - If $\mathsf{pw}^* \neq \perp$, set $\theta = \theta^*_j$; otherwise, $\theta \xleftarrow{\$} \{0,1\}^\lambda$.
   - Update $\mathsf{state}^{\mathsf{sid}}_{\mathcal{P}_j} = \langle \mathsf{server}, \mathcal{P}_i, \mathsf{msg}, \mathsf{msg}', \mathsf{dk}'_0, \mathsf{pdk}'_1, \theta \rangle$.
2. Otherwise, ignore this query.

On $(\mathsf{NewKey}, \mathsf{ssid}, \mathcal{P}_j, \vartheta^*)$ from $\mathcal{A}$:

Set $\mathsf{SK}^* = \theta \oplus \vartheta^*$ and send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\mathrm{mPAKE}}$.

Figure 12: Simulation of honest instances when the CRS setup is functional and the RO setup falls back to CRHF

parameterized with domain $\{0,1\}^*$ and range $\{0,1\}^{3\lambda}$. Abusing notations, we use $\mathcal{H}_i(x)$ to denote the answer of querying $\mathcal{F}_{\text{RO}i}$ on x. We then prove this protocol UC-realizes $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$ in the $\{\mathcal{G}_{\text{CRS}}, \mathcal{F}_{\text{RO}}\}$-hybrid world.

For $\mathcal{S}$ and $\mathcal{A}$, let $\mathsf{IDEAL}^{\mathcal{G}_{\text{CRS}}}_{\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}},\mathcal{S},\mathcal{Z}}$ denote the output of the environment $\mathcal{Z}$ in the ideal world when interacting with $\mathcal{S}$ and $\mathcal{G}_{\text{CRS}}$, and let $\mathsf{REAL}^{\mathcal{G}_{\text{CRS}},\mathcal{F}_{\text{RO}}}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the output of $\mathcal{Z}$ in the real world (i.e., a $\{\mathcal{G}_{\text{CRS}}, \mathcal{F}_{\text{RO}}\}$-hybrid world) when the protocol π in Fig. 6 is being run. We should prove that these distributions are computationally indistinguishable, i.e., $\mathsf{IDEAL}^{\mathcal{G}_{\text{CRS}}}_{\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}},\mathcal{S},\mathcal{Z}} \overset{\text{c}}{\approx} \mathsf{REAL}^{\mathcal{G}_{\text{CRS}},\mathcal{F}_{\text{RO}}}_{\pi,\mathcal{A},\mathcal{Z}}$. In the next, we will also consider a sequence of hybrid executions $\mathbf{Exec}_i$ for $0 \leq i \leq 9$ and let $\mathsf{EXEC}_{i,\mathcal{A},\mathcal{Z}}$ denote the view of $\mathcal{Z}$ in $\mathbf{Exec}_i$, where $\mathbf{Exec}_0$ corresponds to the real-world execution. We will show that $\mathsf{EXEC}_{i,\mathcal{A},\mathcal{Z}} \overset{\text{c}}{\approx} \mathsf{EXEC}_{i-1,\mathcal{A},\mathcal{Z}}$ for all i, and argue that the final view is identical to $\mathsf{IDEAL}^{\mathcal{G}_{\text{CRS}}}_{\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}},\mathcal{S},\mathcal{Z}}$.

Execution $\mathbf{Exec}_0$: Real
This is the real execution of Π where the environment $\mathcal{Z}$ runs the protocol (Fig. 6) with parties $\mathcal{P}_i$ and $\mathcal{P}_j$, both having access to $\mathcal{G}_{\text{CRS}}$ and $\mathcal{F}_{\text{RO}}$, and a dummy adversary $\mathcal{A}$.

Claim 10. $\mathsf{EXEC}_{0,\mathcal{A},\mathcal{Z}} = \mathsf{REAL}^{\mathcal{G}_{\text{CRS}},\mathcal{F}_{\text{RO}}}_{\pi,\mathcal{A},\mathcal{Z}}$.

Execution $\mathbf{Exec}_1$: Adding Ideal Layout
This is same as the real execution, but adding two dummy parties and a dummy functionality $\mathcal{F}$, and all previously existing nodes (except $\mathcal{Z}$) are grouped into one machine called the simulator $\mathcal{S}$.

Claim 11. $\mathsf{EXEC}_{1,\mathcal{A},\mathcal{Z}} = \mathsf{EXEC}_{0,\mathcal{A},\mathcal{Z}}$.

Proof. This is an internal modification, such that the claim holds. □

Execution $\mathbf{Exec}_2$: Adding Book-Keeping, NewKey and RegisterTest Interfaces
Modifications to $\mathcal{F}$:

We now allow $\mathcal{F}$ to do all record-keeping and label all instances described in Fig. 4. $\mathcal{F}$ still forwards NewSession queries from the dummy parties in their enitemizey (including the password) to $\mathcal{S}$.
Modifications to $\mathcal{S}$:

$\mathcal{S}$ creates NewKey queries for $\mathcal{F}$ from whatever output the simulated parties parties produce.

Claim 12. $\mathsf{EXEC}_{2,\mathcal{A},\mathcal{Z}} = \mathsf{EXEC}_{1,\mathcal{A},\mathcal{Z}}$

Proof. These modifications are internal modifications, such that the claim holds. □

Execution $\mathbf{Exec}_3$: Allowing $\mathcal{S}$ to Simulate $\mathcal{F}_{\text{RO}}$
Modifications to $\mathcal{S}$: Initialize three empty lists, $L_{\mathcal{H}_1}$ and $L_{\mathcal{H}_2}$ and $L_{\mathcal{H}_3}$, and simulate $\mathcal{F}_{\text{RO}}$ queries as follows:

- On $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw})$ to $\mathcal{F}_{\text{RO}_1}$:
  - If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega) \in L_{\mathcal{H}_1}$, return ω.
  - Otherwise, return $\omega \xleftarrow{\$} \{0,1\}^\lambda$ and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega)$ to the list $L_{\mathcal{H}_1}$.
- On $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw})$ to $\mathcal{F}_{\text{RO}_2}$:
  - If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}, \hat{\omega}) \in L_{\mathcal{H}_2}$, return $\hat{\omega}$.
  - Otherwise, return $\hat{\omega} \xleftarrow{\$} \{0,1\}^\lambda$ and append $(\mathsf{sid}|i|j|\mathsf{ek}_1|\mathsf{pek}'_1|\mathsf{k}_1|\mathsf{pw}, \hat{\omega})$ to the list $L_{\mathcal{H}_2}$.
- On $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega|\hat{\omega}|\mathsf{pw})$ to $\mathcal{F}_{\text{RO}_3}$:
  - If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega|\hat{\omega}|\mathsf{pw}, \mu|\hat{\mu}|\theta) \in L_{\mathcal{H}_3}$, return $\mu|\hat{\mu}|\theta$.
  - Otherwise, return $\mu|\hat{\mu}|\theta \xleftarrow{\$} \{0,1\}^{3\lambda}$ and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega|\hat{\omega}|\mathsf{pw}, \mu|\hat{\mu}|\theta)$ to the list $L_{\mathcal{H}_3}$.

Claim 13. $\mathsf{EXEC}_{3,\mathcal{A},\mathcal{Z}} = \mathsf{EXEC}_{2,\mathcal{A},\mathcal{Z}}$.

Proof. These modifications are internal modifications, such that the claim holds. □

Execution $\mathbf{Exec}_4$: Allowing $\mathcal{F}$ to Set Session Keys for Two Honest Parties

Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions in Fig. 4 to make a client instance get ready and set session keys when $\mathcal{P}_i$ and $\mathcal{P}_j$ are both honest and no matter $\mathsf{pw}_i = \mathsf{pw}_j$ or $\mathsf{pw}_i \neq \mathsf{pw}_j$.

Modifications to $\mathcal{S}$: When simulating two honest instances with the adversary $\mathcal{A}$ just forwarding all messages sent between them:

- When $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$ receiving $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{client})$ from $\mathcal{F}$:
  - Generate $(\mathsf{pek}_0, \mathsf{pdk}_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $(\mathsf{ek}_1, \mathsf{dk}_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$.
  - Send $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_i$.
  - Wait $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ from $\mathcal{A}$. When it comes, execute the sub-protocol $\mathrm{Prot}_{\textsc{pake}}$ using $\mathsf{ppw}_i = \mathsf{ppw}_j$ as input if both parties use the same password, or a random value chosen from $\{0,1\}^{3\lambda}$ in another case.
  - Until $\mathrm{Prot}_{\textsc{pake}}$ completes with an output, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}$, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^\lambda$.
- When $\Pi^{\mathsf{sid}}_{\mathcal{P}_j}$ receiving $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}_j, \mathsf{server})$ from $\mathcal{F}$:
  1. Wait $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ from $\mathcal{A}$. When it comes: generate $(\mathsf{ek}'_0, \mathsf{dk}'_0) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $(\mathsf{pek}'_1, \mathsf{pdk}'_1) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$; send $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_j$; execute the sub-protocol $\mathrm{Prot}_{\textsc{pake}}$ using a random value chosen from $\{0,1\}^{3\lambda}$ as $\mathcal{P}_j$'s input ppw_j.
  2. Until $\mathrm{Prot}_{\textsc{pake}}$ completes with an output, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}$, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^\lambda$.

Claim 14. $\mathsf{EXEC}_{4,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{3,\mathcal{A},\mathcal{Z}}$.

Proof. Due to the perfect hiding property of the underlying homomorphic NIKE scheme, the modifications on the way generating pek_0 and pek'_1 does not introduce any difference.

Note that $\mathcal{S}$ simulates the two honest instances without making any $\mathcal{F}_{\textsc{ro}}$ queries and any bachpatch method to make $\mathcal{A}$'s subsequent queries be consistent with these values. The adversary $\mathcal{A}$ would notice the difference introduced only when it queried $\mathcal{F}_{\textsc{ro}}$ on correct values, including:

- A $\mathcal{F}_{\textsc{ro}_1}$ query on $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathbf{Key}(\hat{f}_0(pw, \mathsf{pek}_0), \mathsf{ek}'_0)|\mathsf{pw}$ that returns ω;
- A $\mathcal{F}_{\textsc{ro}_2}$ query on $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathbf{Key}(\mathsf{ek}_1, \hat{f}_1(\mathsf{pw}, \mathsf{pek}'_1))|\mathsf{pw}$ that returns $\hat{\omega}$;
- A $\mathcal{F}_{\textsc{ro}_3}$ query on $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega|\hat{\omega}|\mathsf{pw}$ that returns $\mu|\hat{\mu}|\theta$.

In particular, $\mathsf{pek}_0, \mathsf{ek}'_0, \mathsf{ek}_1$ and pek'_1 are simulated messages by two honest instances that used the same password pw. Without loss of generality, we assume the query order is $\mathcal{F}_{\textsc{ro}_1}/\mathcal{F}_{\textsc{ro}_2}$-$\mathcal{F}_{\textsc{ro}_3}$. No matter in which case, the adversary must query $\mathcal{F}_{\textsc{ro}_1}$ on the correct value. If $\mathcal{A}$ makes this event occur, we may construct a PPT algorithm $\mathcal{B}$ to break the Type-II(i) intractability of the underlying homomorphic NIKE scheme. In particular, given a challenge tuple $(\mathsf{pek}^*, \mathsf{ek}^*)$. $\mathcal{B}$ simulate $\mathbf{Exec}_4$ but with modifications as follows:

- When receiving $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{client})$ from $\mathcal{F}$, set $\mathsf{pek}_0 = \mathsf{pek}^* \cdot \mathsf{ek}^{\mathsf{sid}}_i$ with $(\mathsf{ek}^{\mathsf{sid}}_i, \mathsf{dk}^{\mathsf{sid}}_i) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$.
- When receiving $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{server})$ from $\mathcal{F}$, set $\mathsf{ek}'_0 = \mathsf{ek}^* \cdot \mathsf{ek}^{\mathsf{sid}}_j$ with $(\mathsf{ek}^{\mathsf{sid}}_j, \mathsf{dk}^{\mathsf{sid}}_j) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$.
- When $\mathcal{A}$ stops, for every $\mathcal{F}_{\textsc{ro}_2}$ query on $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}$, where pek_0 and ek'_0 were actually simulated as before, do as follows:
  - Compute $\bar{k}_1 = \textsc{Key}(\mathsf{dk}^{\mathsf{sid}}_i, \mathsf{ek}^* \cdot \mathsf{ek}^{\mathsf{sid}}_j)$ and $\bar{k}_2 = \textsc{Key}(\mathsf{dk}^{\mathsf{sid}}_j, \hat{f}_0(\mathsf{pw}, \mathsf{pek}^*))$.
  - Compute $\mathsf{k}^* = \mathsf{k}_0/(\bar{k}_1 \cdot \bar{k}_2)$.
  - Add $(\mathsf{pw}, \mathsf{k}^*)$ into the list of possible solutions.

Note that there exists only one correct value such that

$$\mathsf{k}_0 = \mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}^*) \cdot \mathsf{ek}^{\mathsf{sid}}_i, \mathsf{ek}^* \cdot \mathsf{ek}^{\mathsf{sid}}_j).$$

thus we can adjust the equation so that only $\mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}^*), \mathsf{ek}^*)$ remains on the right. If the above-mentioned event occurs, the correct answer was added in the list. Thus, it only occurs with negligible probability.

To summarize the above, this claim holds. □

Execution $\mathbf{Exec}_6$: Simulating An Honest Server Instance with Corrupt Partner
Modifications to $\mathcal{S}$: When simulating an honest server instance with a corrupt partner, do as follows:

- Initialize three new lists: L_{server} to record $\mathcal{S}$'s simulated information specific to honest server instances; $L_{\mathsf{TestPwd,s}}$ to record the adversary's password-guess against server instances.
- On $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw})$ to $\mathcal{F}_{\mathrm{RO}_1}$, where $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ and $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$:
  - If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0) \in L_{\mathcal{H}_1}$, return ω_0.
  - If $(\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{dk}'_0, \mathsf{pdk}'_1, \eta_1^*|\eta_2^*|\eta_3^*, \vartheta^*) \in L_{\mathsf{server}} \wedge \mathsf{k}_0$ $\triangle$-*satisfies* $(\mathsf{pw}, \mathsf{dk}'_0, \mathsf{pek}_0, \hat{f}_0)$:
    - If $\vartheta^* = \perp$, then append $(\mathsf{sid}, j, \mathsf{pw})$ to $L_{\mathsf{TestPwd,s}}$.
    - Else if $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}, \omega_1) \in L_{\mathcal{H}_2} \wedge \mathsf{k}_1 \triangledown$-*satisfies* $(\mathsf{pw}, \mathsf{pdk}'_1, \mathsf{ek}_1, \hat{f}_1)$, then return $\omega_0 = \eta_1^* \oplus \omega_1$ and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0)$ to $L_{\mathcal{H}_1}$.
  - Otherwise, return $\omega_0 \xleftarrow{\$} \{0,1\}^\lambda$ and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0)$ to $L_{\mathcal{H}_1}$.
- On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{pw}_j, \mathsf{server})$ from $\mathcal{F}$:
  1. Wait $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ from $\mathcal{A}$. When it comes, generate $(\mathsf{ek}'_0, \mathsf{dk}'_0) \xleftarrow{\$} \mathrm{GEN}(\mathsf{pp})$ and $(\mathsf{pek}'_1, \mathsf{pdk}'_1) \xleftarrow{\$} \mathrm{GEN}(\mathsf{pp})$. In addition, send $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_j$.
  2. Sample $\eta_1^*|\eta_2^*|\eta_3^* \xleftarrow{\$} \{0,1\}^{3\lambda}$, and append $(\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{dk}'_0, \mathsf{pdk}'_1, \eta_1^*|\eta_2^*|\eta_3^*, \perp)$ to the list L_{server}.
  3. Continue execute the sub-protocol $\mathrm{Prot}_{\mathrm{PAKE}}$ using $\eta_1^*|\eta_2^*|\eta_3^*$ as $\mathcal{P}_j$'s input, until this sub-protocol completes with an output ϑ_j:
     - If msg was not output by simulation of $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$, $(\mathsf{sid}, j, \mathsf{pw}^*) \in L_{\mathsf{TestPwd,s}}$, and $\mathsf{pw}^* = \mathsf{pw}_j$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}$, where SK^* is computed as the protocol description.
       //Note that from dk'_0 *and* pdk'_1*, the simulator can compute* $\mathsf{k}_0 = \mathrm{KEY}(\mathsf{dk}'_0, \hat{f}_1(\mathsf{pw}, \mathsf{pek}_0))$ *and* $\mathsf{dk}_1^* = \mathrm{EXT}(\mathsf{pw}, \mathsf{pdk}'_1, \hat{f}_0)$, $\mathsf{k}_1 = \mathrm{KEY}(\mathsf{dk}_1^*, \mathsf{ek}_1)$*. Then query the corresponding RO query to retrieve* $\omega_i, \hat{\omega}_i$ *and* $\mu_i|\hat{\mu}_i|\theta_i$*.*
     - Else if msg was not output by simulation of $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$ but $(\mathsf{sid}, j, \mathsf{pw}^*) \notin L_{\mathsf{TestPwd,s}}$, send $(\mathsf{RegisterTest}, \mathsf{sid}, \mathcal{P}_j)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}$ in order, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^\lambda$. In addition, update the associated record in L_{server} as $(\mathsf{sid}, *, j, *, *, *, *, *, \vartheta_j)$ and separately record SK^*.
     - Otherwise, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}$, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^\lambda$.

Claim 15. $\mathsf{EXEC}_{5,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{4,\mathcal{A},\mathcal{Z}}$.

Proof. Due to the perfect hiding property of the underlying homomorphic NIKE scheme, the modification on the generation of pek'_1 does not introduce any difference.

Note that, $\mathcal{S}$ simulates the honest server instance without making any $\mathcal{F}_{\mathrm{RO}}$ queries. There are two cases that need to be distinguished: *(i)* when msg was not output by the simulation of $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$, the subsequent $\mathcal{F}_{\mathrm{RO}}$ queries made by $\mathcal{A}$ is backpatched to be consistent with these random values $\eta^*|\hat{\eta}^*|\hat{\mu}^*$ chosen during the course; *(ii)* when msg was output by the simulation of $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$, no backpatch strategies are used. Then, we consider the following two cases.

Case I: msg was not output by the simulation of $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$.

In this case, the introduced difference would be noticed, if $\mathcal{A}$ has previously used the pair $(\mathsf{pek}_0, \mathsf{ek}'_0)$ in its $\mathcal{F}_{\mathrm{RO}_1}$ queries before receiving ek'_0, or it made two different password-guesses against a single server instance. The latter sub case means that $\mathcal{A}$ made two $\mathcal{F}_{\mathrm{RO}_1}$ queries on $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}$ and $\mathsf{sid}|i|j|\mathsf{pek}_0|\mathsf{ek}'_0|\tilde{\mathsf{k}}_0|\tilde{\mathsf{pw}}$, such that $\mathsf{pw} \neq \tilde{\mathsf{pw}}$, $\mathsf{k}_0 = \mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}_0), \mathsf{ek}'_0)$ and $\tilde{\mathsf{k}}_0 = \mathbf{Key}(\hat{f}_0(\tilde{\mathsf{pw}}, \mathsf{pek}_0), \mathsf{ek}'_0)$.

The first event only occurs with negligible probability since ek'_0 is randomly generated. Then, we show that if the second event occurs, we may construct a PPT algorithm $\mathcal{B}$ to break the Type-II(ii) intractability of the underlying homomorphic NIKE scheme. In particular, given a challenge ek^*, $\mathcal{B}$ simulate $\mathbf{Exec}5$ with modifications as follows:

- When $\Pi^{\mathsf{sid}}_{\mathcal{P}_j}$ receiving msg, set $\mathsf{ek}'_0 = \mathsf{ek}^* \cdot \mathsf{ek}^{\mathsf{sid}}_j$, where $(\mathsf{ek}^{\mathsf{sid}}_j, \mathsf{dk}^{\mathsf{sid}}_j) \xleftarrow{\$} \mathrm{GEN}(\mathsf{pp})$.
- When $\mathcal{A}$ finishes, for every pair of $\mathcal{F}_{\mathrm{RO}_1}$ queries on $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}$ and $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\tilde{\mathsf{k}}_0|\tilde{\mathsf{pw}}$, and ek'_0 was output by $\Pi^{\mathsf{sid}}_{\mathcal{P}_j}$, do as follows:
  - Compute $\bar{\mathsf{k}}_1 = \mathrm{KEY}(\mathsf{dk}^{\mathsf{sid}}_j, \hat{f}_0(\mathsf{pw}, \mathsf{pek}_0))$ and $\bar{\mathsf{k}}_2 = \mathrm{KEY}(\mathsf{dk}^{\mathsf{sid}}_j, \hat{f}_0(\tilde{\mathsf{pw}}, \mathsf{pek}_0))$.

– Compute $\mathsf{k}_0^* = \mathsf{k}_0/\bar{k}_1$ and $\mathsf{k}_1^* = \tilde{\mathsf{k}}_0/\bar{\mathsf{k}}_2$.

– Add $(\mathsf{pek}_0, 0, (\mathsf{pw}, \mathsf{k}_0^*), (\tilde{\mathsf{pw}}, \mathsf{k}_1^*))$ into the list of possible solutions.

Note that if the event in question occurs, the following two equations must hold:

$$\begin{cases} \mathsf{k}_0 = \mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}_0), \mathsf{ek}^* \cdot \mathsf{ek}_j^{\mathsf{sid}}); \\ \tilde{\mathsf{k}}_0 = \mathbf{Key}(\hat{f}_0(\tilde{\mathsf{pw}}, \mathsf{pek}_0), \mathsf{ek}^* \cdot \mathsf{ek}_j^{\mathsf{sid}}). \end{cases}$$

If the above-mentioned event occurs, the correct answer was added to the list. Thus, it only occurs with negligible probability.

Case II: msg was output by the simulation of $\Pi_{\mathcal{P}_i}^{\mathsf{sid}}$.

In this case, both pek_0 and ek_0' are simulated. We can use the same proof idea as in $\mathbf{Expt}_5$ to prove that the adversary cannot query $\mathcal{F}_{\text{RO}}$ for correct $\omega_{j,0}$, and thus cannot query $\mathcal{F}_{\text{RO}}$ for correct $\mu_j|\hat{\mu}_j|\theta_j$. These values are indistinguishable from uniform in the view of the adversary.

To summarize the above, this claim holds. □

Execution $\mathbf{Exec}_6$: Simulating An Honest Client Instance with Corrupt Partner

Modifications to $\mathcal{S}$: When simulating an honest client instance with corrupt partner, do as follows:

- Initialize three new lists, L_{client}, $L_{\mathsf{TestPwd,c}}$. In particular, use L_{client} to record $\mathcal{S}$'s simulated information specific to honest client instances, and use $L_{\mathsf{TestPwd,c}}$ to record password-guesses against honest client instances.
- On $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw})$ to $\mathcal{F}_{\text{RO}_2}$, where $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ and $\mathsf{msg}' = (\mathsf{ek}_0', \mathsf{pek}_1')$:
  - If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}, \omega_1) \in L_{\mathcal{H}_2}$, return ω_1.
  - If $(\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{pdk}_0, \mathsf{dk}_1, \eta_1^*|\eta_2^*|\eta_3^*, \vartheta^*) \in L_{\mathsf{client}} \wedge \mathsf{k}_1$ $\triangle$-*satisfies* $(\mathsf{pw}, \mathsf{dk}_1, \mathsf{pek}_1', \hat{f}_1)$:
    - If $\mathsf{msg}' = \perp \wedge \eta_1^*|\eta_2^*|\eta_3^* = \perp \wedge \vartheta^* = \perp$, then append $(\mathsf{sid}, i, \mathsf{pw}, \mathsf{msg}')$ to $L_{\mathsf{TestPwd,c}}$.
    - If $\mathsf{msg}' \neq \perp \wedge \eta_1^*|\eta_2^*|\eta_3^* \neq \perp \wedge \vartheta^* = \perp$, then append $(\mathsf{sid}, i, \mathsf{pw}, \perp)$ to $L_{\mathsf{TestPwd,c}}$.
    - If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0) \in L_{\mathcal{H}_1} \wedge \mathsf{k}_0$ $\triangledown$-*satisfies* $(\mathsf{pw}, \mathsf{pdk}_0, \mathsf{ek}_0', \hat{f}_0)$, then return $\omega_1 = \eta_1^* \oplus \omega_0$ and append $(\mathsf{sid}|i|j|\mathsf{ek}_1|\mathsf{pek}_1'|\mathsf{k}_1|\mathsf{pw}, \omega_1)$ to $L_{\mathcal{H}_2}$.
  - Otherwise, return $\omega_1 \xleftarrow{\$} \{0,1\}^\lambda$ and append $(\mathsf{sid}|i|j|\mathsf{ek}_1|\mathsf{pek}_1'|\mathsf{k}_1|\mathsf{pw}, \omega_1)$ to $L_{\mathcal{H}_2}$.
- On $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega|\hat{\omega}|\mathsf{pw})$ to $\mathcal{F}_{\text{RO}_3}$, where $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ and $\mathsf{msg}' = (\mathsf{ek}_0', \mathsf{pek}_1')$:
  - If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega|\hat{\omega}|\mathsf{pw}, \mu|\hat{\mu}|\theta) \in L_{\mathcal{H}_3}$, return $\mu|\hat{\mu}|\theta$.
  - Else if one of the following two cases holds:

$$\begin{cases} (\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{pdk}_0, \mathsf{dk}_1, \eta_1^*|\eta_2^*|\eta_3^*, *) \in L_{\mathsf{client}} \wedge \eta_1^* = \omega \oplus \hat{\omega}; \\ (\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega) \in L_{\mathcal{H}_1} \wedge (\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}, \hat{\omega}) \in L_{\mathcal{H}_2}; \\ \mathsf{k}_0 \text{ } \triangledown\text{-}\textit{satisfies} \text{ } (\mathsf{pw}, \mathsf{pdk}_0, \mathsf{ek}_0', \hat{f}_0) \wedge \mathsf{k}_1 \text{ } \triangle\text{-}\textit{satisfies} \text{ } (\mathsf{pw}, \mathsf{dk}_1, \mathsf{pek}_1', \hat{f}_1). \end{cases}$$

$$\begin{cases} (\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{dk}_0', \mathsf{pdk}_1', \eta_1^*|\eta_2^*|\eta_3^*, *, *) \in L_{\mathsf{server}} \wedge \eta_1^* = \omega \oplus \hat{\omega}; \\ (\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega) \in L_{\mathcal{H}_1} \wedge (\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}, \hat{\omega}) \in L_{\mathcal{H}_2}; \\ \mathsf{k}_0 \text{ } \triangle\text{-}\textit{satisfies} \text{ } (\mathsf{pw}, \mathsf{dk}_0', \mathsf{pek}_0, \hat{f}_0) \wedge \mathsf{k}_1 \text{ } \triangledown\text{-}\textit{satisfies} \text{ } (\mathsf{pw}, \mathsf{pdk}_1', \mathsf{ek}_1, \hat{f}_1). \end{cases}$$

    Set $\mu = \eta_2^* \odot \mathsf{k}_0$ and $\hat{\mu} = \eta_3^* \odot \mathsf{k}_1$. If the last variable $\vartheta^* \neq \perp$ in the associated record in L_{client} (resp., L_{server}) and $\mathsf{pw} = \mathsf{pw}_i$ (resp., $\mathsf{pw} = \mathsf{pw}_j$), set $\theta = \mathsf{SK}^* \oplus \vartheta^*$; otherwise, set $\theta \xleftarrow{\$} \{0,1\}^\lambda$. Return $\mu|\hat{\mu}|\theta$ with and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega|\hat{\omega}|\mathsf{pw}, \mu|\hat{\mu}|\theta)$ to $L_{\mathcal{H}_3}$.
  - Otherwise, return $\mu|\hat{\mu}|\theta \xleftarrow{\$} \{0,1\}^{3\lambda}$ and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega|\hat{\omega}|\mathsf{pw}, \mu|\hat{\mu}|\theta)$ to the list $L_{\mathcal{H}_3}$.
- On $\mathsf{msg}' = (\mathsf{ek}_0', \mathsf{pek}_1')$ from $\mathcal{A}$:
  1. If $(\mathsf{sid}, i, *, *, \perp, *, *, *, *) \in L_{\mathsf{client}}$, update this record as $(\mathsf{sid}, i, *, *, \mathsf{msg}', *, *, *, *)$.
  2. If msg' was not output by the simulation of $\Pi_{\mathcal{P}_j}^{\mathsf{sid}}$:
     (a) If $(\mathsf{sid}, i, \mathsf{pw}^*, \mathsf{msg}') \in L_{\mathsf{TestPwd,c}}$:

i. If $\mathsf{pw}^* = \mathsf{pw}_i$, execute the sub-protocol $\mathrm{Prot}_{\mathrm{PAKE}}$ using the correct input $(\omega_{i,0} \oplus \omega\hat{}_{i,0})|(\mu_i \odot \mathsf{k}_0)|(\hat{\mu}_i \odot \mathsf{k}_1)$.
//Note that from pdk_0 *and* dk_1*, the simulator can compute* $\mathsf{dk}_0^* = \textsc{Ext}(\mathsf{pw}, \mathsf{pdk}_0, \hat{f}_0)$, $\mathsf{k}_0 = \textsc{Key}(\mathsf{dk}_0^*, \mathsf{ek}_0')$ *and* $\mathsf{k}_1 = \textsc{Key}(\mathsf{dk}_1, \hat{f}_1(\mathsf{pw}, \mathsf{pek}_1'))$*. Then query ROs to retrieve* $\omega_{i,0}, \omega_{i,1}$ *and* $\mu_i|\hat{\mu}_i|\theta_i$*.*
Until $\mathrm{Prot}_{\mathrm{PAKE}}$ completes with an output ϑ_i, set $\mathsf{SK}^* = \theta_i \oplus \vartheta_i$ and send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}$.

ii. Otherwise, execute the sub-protocol $\mathrm{Prot}_{\mathrm{PAKE}}$ using a random value chosen from $\{0,1\}^{3\lambda}$. Until $\mathrm{Prot}_{\mathrm{PAKE}}$ completes with an output, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}$, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^{\lambda}$.

(b) Otherwise, execute the sub-protocol $\mathrm{Prot}_{\mathrm{PAKE}}$ using $\eta_1^*|\eta_2^*|\eta_3^* \xleftarrow{\$} \{0,1\}^{3\lambda}$ as $\mathcal{P}_i$'s input and update the associated record in L_{client} as $(\mathsf{sid}, i, *, *, \mathsf{msg}', *, *, \eta_1^*|\eta_2^*|\eta_3^*, *)$. Until $\mathrm{Prot}_{\mathrm{PAKE}}$ completes with an output ϑ_i:

– If now there exists $(\mathsf{sid}, i, \mathsf{pw}^*, \perp) \in L_{\mathsf{TestPwd,C}}$:

i. If $\mathsf{pw}^* = \mathsf{pw}_i$, retrieve θ_i using the same method in Case (a)-i.

ii. Otherwise, set $\theta_i \xleftarrow{\$} \{0,1\}^{\lambda}$.

Finally, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}$, where $\mathsf{SK}^* = \theta_i \oplus \vartheta_i$.

– Otherwise, send $(\mathsf{RegisterTest}, \mathsf{sid}, \mathcal{P}_i)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}$ in order, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^{\lambda}$. In addition, update the associated record in L_{client} as $(\mathsf{sid}, i, *, *, *, *, *, *, \vartheta_i)$ and separately record SK^*.

3. Otherwise, continue the simulation. If it concludes with a session key, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}$, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^{\lambda}$.

Claim 16. $\mathsf{EXEC}_{6,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{5,\mathcal{A},\mathcal{Z}}$.

Proof. First, due to the perfect hiding property of the underlying homomorphic NIKE scheme, the modification on the generation of pek_0 does not introduce any difference. Then, we finish the proof by considering the following three cases.

Case I: msg' was not output by the simulation of $\Pi_{\mathcal{P}_j}^{\mathsf{sid}}$.

First, we will show that for each pek_1', at most one such record will be found in $L_{\mathsf{TestPwd,C}}$. Otherwise, the adversary may notice the introduced difference. By this, we want to prove the adversary $\mathcal{A}$ cannot make two different password guesses against a single client instance.

In order to deduce a contradiction, assume $\mathcal{A}$ can make this event occurs, there should exist two different records $(\mathsf{sid}, i, \mathsf{pw}, \mathsf{pek}_1') \in L_{\mathsf{TestPwd,C}}$ and $(\mathsf{sid}, i, \tilde{\mathsf{pw}}, \mathsf{pek}_1') \in L_{\mathsf{TestPwd,C}}$ with $\mathsf{pw} \neq \tilde{\mathsf{pw}}$. That also means the adversary made two $\mathcal{F}_{\mathrm{RO}_2}$ queries on $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}$ and $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\tilde{\mathsf{k}}_1|\tilde{\mathsf{pw}}$, such that $\mathsf{pw} \neq \tilde{\mathsf{pw}}$, $\mathsf{k}_1 = \mathbf{Key}(\mathsf{ek}_1, \hat{f}_1(\mathsf{pw}, \mathsf{pek}_1'),)$ and $\tilde{\mathsf{k}}_1 = \mathbf{Key}(\mathsf{ek}_1, \hat{f}_1(\tilde{\mathsf{pw}}, \mathsf{pek}_1'))$.

We then show how to construct a PPT algorithm $\mathcal{B}$ to break the Intractability-II (ii) of the underlying homomorphic NIKE scheme. In particular, given a challenge ek^*, $\mathcal{B}$ simulate $\mathbf{Exec}6$ with modifications as follows:

- When $\Pi_{\mathcal{P}_i}^{\mathsf{sid}}$ receiving $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{pw}_i, \mathsf{client})$ from $\mathcal{F}$, set $\mathsf{ek}_1 = \mathsf{ek}^* \cdot \mathsf{ek}_i^{\mathsf{sid}}$, where $(\mathsf{ek}_i^{\mathsf{sid}}, \mathsf{dk}_i^{\mathsf{sid}}) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$.
- When $\mathcal{A}$ stops, for every pair of $\mathcal{F}_{\mathrm{RO}_2}$ queries on $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}$ and $\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\tilde{\mathsf{k}}_1|\tilde{\mathsf{pw}}$, where pek_1' was received from the adversary $\mathcal{A}$ and ek_1 was actually simulated as before, do as follows:
  - Compute $\bar{\mathsf{k}}_1 = \textsc{Key}(\mathsf{dk}_i^{\mathsf{sid}}, \hat{f}_1(\mathsf{pw}, \mathsf{pek}_1'))$ and $\bar{\mathsf{k}}_2 = \textsc{Key}(\mathsf{dk}_i^{\mathsf{sid}}, \hat{f}_1(\tilde{\mathsf{pw}}, \mathsf{pek}_1'))$.
  - Compute $\mathsf{k}_0^* = \mathsf{k}_1/\bar{k}_1$ and $\mathsf{k}_1^* = \tilde{\mathsf{k}}_1/\bar{\mathsf{k}}_2$.
  - Add $(\mathsf{pek}_1', 1, (\mathsf{pw}, \mathsf{k}_0^*), (\tilde{\mathsf{pw}}, \mathsf{k}_1^*))$ into the list of possible solutions.

Note that if the event in question occurs, the following two equations hold:

$$\begin{cases} \mathsf{k}_1 = \mathbf{Key}(\mathsf{ek}^* \cdot \mathsf{ek}_i^{\mathsf{sid}}, \hat{f}_1(\mathsf{pw}, \mathsf{pek}_1')); \\ \tilde{\mathsf{k}}_1 = \mathbf{Key}(\mathsf{ek}^* \cdot \mathsf{ek}_i^{\mathsf{sid}}, \hat{f}_1(\tilde{\mathsf{pw}}, \mathsf{pek}_1')). \end{cases}$$

If the event in question occurs, $\mathcal{B}$ adds the correct answer to the list. Thus, this event only occurs with negligible probability.

Case II: msg' was output by the simulation of $\Pi^{\mathsf{sid}}_{\mathcal{P}_j}$.

In this case, both $(\mathsf{ek}_1, \mathsf{pek}'_1)$ as simulated. We can use the same proof idea as in $\mathbf{Expt}_4$ to prove that the adversary cannot query $\mathcal{F}_{\text{RO}}$ for correct $\omega_{i,1}$, and thus cannot query $\mathcal{F}_{\text{RO}}$ for correct $\mu_i|\hat{\mu}_i|\theta_i$. These values are indistinguishable from uniform in the view of the adversary $\mathcal{A}$.

Summarize the above, this claim holds. □

Execution $\mathbf{Exec}_7$: Allowing $\mathcal{F}$ to Set Session Keys for An Honest Instance with Corrupt Partner

Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions Fig. 4 to set session keys for an honest party ($\mathcal{P}_i$ or $\mathcal{P}_j$) with corrupt partner.

Claim 17. $\mathsf{EXEC}_{7,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{6,\mathcal{A},\mathcal{Z}}$.

Proof. If there exits $(\mathsf{sid}, i, \mathsf{pw}^*, *) \in L_{\mathsf{TestPwd,C}}$, or $(\mathsf{sid}, j, \mathsf{pw}^*) \in L_{\mathsf{TestPwd,S}}$, $\mathcal{S}$ extracts the adversary's input. If pw^* is a correct guess. The simulator $\mathcal{S}$ can compute the exact session key as the adversary do, with knowledge of pw^*. Otherwise, $\mathcal{A}$ has not queried $\mathcal{F}_{\text{RO}}$ on correct values, the session key is random. □

Execution $\mathbf{Exec}_8$: Adding TestPwd and LateTestPwd Interfaces, Removing Password Forwarding

Modifications to $\mathcal{F}$: We now allow $\mathcal{F}$ to follow the instructions Fig. 4 to poss TestPwd and LateTestPwd queries. $\mathcal{F}$ still forwards NewSession queries from the dummy parties but not in their enitemizey to $\mathcal{S}$, as the password is removed.

Modifications to $\mathcal{S}$: When simulating RO queries, the local password equality tests of pw^* and pw_i (resp., pw_j) are replaced by sending $\mathsf{LateTestPwd}(\mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ (resp., $\mathsf{LateTestPwd}(\mathsf{sid}, \mathcal{P}_j, \mathsf{pw}^*)$) to $\mathcal{F}$. In addition, $\mathcal{S}$ no longer stores SK^* in the associated cases. Instead, the replied SK^* (that is received from $\mathcal{F}$ via invoking LateTestPwd) is used. All other tests of pw^* and pw_i (resp., pw_j) are replaced by sending $\mathsf{TestPwd}(\mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ (resp.,$\mathsf{TestPwd}(\mathsf{sid}, \mathcal{P}_j, \mathsf{pw}^*)$) to $\mathcal{F}$. In particular, the simulator will no longer differentiate between equality; instead, it will always treat them as equal to continue the simulation.

Claim 18. $\mathsf{EXEC}_{8,\mathcal{A},\mathcal{Z}} \overset{c}{\approx} \mathsf{EXEC}_{7,\mathcal{A},\mathcal{Z}}$.

Proof. From $\mathbf{Exec}_6$, $\mathcal{S}$ is unnecessary to use pw_i or pw_j to simulate the messages expected to be sent to $\mathcal{A}$. Therefore, the internal modifications of invoking the TestPwd and LateTestPwd interfaces to replace local password equality tests do not introduce any observable difference.

□

In the final execution, $\mathcal{F}$ is equivalent to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$. The simulator $\mathcal{S}$ is constructed as shown in Fig. 13 and Fig. 14, where does not need to use the passwords. In addition, the simulator perfectly simulates the ideal world except for the cases where it aborts, which we have already shown to happen with negligible probability.

Combine all above discussions together, we conclude that

$$\mathsf{IDEAL}^{\mathcal{G}_{\text{CRS}}}_{\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}},\mathcal{S},\mathcal{Z}} \overset{c}{\approx} \mathsf{REAL}^{\mathcal{G}_{\text{CRS}},\mathcal{F}_{\text{RO}}}_{\pi,\mathcal{A},\mathcal{Z}}.$$

D Proof of Theorem 6

Now, we complete the proof of Theorem 6, by showing the indistinguishability between the real world and ideal world.

In the real world: The input password pw_i for an honest party $\mathcal{P}_i$ is phrased as $\mathsf{pw}^1_i|\ldots|\mathsf{pw}^n_i$, where each sub-password is used as input to the sub-protocol $\text{Prot}_{\text{UNIT}}$. If $\mathcal{A}$ wants to make a correct password guess for sid of the whole protocol, then it must make correct password guesses for ssid_k for $k \in [n]$ of the sub-protocol $\text{Prot}_{\text{UNIT}}$, simultaneously. Note that there are two strategies for $\mathcal{A}$ to test its password guess $\widehat{\mathsf{pw}}^k_i$ for a sub-session ssid_k:

On $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw})$ to $\mathcal{F}_{\mathrm{RO}_1}$, where $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ and $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$:

- If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0) \in L_{\mathcal{H}_1}$, return ω_0.
- If $(\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{dk}'_0, \mathsf{pdk}'_1, \eta_1^*|\eta_2^*|\eta_3^*, \vartheta^*) \in L_{\mathsf{server}} \wedge \mathsf{k}_0$ $\triangle$-*satisfies* $(\mathsf{pw}, \mathsf{dk}'_0, \mathsf{pek}_0, \hat{f}_0)$:
  - If $\vartheta^* = \perp$, then append $(\mathsf{sid}, j, \mathsf{pw})$ to $L_{\mathsf{TestPwd,S}}$.
  - Else if $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}, \omega_1) \in L_{\mathcal{H}_2} \wedge \mathsf{k}_1 \triangledown$-*satisfies* $(\mathsf{pw}, \mathsf{pdk}'_1, \mathsf{ek}_1, \hat{f}_1)$, then return $\omega_0 = \eta_1^* \oplus \omega_1$ and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0)$ to $L_{\mathcal{H}_1}$.
- Otherwise, return $\omega_0 \xleftarrow{\$} \{0,1\}^\lambda$ and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0)$ to $L_{\mathcal{H}_1}$.

On $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw})$ to $\mathcal{F}_{\mathrm{RO}_2}$, where $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ and $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$:

- If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}, \omega_1) \in L_{\mathcal{H}_2}$, return ω_1.
- If $(\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{pdk}_0, \mathsf{dk}_1, \eta_1^*|\eta_2^*|\eta_3^*, \vartheta^*) \in L_{\mathsf{client}} \wedge \mathsf{k}_1$ $\triangle$-*satisfies* $(\mathsf{pw}, \mathsf{dk}_1, \mathsf{pek}'_1, \hat{f}_1)$:
  - If $\mathsf{msg}' = \perp \wedge \eta_1^*|\eta_2^*|\eta_3^* = \perp \wedge \vartheta^* = \perp$, then append $(\mathsf{sid}, i, \mathsf{pw}, \mathsf{msg}')$ to $L_{\mathsf{TestPwd,C}}$.
  - If $\mathsf{msg}' \neq \perp \wedge \eta_1^*|\eta_2^*|\eta_3^* \neq \perp \wedge \vartheta^* = \perp$, then append $(\mathsf{sid}, i, \mathsf{pw}, \perp)$ to $L_{\mathsf{TestPwd,C}}$.
  - If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0) \in L_{\mathcal{H}_1} \wedge \mathsf{k}_0 \triangledown$-*satisfies* $(\mathsf{pw}, \mathsf{pdk}_0, \mathsf{ek}'_0, \hat{f}_0)$, then return $\omega_1 = \eta_1^* \oplus \omega_0$ and append $(\mathsf{sid}|i|j|\mathsf{ek}_1|\mathsf{pek}'_1|\mathsf{k}_1|\mathsf{pw}, \omega_1)$ to $L_{\mathcal{H}_2}$.
- Otherwise, return $\omega_1 \xleftarrow{\$} \{0,1\}^\lambda$ and append $(\mathsf{sid}|i|j|\mathsf{ek}_1|\mathsf{pek}'_1|\mathsf{k}_1|\mathsf{pw}, \omega_1)$ to $L_{\mathcal{H}_2}$.

On $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega_0|\omega_1|\mathsf{pw})$ to $\mathcal{F}_{\mathrm{RO}_3}$, where $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ and $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$:

- If $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega_0|\omega_1|\mathsf{pw}, \mu|\hat{\mu}|\theta) \in L_{\mathcal{H}_3}$, return $\mu|\hat{\mu}|\theta$.
- Else if one of the following two cases holds:

$$\text{Case 1} \begin{cases} (\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{pdk}_0, \mathsf{dk}_1, \eta_1^*|\eta_2^*|\eta_3^*, *) \in L_{\mathsf{client}} \wedge \eta_1^* = \omega_0 \oplus \omega_1; \\ (\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0) \in L_{\mathcal{H}_1} \wedge (\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}, \omega_1) \in L_{\mathcal{H}_2}; \\ \mathsf{k}_0 \triangledown\text{-}\textit{satisfies}\ (\mathsf{pw}, \mathsf{pdk}_0, \mathsf{ek}'_0, \hat{f}_0) \wedge \mathsf{k}_1 \triangle\text{-}\textit{satisfies}\ (\mathsf{pw}, \mathsf{dk}_1, \mathsf{pek}'_1, \hat{f}_1). \end{cases}$$

$$\text{Case 2} \begin{cases} (\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{dk}'_0, \mathsf{pdk}'_1, \eta_1^*|\eta_2^*|\eta_3^*, *) \in L_{\mathsf{server}} \wedge \eta_1^* = \omega_0 \oplus \omega_1; \\ (\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_0|\mathsf{pw}, \omega_0) \in L_{\mathcal{H}_1} \wedge (\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\mathsf{k}_1|\mathsf{pw}, \omega_1) \in L_{\mathcal{H}_2}; \\ \mathsf{k}_0 \triangle\text{-}\textit{satisfies}\ (\mathsf{pw}, \mathsf{dk}'_0, \mathsf{pek}_0, \hat{f}_0) \wedge \mathsf{k}_1 \triangledown\text{-}\textit{satisfies}\ (\mathsf{pw}, \mathsf{pdk}'_1, \mathsf{ek}_1, \hat{f}_1). \end{cases}$$

  Set $\mu = \eta_2^* \odot \mathsf{k}_0$ and $\hat{\mu} = \eta_3^* \odot \mathsf{k}_1$. If the last variable $\vartheta^* \neq \perp$ in the associated record in L_{client} (resp., L_{server}), send $(\mathsf{LateTestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw})$ (resp., $(\mathsf{LateTestPwd}, \mathsf{sid}, \mathcal{P}_j, \mathsf{pw})$) to $\mathcal{F}_{\text{le-mPAKE}}^{\mathcal{D}}$. After receiving a reply SK^*, set $\theta = \mathsf{SK}^* \oplus \vartheta^*$; otherwise, set $\theta \xleftarrow{\$} \{0,1\}^\lambda$. Return $\mu|\hat{\mu}|\theta$ with and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega_0|\omega_1|\mathsf{pw}, \mu|\hat{\mu}|\theta)$ to $L_{\mathcal{H}_3}$.
- Otherwise, return $\mu|\hat{\mu}|\theta \xleftarrow{\$} \{0,1\}^{3\lambda}$ and append $(\mathsf{sid}|i|j|\mathsf{msg}|\mathsf{msg}'|\omega_0|\omega_1|\mathsf{pw}, \mu|\hat{\mu}|\theta)$ to $L_{\mathcal{H}_3}$.

Figure 13: Simulation of $\mathcal{F}_{\mathrm{RO}}$ queries when the RO setup is functional and the CRS setup falls back to gCRS

(1) First query $(\mathsf{TestPwd}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{pw}}_i^k)$, then query $(\mathsf{NewKey}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{SK}}_i^k)$ to make this session computes the session key;
(2) First query $(\mathsf{RegisterTest}, \mathsf{ssid}_k, \mathcal{P}_i)$ to register a future password guess, then query $(\mathsf{NewKey}, \mathsf{ssid}_k, \mathcal{P}_i, *)$ to make the target the session computing the session key, finally query $(\mathsf{LateTestPwd}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{pw}}_i^k)$.

For Case-(1), if $\widehat{\mathsf{pw}}_i^k$ is a correct password guess, $\mathcal{A}$ determines the session key of ssid_k as $\widehat{\mathsf{SK}}_i^k$, otherwise that will be randomly choosen. While for Case-(2), the session key of ssid_k is always randomly choosen; but if $\widehat{\mathsf{pw}}_i^k$ is a correct password guess, $\mathcal{A}$ should receive the same sub-session key as the honest party.

Recall that $\mathcal{P}_i$'s output sesssion key is computed by XORing all sub-session keys from executing the sub-protocol $\mathrm{Prot}_{\mathrm{UNIT}}$. There are three associated cases:

(1) If $\mathcal{A}$ correctly query $(\mathsf{TestPwd}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{pw}}_i^k)$ for all $k \in [n]$ simultaneously, it can determine $\mathcal{P}_i$'s output session key;
(2) If $\mathcal{A}$ correctly query $(\mathsf{TestPwd}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{pw}}_i^k)$ for $k \in T \subset [n]$ and $(\mathsf{LateTestPwd}, \mathsf{ssid}_{k'}, \mathcal{P}_i, \widehat{\mathsf{pw}}_i^{k'})$ for

Simulating an honest client instance $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathsf{client})$ from $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$:

1. Generate $(\mathsf{pek}_0, \mathsf{pdk}_0) \xleftarrow{\$} \text{GEN}(\mathsf{pp})$ and $(\mathsf{ek}_1, \mathsf{dk}_1) \xleftarrow{\$} \text{GEN}(\mathsf{pp})$.
2. Send $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_0)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_i$.
3. Append $(\mathsf{sid}, i, j, \mathsf{msg}, \perp, \mathsf{pdk}_0, \mathsf{dk}_1, \perp, \perp)$ to L_{client}.

On $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ from $\mathcal{A}$:

1. If $(\mathsf{sid}, i, *, *, \perp, *, *, *, *, *) \in L_{\mathsf{client}}$, update this record as $(\mathsf{sid}, i, *, *, \mathsf{msg}', *, *, *, *)$.
2. If msg' was not output by the simulation of $\Pi^{\mathsf{sid}}_{\mathcal{P}_j}$:
   (a) If $(\mathsf{sid}, i, \mathsf{pw}^*, \mathsf{msg}') \in L_{\mathsf{TestPwd,C}}$:
   First, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$. Then, execute the sub-protocol $\text{Prot}_{\text{PAKE}}$ using the input $(\omega_i \oplus \hat{\omega}_i)|(\mu_i \odot \mathsf{k}_0)|(\hat{\mu}_i \odot \mathsf{k}_1)$.
   // Note that from pdk_0 *and* dk_1*, the simulator can compute* $\mathsf{dk}^*_0 = \text{EXT}(\mathsf{pw}, \mathsf{pdk}_0, \hat{f}_0)$, $\mathsf{k}_0 = \text{KEY}(\mathsf{dk}^*_0, \mathsf{ek}'_0)$ *and* $\mathsf{k}_1 = \text{KEY}(\mathsf{dk}_1, \hat{f}_1(\mathsf{pw}, \mathsf{pek}'_1))$*. Then query ROs to retrieve* $\omega_i, \hat{\omega}_i$ *and* $\mu_i|\hat{\mu}_i|\theta_i$*.*
   Until the sub-protocol $\text{Prot}_{\text{PAKE}}$ completes with ϑ_i, set $\mathsf{SK}^* = \theta_i \oplus \vartheta_i$ and send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$.
   (b) Otherwise, execute the sub-protocol $\text{Prot}_{\text{PAKE}}$ using $\eta^*_1|\eta^*_2|\eta^*_3 \xleftarrow{\$} \{0,1\}^{6\lambda}$ as $\mathcal{P}_i$'s input and update the associated record in L_{client} as $(\mathsf{sid}, i, *, *, \mathsf{msg}', *, *, \eta^*_1|\eta^*_2|\eta^*_3, *)$. Until the sub-protocol $\text{Prot}_{\text{PAKE}}$ completes with an output ϑ_i:
   - If now there exists $(\mathsf{sid}, i, \mathsf{pw}^*, \perp) \in L_{\mathsf{TestPwd,C}}$, send $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$. In addition, retrieve θ_i using the same method as in Case 2.(a). Then, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$, where $\mathsf{SK}^* = \theta_i \oplus \vartheta_i$.
   - Otherwise, send $(\mathsf{RegisterTest}, \mathsf{sid}, \mathcal{P}_i)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$ in order, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^{\lambda}$. In addition, update the associated record in L_{client} as $(\mathsf{sid}, i, *, *, *, *, *, *, \vartheta_i)$.
3. Otherwise, continue the simulation. If it concludes with a session key, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^{\lambda}$.

Simulating an honest server instance $\Pi^{\mathsf{sid}}_{\mathcal{P}_j}$

On $(\mathsf{NewSession}, \mathsf{sid}, \mathcal{P}_j, \mathcal{P}_i, \mathsf{server})$ from $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$:

1. Wait $\mathsf{msg} = (\mathsf{pek}_0, \mathsf{ek}_1)$ from $\mathcal{A}$. When it comes, generate $(\mathsf{ek}'_0, \mathsf{dk}'_0) \xleftarrow{\$} \text{GEN}(\mathsf{pp})$ and $(\mathsf{pek}'_1, \mathsf{pdk}'_1) \xleftarrow{\$} \text{GEN}(\mathsf{pp})$. In addition, send $\mathsf{msg}' = (\mathsf{ek}'_0, \mathsf{pek}'_1)$ to $\mathcal{A}$ on behalf of $\mathcal{P}_j$.
2. Append $(\mathsf{sid}, i, j, \mathsf{msg}, \mathsf{msg}', \mathsf{dk}'_0, \mathsf{pdk}'_1, \eta^*_1|\eta^*_2|\eta^*_3, \perp)$ to L_{server}, where $\eta^*_1|\eta^*_2|\eta^*_3 \xleftarrow{\$} \{0,1\}^{3\lambda}$.
3. Execute the sub-protocol $\text{Prot}_{\text{PAKE}}$ using $\eta^*_1|\eta^*_2|\eta^*_3$ as $\mathcal{P}_j$'s input, until this sub-protocol completes with an output ϑ_j:
   - If msg was not output by simulation of $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$ and $(\mathsf{sid}, j, \mathsf{pw}^*) \in L_{\mathsf{TestPwd,S}}$, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$, where SK^* is computed as the protocol description.
     //Note that from dk'_0 *and* pdk'_1*, the simulator can compute* $\mathsf{k}_0 = \text{KEY}(\mathsf{dk}'_0, \hat{f}_1(\mathsf{pw}, \mathsf{pek}_0))$ *and* $\mathsf{dk}^*_1 = \text{EXT}(\mathsf{pw}, \mathsf{pdk}'_1, \hat{f}_0)$*,* $\mathsf{k}_1 = \text{KEY}(\mathsf{dk}^*_1, \mathsf{ek}_1)$*. Then query the corresponding RO query to retrieve* $\omega_i, \hat{\omega}_i$ *and* $\mu_i|\hat{\mu}_i|\theta_i$*.*
   - Else if msg was not output by simulation of $\Pi^{\mathsf{sid}}_{\mathcal{P}_i}$ but $(\mathsf{sid}, j, \mathsf{pw}^*) \notin L_{\mathsf{TestPwd,S}}$, send $(\mathsf{RegisterTest}, \mathsf{sid}, \mathcal{P}_j)$ and $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$ in order, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^{\lambda}$. In addition, update the associated record in L_{server} as $(\mathsf{sid}, *, j, *, *, *, *, *, \vartheta_j)$.
   - Otherwise, send $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_j, \mathsf{SK}^*)$ to $\mathcal{F}^{\mathcal{D}}_{\text{le-mPAKE}}$, where $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^{\lambda}$.

Figure 14: Simulation of honest instances when the RO setup is functional and the CRS setup falls back to gCRS

$k' \in [n] \setminus T$, $\mathcal{A}$ should reconstruct $\mathcal{P}_i$'s output session key.
(3) In all other cases, $\mathcal{P}_i$'s output session key is uniformly distributed.

In the ideal world: For each sub-session ssid_k with $k \in [n]$, the simulator $\mathcal{S}$ creates a record $\langle \mathsf{ssid}_k, \mathcal{P}_i, \mathcal{P}_j, \mathsf{role}, \mathsf{pw}_i^k, \mathsf{SK}_i^k \rangle$, where the last two variables pw_i^k and SK_i^k are updated as the simulation progresses:

- When such a record is created, both variables pw_i^k and SK_i^k are set as $\perp$.
- When $(\mathsf{TestPwd}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{pw}}_i^k)$ is queried, the variable pw_i^k is updated as $\widehat{\mathsf{pw}}_i^k$; in contrast, when $(\mathsf{RegisterTest}, \mathsf{ssid}_k, \mathcal{P}_i)$ is queried, the variable pw_i^k is updated as $\top$.
- When $(\mathsf{NewKey}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{SK}}_i^k)$ is queried: if $\mathsf{pw}_i^k \neq \perp \wedge \mathsf{pw}_i^k \neq \top$, the variable SK_i^k is updated as $\widehat{\mathsf{SK}}_i^k$; otherwise, it is updated as $\top$.
- When $(\mathsf{LateTestPwd}, \mathsf{ssid}_k, \mathcal{P}_i, \widehat{\mathsf{pw}}_i^k)$ is queried: Only when $\mathsf{pw}_i^k = \top \wedge \mathsf{SK}_i^k = \top$ holds, this query will not be ignored.

According to the assigned values of the two variables pw_i^k and SK_i^k, the simulator $\mathcal{S}$ can distinguish the adversary's different strategies of guessing the password of the whole protocol. Since $\mathsf{ssid}_k := \mathsf{sid}|i|j|k$, the simulator $\mathcal{S}$ can find all records correspongding to the same sid. Using $\mathsf{pw}_i^k \neq \perp$ for all $k \in [n]$, the simulator $\mathcal{S}$ can extract the adversary's password guess for sid: if at least one of these values equals to $\top$, $\mathcal{S}$ sends $(\mathsf{RegisterTest}, \mathsf{sid}, \mathcal{P}_i)$ to $\mathcal{F}$; otherwise, $\mathcal{S}$ sends $(\mathsf{TestPwd}, \mathsf{sid}, \mathcal{P}_i, \mathsf{pw}_i^1|\ldots|\mathsf{pw}_i^n)$ to $\mathcal{F}$. In addition, using $\mathsf{SK}_i^k \neq \perp$ for all $k \in [n]$, the simulator $\mathcal{S}$ can form a proper query $(\mathsf{NewKey}, \mathsf{sid}, \mathcal{P}_i, \mathsf{SK}^*)$ to $\mathcal{F}$: if at least one of these values equals to $\top$, $\mathsf{SK}^* \xleftarrow{\$} \{0,1\}^\lambda$; otherwise, $\mathsf{SK}^* := \bigoplus_{k \in [n]} \mathsf{SK}_i^k$.

In the case that $\mathcal{A}$ makes correct password guesses for all sub-sessions via $\mathsf{TestPwd}$ queries, $\mathcal{F}$ will output SK^*, determined by $\mathcal{A}$, to $\mathcal{P}_i$ as its session key. In the case that $\mathcal{A}$ makes correct password guesses for part of the sub-sessions via $\mathsf{LateTestPwd}$ queries and other part of the sub-sessions via $\mathsf{TestPwd}$ queries, the simulator $\mathcal{S}$ ensures that $\mathcal{A}$'s received sub-session keys in $\mathsf{LateTestPwd}$ queries are consistent with $\mathcal{P}_i$'s session key it received from $\mathcal{F}$. In all other cases, $\mathcal{F}$ will output a random value to $\mathcal{P}_i$ as its session key.

We can conlude that the simulator $\mathcal{S}$ shown in Fig. 8 perfectly simulates the view of $\mathcal{A}$ in the real world.

E Our Further Result in the BPR Framework

For thoroughness, we present a further result within the BPR framework. Firstly, our scheme's security can be established with a CRS setup. Secondly, even in the event of the CRS trapdoor being compromised to the adversary, its security remains provable within a RO setup.

We adhere to the overarching concept of initially executing a RO-based sub-protocol, followed by the execution of a CRS-based sub-protocol. However, we invoke them in a non black-box manner. For the RO-based sub-protocol, both parties only need to send a single message that commit their own input passwords. This protocol also provides mutual key confirmation mechanism. For the CRS-based sub-protocol, we employ a particular variant of the GK-design [GK10] to instantiate it, which directly provides mutual key confirmation mechanism and is relative efficient than other CRS-based PAKEs. In contrast to the GK-design [GK10], the primary distinction lies in the method of generating the two ciphertexts. In the variant we employed, the two ciphertexts encrypt two unequal but related values. This adjustment guarantees that the mutual key confirmation mechanism provided by the RO-based sub-protocol remains effective even if the adversary possesses knowledge of the CRS trapdoor.

E.1 Description of the construction

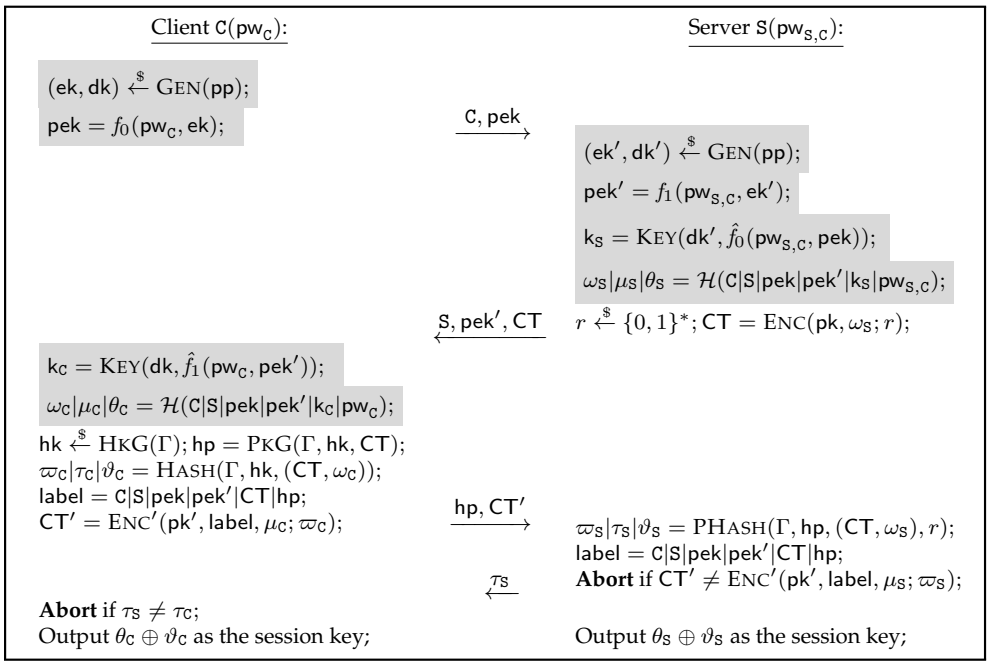
Our construction uses: A homomorphic NIKE scheme $\Pi = (\textsc{Setup}, \textsc{Gen}, \textsc{Key})$ with associated functions $\{f_\sigma, \hat{f}_\sigma\}_{\sigma \in \{0,1\}}$; A PKE scheme $\Sigma = (\mathrm{KG}, \textsc{Enc}, \textsc{Dec})$ with an associated SPHF scheme $\Omega_\Sigma = (\mathrm{PG}, \textsc{Hkg}, \textsc{Pkg}, \textsc{Hash}, \textsc{Phash})$; A labeled PKE scheme $\Sigma' = (\mathrm{KG}', \textsc{Enc}', \textsc{Dec}')$.

Public parameters. A setup phase is required to assign the security parameter λ and the following parameters:

– A public parameter $\mathsf{pp} \xleftarrow{\$} \textsc{Setup}(1^\lambda)$;

- Two public keys pk and pk' generated by $\mathrm{KG}(1^\lambda)$ and $\overline{\mathrm{KG}}(1^\lambda)$, respectively; and a parameter Γ generated by $\mathrm{PG}(1^\lambda)$. In particular, we use $\mathcal{L}$ to denote the language determined by (pk, Γ).
- Description for one hash function $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{6\lambda}$.

Protocol execution. Consider a client $\mathtt{C}$ and a server $\mathtt{S}$ holding their input passwords $\mathsf{pw}_\mathtt{C}$ and $\mathsf{pw}_{\mathtt{S},\mathtt{C}}$, respectively. They interact as shown in Fig. 15.



‡ The text with gray background embodies the RO-based subroutine we use to achieve BPR-secure PAKE with a weaker fine-grained CoR-setup, which is different from the one we used in the UC framework. In this process, each party only needs to send a single message that can be viewed as a commitment of its input password.

Figure 15: Our proposal in the BPR framework

E.2 Security analysis

For the protocol shown in Fig. 15, we have the following theorem.

Theorem 7. *Given the following building blocks:*

- Π *is a OW-secure homomorphic NIKE scheme with associated functions* $\{f_\sigma, \hat{f}_\sigma\}_{\sigma\in\{0,1\}}$*, where the perfect hiding property holds; In particular, the equivocability and Type-I intractability also holds in the RO model.*
- Σ *is a CPA-secure PKE scheme with an associated SPHF scheme* Ω_Σ*;*
- Σ' *is a CCA-secure labeled PKE scheme;*
- $\mathcal{H}$ *is a hash function.*

The protocol in Fig. 15 is secure in the BPR framework. In particular:

(1) It is BPR-secure with a CRS setup when $\mathcal{H}$ is sampled from a collision-resistant hash family;
(2) It is BPR-secure with a RO setup by modeling $\mathcal{H}$ as a RO, even if the CRS trapdoor is compromised by the adversary.

Proof. For simplicity, we divide the Send queries into five sub-queries:

- $\mathrm{Send}_0(\mathtt{C}, i, \mathtt{S})$ models that the adversary activates a client session $\Pi^i_\mathtt{C}$ with a partner $\mathtt{S}$.
- $\mathrm{Send}_1(\mathtt{S}, j, \mathsf{msg}_1)$ models that the adversary sends the first message (denoted as msg_1) to $\Pi^j_\mathtt{S}$.
- $\mathrm{Send}_2(\mathtt{C}, i, \mathsf{msg}_2)$ models that the adversary sends the second message (denoted as msg_2) to $\Pi^i_\mathtt{C}$.
- $\mathrm{Send}_3(\mathtt{S}, j, \mathsf{msg}_3)$ models that the adversary sends the third message (denoted as msg_3) to $\Pi^j_\mathtt{S}$.
- $\mathrm{Send}_4(\mathtt{C}, i, \mathsf{msg}_4)$ models thats the adversary sends the last message (denoted as msg_4) to $\Pi^i_\mathtt{C}$.

An adversary $\mathcal{A}$ may impersonate client or server to launch active attacks. Without loss of generality, we use the following terminologies:

- **Attack to a client instance** to indicate "$\mathcal{A}$ is impersonating a server $\mathtt{S}$ to attack a client session $\Pi^i_{\mathtt{C}}$". In this case, $\mathcal{A}$ should query $\mathrm{Send}_0(\mathtt{C}, i, \mathtt{S})$ that returns msg_1 and $\mathrm{Send}_2(\mathtt{C}, i, \mathsf{msg}_2)$ that returns msg_3 in order. Optionally, $\mathcal{A}$ should query $\mathrm{Send}_4(\mathtt{C}, i, \mathsf{msg}_4)$ to make the session $\Pi^i_{\mathtt{C}}$ complete.
- **Attack to a server instance** to indicate "$\mathcal{A}$ is impersonating a client $\mathtt{C}$ to attack a server session $\Pi^j_{\mathtt{S}}$". In this case, $\mathcal{A}$ should query $\mathrm{Send}_1(\mathtt{S}, j, \mathsf{msg}_1)$ that returns msg_2 and $\mathrm{Send}_3(\mathtt{S}, j, \mathsf{msg}_3)$ that returns msg_4 in order.

Note that the input message of each Send query may be adversarially generated or just a message replay. There exists a special case that the adversary attacks a protocol instance on both sides simultaneously and the input of querying $\mathrm{Send}_k(\cdot)$ is the output of querying $\mathrm{Send}_{k-1}(\cdot)$ for all $k \in [4]$.

Proof of Case-(1):

The proof follows the one by Groth and Katz [GK10]. The correctness can be easily verified. If both users honestly follow the protocol, they must conclude with the same session key, because it is easy to verify that $\theta_{\mathtt{C}} = \theta_{\mathtt{S}}$ and $\vartheta_{\mathtt{C}} = \vartheta_{\mathtt{S}}$ hold simultaneously in this case. We denote the number of $\mathrm{Execute}$ and Send queries made by $\mathcal{A}$ as n_{Execute} and n_{Send}, respectively. Next, we will use a series of experiments $\mathbf{Expt}_0, \ldots, \mathbf{Expt}_8$ to bound the advantage of $\mathcal{A}$. In particular, $\mathbf{Expt}_0$ is the original experiment, where an independently chosen random password will be fixed for each pair of users $(\mathtt{C}, \mathtt{S})$. We denote the advantage of $\mathcal{A}$ in experiment $\mathbf{Expt}_{\mathrm{i}}$ as:

$$\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},\mathrm{i}}(\lambda) := 2|\Pr[\mathcal{A} \text{ succeeds in } \mathbf{Expt}_{\mathrm{i}}] - 1|.$$

It immediately holds that $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},0} = \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},\pi}$.

Experiment $\mathbf{Expt}_1$: This experiment is same as $\mathbf{Expt}_0$ except that for answering an $\mathrm{Execute}(\mathtt{C}, i, \mathtt{S}, j)$ query, generate CT as an encryption of a random value $\omega \xleftarrow{\$} \{0,1\}^\lambda$ rather than $\omega_{\mathtt{S}}$.

Claim 19. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},1} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},0} + \mathsf{negl}(\lambda)$.

Proof. This follows in a straightforward way from the IND-CPA security of the underlying scheme Σ. We construct a PPT adversary $\mathcal{B}$ attacking it as follows: given a public key pk, $\mathcal{B}$ simulates the experiment for $\mathcal{A}$ including choosing random passwords for each pair of users. When a ciphertext CT is expected to be sent to $\mathcal{A}$, $\mathcal{B}$ using $\omega_{\mathtt{S}}$ or ω (i.e., a random value independent of the anything that previously occurred) as its pair of messages to query its oracle; upon receiving a ciphertext, return it to $\mathcal{A}$. Note that $\mathcal{B}$ can compute correct session keys $\mathsf{SK}^i_{\mathtt{S}} = \mathsf{SK}^j_{\mathtt{C}}$, since $\Pi^i_{\mathtt{C}}$ is still simulated exactly as in the real protocol. At the end of the experiment, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ succeeds. The distinguishing advantage of $\mathcal{B}$ is exactly $|\mathsf{Adv}_1 - \mathsf{Adv}_0|$, and IND-CPA security of the underlying scheme Σ yields this Claim. □

Experiment $\mathbf{Expt}_2$: This experiment is same as $\mathbf{Expt}_1$ except that in response to an $\mathrm{Execute}(\mathtt{C}, i, \mathtt{S}, j)$ query, randomly choose $\varpi_{\mathtt{C}}|\tau_{\mathtt{C}}|\vartheta_{\mathtt{C}} \xleftarrow{\$} \{0,1\}^{3\lambda}$ and set $\varpi_{\mathtt{S}}|\tau_{\mathtt{S}}|\vartheta_{\mathtt{S}} = \varpi_{\mathtt{C}}|\tau_{\mathtt{C}}|\vartheta_{\mathtt{C}}$.

Claim 20. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},2} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},1} + \mathsf{negl}(\lambda)$.

Proof. This follows in a straightforward way from the smoothness of the underlying SPHF scheme Ω_Σ. In particular, as CT generated in $\mathbf{Expt}_1$ is an encryption of a random value ω rather than $\omega_{\mathtt{S}}$ (that equals to $\omega_{\mathtt{C}}$), thus $(\mathsf{CT}, \omega_{\mathtt{C}}) \notin \mathcal{L}$ holds. Therefore, the SPHF hash value computed is statistically close to uniform even conditioned on hp. □

Experiment $\mathbf{Expt}_3$: This experiment is same as $\mathbf{Expt}_2$ except that in response to an $\mathrm{Execute}(\mathtt{C}, i, \mathtt{S}, j)$ query, choose $\mathsf{SK}^i_{\mathtt{S}} \xleftarrow{\$} \{0,1\}^\lambda$ and set $\mathsf{SK}^j_{\mathtt{C}} = \mathsf{SK}^i_{\mathtt{S}}$.

Claim 21. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},3} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},2} + \mathsf{negl}(\lambda)$.

Proof. It is straightforward since $\vartheta_{\mathtt{C}} = \vartheta_{\mathtt{S}}$ are set randomly in $\mathbf{Expt}_2$. □

Experiment $\mathbf{Expt}_4$: This experiment is same as $\mathbf{Expt}_3$ except that in response to an $\mathrm{Execute}(\mathtt{C}, i, \mathtt{S}, j)$ query, set CT' as an encryption of a random $\mu \xleftarrow{\$} \{0,1\}^\lambda$ rather than $\mu_{\mathtt{C}}$.

Claim 22. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},4} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},3} + \mathsf{negl}(\lambda)$.

Proof. This follows in a straightforward way from the IND-CCA security of the underlying PKE scheme Σ'. Note that in $\mathbf{Expt}_3$, the ciphertext CT' is encrypted using truly randomness, thus we can build a PPT adversary $\mathcal{B}$ attacking this PKE scheme as follows: given a public key pk', $\mathcal{B}$ simulates the experiment for $\mathcal{A}$. When a ciphertext CT' is expected to be sent to $\mathcal{A}$, $\mathcal{B}$ using the real $\mu_{\mathtt{C}}$ or μ (i.e., a random value independent of the anything that previously occurred) as its pair of messages to query its oracle; upon receiving a ciphertext, return it to $\mathcal{A}$. Note that session keys $\mathsf{SK}^i_{\mathtt{S}} = \mathsf{SK}^j_{\mathtt{C}}$ are set random. At the end of the experiment, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ succeeds. The distinguishing advantage of $\mathcal{B}$ is exactly $|\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},4} - \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},3}|$. Therefore, the IND-CCA (in fact, IND-CPA security is enough here) security of the underlying PKE scheme Σ' yields this Claim. □

Experiment $\mathbf{Expt}_5$: This experiment is same as $\mathbf{Expt}_4$ except that now the secret keys $(\mathsf{sk}, \mathsf{sk}')$ are recorded when the public keys $(\mathsf{pk}, \mathsf{pk}')$ in the CRS are generated. Furthermore, the way to answer a $\mathrm{Send}_3(\mathtt{S}, j, (\mathsf{hp}, \mathsf{CT}'))$ query is changed as follows:

- If the message $(\mathsf{hp}, \mathsf{CT}')$ was output by a previous query $\mathrm{Send}_2(\mathtt{C}, i, \cdot)$, continue as before.
- Else if the message $(\mathsf{hp}, \mathsf{CT}')$ was not previously output, decrypt it (using sk') to derive μ, halt and declare "success" if $\mu = \mu_{\mathtt{S}}$.
- Otherwise, abort the session $\Pi^j_{\mathtt{S}}$.

Claim 23. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},5} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},4}$.

Proof. Assume the password of the server $\mathtt{S}$ associated with $\mathtt{C}$ is pw^*. Since $\mathcal{H}$ is collision resistant, no other password $\mathsf{pw} \neq \mathsf{pw}^*$ can make the following equality hold simultaneously

$$\omega_{\mathtt{S}} | \mu_{\mathtt{S}} | \theta_{\mathtt{S}} = \mathcal{H}(\mathtt{C} | \mathtt{S} | \mathsf{pek} | \mathsf{pek}' | \mathsf{k}_{\mathtt{S}} | \mathsf{pw}_{\mathtt{S},\mathtt{C}}).$$

Therefore, check whether $\mu = \mu_{\mathtt{S}}$ is equivalent to check whether the adversary guesses the correct password pw^*, except with negligible probability. The change still introduces a new way for $\mathcal{A}$ to succeed, however, such that $\mathcal{A}$'s advantage increases. □

Experiment $\mathbf{Expt}_6$: This experiment is same as $\mathbf{Expt}_5$ except that the ways to answer a $\mathrm{Send}_1(\mathtt{S}, j, (\mathtt{C}, \mathsf{pek}))$ and a $\mathrm{Send}_3(\mathtt{S}, j, (\mathsf{hp}, \mathsf{CT}'))$ query are changed. In particular, in response to a $\mathrm{Send}_1(\mathtt{S}, j, (\mathtt{C}, \mathsf{pek}))$ query, generate CT as an encryption of the random value $\omega \xleftarrow{\$} \{0,1\}^\lambda$ rather than $\omega_{\mathtt{S}}$. In response to a $\mathrm{Send}_3(\mathtt{S}, j, (\mathsf{hp}, \mathsf{CT}'))$ query:

1. If the message $(\mathsf{hp}, \mathsf{CT}')$ was output by a previous query $\mathrm{Send}_2(\mathtt{C}, i, (\mathtt{C}, \mathsf{pek}', \mathsf{CT}))$ and the same transcripts (i.e., $\mathtt{C}, \mathtt{S}, \mathsf{pek}, \mathsf{pek}', \mathsf{CT}$) are recorded in both $\Pi^i_{\mathtt{C}}$ and $\Pi^j_{\mathtt{S}}$, set $\varpi_{\mathtt{S}} | \tau_{\mathtt{S}} | \vartheta_{\mathtt{S}} = \varpi_{\mathtt{C}} | \tau_{\mathtt{C}} | \vartheta_{\mathtt{C}}$ and $\mathsf{SK}^j_{\mathtt{S}} = \mathsf{SK}^i_{\mathtt{C}}$ using the internal variables of the session $\Pi^i_{\mathtt{C}}$. Then send $\tau_{\mathtt{S}}$ to the adversary in the name of the server user $\mathtt{S}$.
2. Else if the message $(\mathsf{hp}, \mathsf{CT}')$ was not previously output, decrypt it (using sk') to derive μ, halt and declare "success" if $\mu = \mu_{\mathtt{S}}$;
3. Otherwise, abort the session $\Pi^j_{\mathtt{S}}$.

Claim 24. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},6} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},5} + \mathsf{negl}(\lambda)$.

Proof. Consider an intermediate experiment $\mathbf{Expt}_{5'}$, where the Send_3 oracle is modified as described above, but Send_1 still computes CT exactly as in $\mathbf{Expt}_5$. This is simply a syntactic rewriting of $\mathbf{Expt}_5$, and so the adversary's advantage remains unchanged.

Next we show that $\mathcal{A}$'s advantage can change by only a negligible amount in moving from $\mathbf{Expt}_{5'}$ to $\mathbf{Expt}_6$. It comes in a straightforward way from the IND-CPA security of the underlying PKE scheme Σ. We construct a PPT adversary $\mathcal{B}$ attacking it as follows: given a public key pk, $\mathcal{B}$ simulates the experiment for $\mathcal{A}$ including generating pk' and recording the corresponding secret key sk' by itself. When a ciphertext CT is expected to be sent to $\mathcal{A}$, $\mathcal{B}$ using the real $\omega_{\mathtt{S}}$ or ω (i.e., a random value independent of the anything that

previously occurred) as its pair of messages to query its oracle; upon receiving a ciphertext, return it to $\mathcal{A}$. Note that $\mathcal{B}$ can still respond to $\mathrm{Send}_3(\mathtt{S}, j, (\mathsf{hp}, \mathsf{CT}'))$ queries, since knowledge of the randomness used to generate CT' is no longer used (in either $\mathbf{Expt}_{5'}$ or $\mathbf{Expt}_6$). At the end of the experiment, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ succeeds. The distinguishing advantage of $\mathcal{B}$ is exactly $|\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},6} - \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},5'}|$. Therefore, IND-CPA security of the underlying PKE scheme Σ yields this Claim. □

Experiment $\mathbf{Expt}_7$: This experiment is same as $\mathbf{Expt}_6$ except that the way to answer a $\mathrm{Send}_2(\mathtt{C}, i, (\mathtt{S}, \mathsf{pek}', \mathsf{CT}))$ query is changed as following:

- If $(\mathtt{S}, \mathsf{pek}', \mathsf{CT})$ was not output by a previous query $\mathrm{Send}_1(\mathtt{S}, j, (\mathtt{C}, \mathsf{pek}))$, decrypt it (using sk) to derive ω, halt and declare "success" if $\omega = \omega_{\mathtt{C}}$;
- Otherwise, randomly set $\varpi_{\mathtt{C}}|\tau_{\mathtt{C}}|\vartheta_{\mathtt{C}}$ and continue as before.

Claim 25. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},7} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},6} + \mathsf{negl}(\lambda)$.

Proof. Assume the password of the client $\mathtt{C}$ is pw^*. Since $\mathcal{H}$ is collision resistant, no other password $\mathsf{pw} \neq \mathsf{pw}^*$ can make the following equality holds:

$$\omega_{\mathtt{C}}|\mu_{\mathtt{C}}|\theta_{\mathtt{C}} = \mathcal{H}(\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}_{\mathtt{C}}|\mathsf{pw}_{\mathtt{C}})$$

Therefore, check whether $\omega = \omega_{\mathtt{C}}$ is equivalent to check whether the adversary guesses the correct password pw^*, except with negligible probability. Case 1 still introduces a new way for $\mathcal{A}$ to succeed, however, such that $\mathcal{A}$'s advantage increases. Case 2 introduces negligible advantage for $\mathcal{A}$ and this follows from the smoothness of the underlying SPHF scheme Ω_Σ when $(\mathsf{CT}, \omega_{\mathtt{C}}) \notin \mathcal{L}$ holds. Therefore, the output is statistically close to uniform even conditioned on hp. □

Experiment $\mathbf{Expt}_8$: The way to answer a $\mathrm{Send}_2(\mathtt{C}, i, (\mathtt{S}, \mathsf{pek}', \mathsf{CT}))$ query is changed again. In particular, for Case 2 in $\mathbf{Expt}_7$, CT' is generated as an encryption of a random value $\mu \xleftarrow{\$} \{0,1\}^\lambda$ rather than $\mu_{\mathtt{C}}$.

Claim 26. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},8} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},7} + \mathsf{negl}(\lambda)$.

Proof. The change introduces negligible advantage for $\mathcal{A}$ and this can be shown by the IND-CCA security of the underlying PKE scheme Σ'. There are two observations: (1) the randomness used to generate CT' is truly random now; (2) we may need to decrypt CT' in Send_3 queries. We construct a PPT adversary $\mathcal{B}$ attacking this PKE scheme as follows: given public key pk', $\mathcal{B}$ simulates the experiment for $\mathcal{A}$ including generating pk and recording the corresponding secret key sk by itself. When a ciphertext CT' is expected to be sent to $\mathcal{A}$, $\mathcal{B}$ using the real $\mu_{\mathtt{C}}$ or μ (i.e., a random value independent of the anything that previously occurred) as its pair of messages to query its oracle; upon receiving a ciphertext, return it to $\mathcal{A}$. Note that $\mathcal{B}$ can still respond to $\mathrm{Send}_3(\mathtt{S}, j, (\mathsf{hp}, \mathsf{CT}'))$ queries using its decryption oracle. At the end of the experiment, $\mathcal{B}$ outputs 1 if and only if $\mathcal{A}$ succeeds. The distinguishing advantage of $\mathcal{B}$ is exactly $|\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},8} - \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},7}|$. Therefore, the IND-CCA security of the underlying PKE scheme Σ' yields this Claim. □

Bounding the advantage in $\mathbf{Expt}_8$. $\mathcal{A}$ can succeed in following ways:

1. Successfully launch off-line guessing attack to some honest party via the transcripts it has seen before making any $\mathrm{Corrupt}$ query.
2. Query $\mathrm{Send}_2(\mathtt{C}, i, (\mathsf{pek}', \mathsf{CT}))$ with $\textsc{Dec}(\mathsf{sk}, \mathsf{CT}) = \omega_{\mathtt{C}}$.
3. Query $\mathrm{Send}_3(\mathtt{S}, j, (\mathsf{hp}, \mathsf{CT}'))$ with $\textsc{Dec}'(\mathsf{sk}', \mathsf{CT}') = \mu_{\mathtt{S}}$.
4. Query $\mathrm{Send}_4(\mathtt{C}, i, \tau)$ with $\tau_{\mathtt{S}} = \tau_{\mathtt{C}}$ but $\tau_{\mathtt{S}}$ was not output by $\Pi^i_{\mathtt{C}}$'s paired instance.
5. Successfully guess the bit of the Test oracle.

In the view of the adversary, all transcripts except pek and pek' are simulated and independent of all passwords. As for pek and pek', they perfectly hide the passwords according perfect hiding property of the underlying homomorphic NIKE scheme. Therefore, the adversary is *information-theoretically* unable to make Case 1 occur. Case 2 or Case 3 occurs only with probability at most $n_{\mathrm{Send}}/|\mathcal{PW}|$, since the view of the adversary is independent of all correct passwords until one of the two cases occurs. Case 4 only occurs with negligible probability, since $\tau_{\mathtt{C}}$ is a uniform λ-bit string that is independent of the adversary's view if $\tau_{\mathtt{C}}$ was not output by any instance paired with $\Pi^i_{\mathtt{C}}$. Conditioned on Case 1 - Case 3 not occurring, the adversary can only success in Case 5. But in that case, all session keys are chosen uniformly and independently at

random with non-paired instances, so the success probability is exactly $1/2$. Thus, it is straightforward that $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},8} \leq n_{\mathrm{Send}}/|\mathcal{PW}|$. Taken all above together, we have that $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},\pi} \leq n_{\mathrm{Send}}/|\mathcal{PW}| + \mathsf{negl}(\lambda)$.

Proof of Case-(2):

We model $\mathcal{H}$ as RO built on the fly, i.e., each new query is answered with a fresh random output, and each query that is not new is answered consistently with the previous quires. In particular, one empty list $L_{\mathcal{H}}$ is initialized first. Then, for each $\mathcal{H}$ query on $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}$ is answered as follows:

- If there exists $(\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}, \omega|\mu|\theta) \in L_{\mathcal{H}}$, return $\omega|\mu|\theta$.
- Otherwise, return a random value $\omega|\mu|\theta \xleftarrow{\$} \{0,1\}^{3\lambda}$.
  Append $(\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}, \omega|\mu|\theta)$ to $L_{\mathcal{H}}$.

For each pair of users $(\mathtt{C}, \mathtt{S})$, fix their shared password as an independently chosen random value $\mathsf{pw}^*_{\mathtt{S},\mathtt{C}}$. If $\mathcal{A}$ has never queried $\mathcal{H}$ on the correct input associated to the test session, the session key is perfectly indistinguishable from random. For a client instance $\Pi^i_{\mathtt{C}}$, it is called *paired with* a server instance $\Pi^j_{\mathtt{S}}$ if there is a $\mathrm{Send}_0(\mathtt{C}, i, \mathtt{S})$ query that returns $(\mathtt{C}, \mathsf{pek})$, a $\mathrm{Send}_1(\mathtt{S}, j, (\mathtt{C}, \mathsf{pek}))$ query that returns $(\mathtt{S}, \mathsf{pek}', \mathsf{CT})$, and a $\mathrm{Send}_2(\mathtt{C}, i, (\mathsf{pek}', *))$ query. For a server instance $\Pi^j_{\mathtt{S}}$, it is called *paired with* a client instance $\Pi^i_{\mathtt{C}}$ if there is a $\mathrm{Send}_0(\mathtt{C}, i, \mathtt{S})$ that returns $(\mathtt{C}, \mathsf{pek})$, and a $\mathrm{Send}_1(\mathtt{S}, j, (\mathtt{C}, \mathsf{pek}))$ query.

In the next, we define several events corresponding to $\mathcal{A}$ successfully launching password-guessing attacks via Send or $\mathrm{Execute}$ queries. In each case, an associated value for the event is defined, which is determined by the protocol before that event occurs.

– $\mathsf{testpwC}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw})$: ($\mathcal{A}$ is attacking to a client instance.) For some $(\mathsf{pek}, \mathsf{pek}', \mathsf{CT}, \mathsf{pw})$, $\mathcal{A}$ makes a $\mathcal{H}$ query on $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}$, a $\mathrm{Send}_0(\mathtt{C}, i, \mathtt{S})$ query that returns $(\mathtt{C}, \mathsf{pek})$, a $\mathrm{Send}_2(\mathtt{C}, i, (\mathtt{S}, \mathsf{pek}', \mathsf{CT}))$ query, where the latest query is either $\mathrm{Send}_2(\cdot)$ or $\mathcal{H}$,

$$\mathsf{k} = \mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}), \hat{f}_1(\mathsf{pw}, \mathsf{pek}')).$$

The associated value is the output of the $\mathcal{H}$ query, denoted as $\omega^i_{\mathtt{C}}|\mu^i_{\mathtt{C}}|\theta^i_{\mathtt{C}}$.

– $\mathsf{testpwC}^*(\mathtt{C}, i, \mathtt{S}, \mathsf{pw})$: For some $(\mathsf{pek}, \mathsf{pek}', \mathsf{CT}, \mathsf{pw})$, a $\mathrm{Send}_2(\mathtt{C}, i, (\mathtt{C}, \mathsf{pek}', \mathsf{CT}))$ query causes a $\mathsf{testpwC}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw})$ event to occur, with associated value $\omega^i_{\mathtt{C}}|\mu^i_{\mathtt{C}}|\theta^i_{\mathtt{C}}$.

– $\mathsf{testpwC}^{\#}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw})$: $\mathcal{A}$ makes a $\mathrm{Send}_4(\mathtt{C}, i, \tau_{\mathtt{S}})$ query, and previously made a $\mathsf{testpwC}^*(\mathtt{C}, i, \mathtt{S}, \mathsf{pw})$ event occurs with associated value $\omega^i_{\mathtt{C}}|\mu^i_{\mathtt{C}}|\theta^i_{\mathtt{C}}$, and $\tau_{\mathtt{S}}$ equals to $\tau_{\mathtt{C}}$ computed from $\omega^i_{\mathtt{C}}$ as the protocol description.

– $\mathsf{testpwS}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw})$: ($\mathcal{A}$ is attacking to a server instance.) For some $(\mathsf{pek}, \mathsf{pek}', \mathsf{CT}, \mathsf{pw})$, $\mathcal{A}$ makes a $\mathcal{H}$ query on $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}$, and previously made a $\mathrm{Send}_1(\mathtt{S}, j, (\mathtt{C}, \mathsf{pek}))$ query that returns $(\mathtt{S}, \mathsf{pek}', \mathsf{CT})$, where

$$\mathsf{k} = \mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}), \hat{f}_1(\mathsf{pw}, \mathsf{pek}')).$$

The associated value is the output of the $\mathcal{H}$ query, denoted as $\omega^j_{\mathtt{S}}|\mu^j_{\mathtt{S}}|\theta^j_{\mathtt{S}}$.

– $\mathsf{testpwS}^{\#}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw})$: $\mathcal{A}$ makes a $\mathrm{Send}_3(\mathtt{S}, j, (\mathsf{hp}, \mathsf{CT}'))$ query, and previously made a $\mathsf{testpwS}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw})$ event occurs with associated value $\omega^j_{\mathtt{S}}|\mu^j_{\mathtt{S}}|\theta^j_{\mathtt{S}}$, and $\mathsf{CT}' = \mathrm{ENC}'(\mathsf{pk}', \mathsf{label}, \mu^j_{\mathtt{S}}; \varpi_{\mathtt{S}})$ where $\varpi_{\mathtt{S}}$ is computed according to the protocol description.

– $\mathsf{testpwExec}(\mathtt{C}, i, \mathtt{S}, j, \mathsf{pw})$: ($\mathcal{A}$ is passively attacking via $\mathrm{Execute}$ oracle.) For some tuple $(\mathsf{pek}, \mathsf{pek}', \mathsf{pw})$, $\mathcal{A}$ makes a $\mathcal{H}$ query on $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}$, and previously made an $\mathrm{Execute}(\mathtt{C}, i, \mathtt{S}, j)$ query that generates $(\mathsf{pek}, \mathsf{pek}')$, where

$$\mathsf{k} = \mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}), \hat{f}_1(\mathsf{pw}, \mathsf{pek}')).$$

The associated value is the output of the $\mathcal{H}$ query, denoted as $\omega^i_{\mathtt{S}}|\mu^i_{\mathtt{S}}|\theta^i_{\mathtt{S}} = \omega^j_{\mathtt{C}}|\mu^j_{\mathtt{C}}|\theta^j_{\mathtt{C}}$.

– $\mathsf{testpw}(\mathtt{C}, i, \mathtt{S}, j, \mathsf{pw})$: ($\mathcal{A}$ is passively attacking via Send oracle.)
Both a $\mathsf{testpwC}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw})$ and a $\mathsf{testpwS}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw})$ event occur, where $\Pi^i_{\mathtt{C}}$ is paired with $\Pi^j_{\mathtt{S}}$ and $\Pi^j_{\mathtt{S}}$ is paired with $\Pi^i_{\mathtt{C}}$ after its corresponding $\mathrm{Send}_1(\cdot)$ query.

– pairedpwGuess: A $\mathsf{testpw}(\mathtt{C}, i, \mathtt{S}, j, \mathsf{pw})$ event occurs for some $(\mathtt{C}, i, \mathtt{S}, j)$.

– doublepwC: Both a $\mathsf{testpwC}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_0)$ event and a $\mathsf{testpwC}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_1)$ event occur, for some ($\mathtt{C}$, $\mathtt{S}$, i, pw_0, pw_1) and $\mathsf{pw}_0 \neq \mathsf{pw}_1$.

– doublepwS: Both a $\mathsf{testpwS}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw}_0)$ and a $\mathsf{testpwS}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw}_1)$ occur, for some ($\mathtt{S}$, $\mathtt{C}$, j, pw_0, pw_1) and $\mathsf{pw}_0 \neq \mathsf{pw}_1$.

- correctpwSend: Before any Corrupt(·) query, a $\mathsf{testpwC}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw})$ event occurs for some $(\mathtt{S}, \mathtt{C}, i)$; *or* a $\mathsf{testpwS}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw})$ event occurs for some $(\mathtt{C}, \mathtt{S}, j)$.
- correctpwExec: A $\mathsf{testpwExec}(\mathtt{C}, i, \mathtt{S}, j, \mathsf{pw})$ event occurs for some $(\mathtt{C}, i, \mathtt{S}, j)$.

We denote the number of $\mathrm{Execute}$, Send and RO queries made by $\mathcal{A}$ as n_{Execute}, n_{Send} and n_{RO}, respectively. We will use a series of experiments $\mathbf{Expt}_0, \ldots, \mathbf{Expt}_8$ to bound the advantage of $\mathcal{A}$. We denote the advantage of $\mathcal{A}$ in $\mathbf{Expt}_{\mathrm{i}}$ as:

$$\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},\mathrm{i}}(\lambda) := 2|\Pr[\mathcal{A} \text{ succeeds in } \mathbf{Expt}_{\mathrm{i}}] - 1|.$$

In particular, we let $\mathbf{Expt}_0$ denote the original experiment, such that $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},0} = \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},\pi}$ holds.

Experiment $\mathbf{Expt}_1$: This experiment is same as $\mathbf{Expt}_0$ except that it halts whenever honest parties randomly choose pek or pek' seen previously.

Claim 27. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},1} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},0} + \mathsf{negl}(\lambda)$.

Proof. Let E_1 be the event that a pek value generated in a Send_0 query or $\mathrm{Execute}$ query is equal to a pek value generated in previous Send_0 query or $\mathrm{Execute}$ query, or a pek value sent as input in a previous Send_1 query, or a pek value in a previous $\mathcal{H}$ query. Let E_2 be the event that a pek' value generated in a Send_1 query or $\mathrm{Execute}$ query is equal to a pek' value generated in a previous Send_1 query or $\mathrm{Execute}$ query, or a pek' value sent as input in a previous Send_2 query, or a pek' value in a previous $\mathcal{H}$ query. Let $E = E_1 \vee E_2$. If E does not occur, $\mathbf{Expt}_1$ is identical to $\mathbf{Expt}_0$. There are $n_{\mathrm{Send}} + n_{\mathrm{Execute}}$ such values are required to be generated, thus $\Pr[E \text{ occurs }] \leq (n_{\mathrm{Send}} + n_{\mathrm{Execute}})(n_{\mathrm{Send}} + n_{\mathrm{Execute}} + n_{\mathrm{RO}})/|\mathcal{EK}|$. □

Experiment $\mathbf{Expt}_2$: This experiment is same as $\mathbf{Expt}_1$ except that the ways to answer $\mathrm{Execute}$ and Send queries are changed without making any RO (i.e., $\mathcal{H}$) queries. Subsequent RO queries made by the adversary are backpatched to be consistent with these responses. In particular:

- In an $\mathrm{Execute}(\mathtt{C}, i, \mathtt{S}, j)$ query: set $(\mathsf{pek}, \mathsf{pdk}^i_{\mathtt{C}}) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$, $(\mathsf{pek}', \mathsf{pdk}^j_{\mathtt{S}}) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $\omega^i_{\mathtt{C}}|\mu^i_{\mathtt{C}}|\theta^i_{\mathtt{C}} = \omega^j_{\mathtt{S}}|\mu^j_{\mathtt{S}}|\theta^j_{\mathtt{S}} \xleftarrow{\$} \{0,1\}^{3\lambda}$.
- In a $\mathrm{Send}_0(\mathtt{C}, i, \mathtt{S})$ query: set $(\mathsf{pek}, \mathsf{pdk}^i_{\mathtt{C}}) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$.
- In a $\mathrm{Send}_1(\mathtt{S}, j, (\mathtt{C}, \mathsf{pek}))$ query: set $(\mathsf{pek}', \mathsf{pdk}^j_{\mathtt{S}}) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$ and $\omega^j_{\mathtt{S}}|\mu^j_{\mathtt{S}}|\theta^j_{\mathtt{S}} \xleftarrow{\$} \{0,1\}^{3\lambda}$.
- In a $\mathrm{Send}_2(\mathtt{C}, i, (\mathtt{S}, \mathsf{pek}', \mathsf{CT}))$ query:
  (1) If it causes a $\mathsf{testpwC}^*(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}^*_{\mathtt{S},\mathtt{C}})$ event to occur, set $\omega^i_{\mathtt{C}}|\mu^i_{\mathtt{C}}|\theta^i_{\mathtt{C}}$ as the associated value of that event.
  (2) Else if $\Pi^j_{\mathtt{S}}$ is paired with a session $\Pi^i_{\mathtt{C}}$, set $\omega^i_{\mathtt{C}}|\mu^i_{\mathtt{C}}|\theta^i_{\mathtt{C}} = \omega^j_{\mathtt{S}}|\mu^j_{\mathtt{S}}|\theta^j_{\mathtt{S}}$.
  (3) Otherwise, set $\omega^i_{\mathtt{C}}|\mu^i_{\mathtt{C}}|\theta^i_{\mathtt{C}} \xleftarrow{\$} \{0,1\}^{3\lambda}$.
- In a $\mathrm{Send}_3(\mathtt{S}, j, (\mathsf{hp}, \mathsf{CT}'))$ query: if it causes a $\mathsf{testpwS}^{\#}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw}^*_{\mathtt{S},\mathtt{C}})$ event to occur or $\Pi^j_{\mathtt{S}}$ is paired with a session $\Pi^i_{\mathtt{C}}$, continue; otherwise, abort.
- In a $\mathrm{Send}_4(\mathtt{C}, i, \tau_{\mathtt{S}})$ query: if it causes a $\mathsf{testpwC}^{\#}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}^*_{\mathtt{S},\mathtt{C}})$ event to occur or $\Pi^i_{\mathtt{C}}$ is paired with a session $\Pi^j_{\mathtt{S}}$, continue; otherwise, abort.
- For a $\mathcal{H}$ query on $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}$:
  (1) If it causes a $\mathsf{testpwC}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}^*_{\mathtt{S},\mathtt{C}})$ event, a $\mathsf{testpwS}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw}^*_{\mathtt{S},\mathtt{C}})$ event or a $\mathsf{testpwExec}(\mathtt{C}, i, \mathtt{S}, j, \mathsf{pw}^*_{\mathtt{S},\mathtt{C}})$ event to occur, output the associated value of that event.
  (2) Otherwise, output random values.

Claim 28. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},2} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},1} + \mathsf{negl}(\lambda)$.

Proof. Due to the perfect binding property of the underlying homomorphic NIKE scheme, the modification on the generations of pek and pek' does not introduce any difference.

Due to the equivocability of the underlying homomorphic NIKE scheme, the above modification does not influence detecting the occurrence of the events $\mathsf{testpwC}$, $\mathsf{testpwS}$ and $\mathsf{testpwExec}$. Because with the algorithm $\textsc{Ext}$, we can easily compute the secret key of $\hat{f}_\sigma(\mathsf{pw}, \mathsf{pek})$ for any password pw if knowing the corresponding secret key of pek.

From $\mathbf{Expt}_1$ we can see that $\mathcal{H}$ queries are new, therefore the values $\omega_{\mathsf{S}}^j|\mu_{\mathsf{S}}^j|\theta_{\mathsf{S}}^j$ created by a server instance Π_{C}^i in a $\mathrm{Send}_1(\mathtt{S}, j, (\mathtt{C}, \mathsf{pek}))$ query are uniformly chosen from $\{0,1\}^{3\lambda}$, independently of anything that previously occurred. Then in a $\mathrm{Send}_3(\mathtt{S}, j, (\mathsf{hp}, \mathsf{CT}'))$ query, if it causes a $\mathsf{testpwS}^{\#}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event to occur or Π_{S}^j is paired, the instance continues (in particular, terminates after sending the reply message); otherwise, the instance terminates or aborts. The total probability of any instance terminating in the second case is at most $\max(n_{\mathrm{Send}}/2^\lambda, n_{\mathrm{Send}}/|\mathcal{CT}'|)$, where $\mathcal{CT}'$ denotes the ciphertext space of the underlying PKE scheme Σ'.

For a client instance Π_{C}^i, either:

- A $\mathsf{testpwC}^*(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event occurs, the values $\omega_{\mathsf{C}}^i|\mu_{\mathsf{C}}^i|\theta_{\mathsf{C}}^i$ are set as the associated value of that event, which are guaranteed by the original assumption.
- No $\mathsf{testpwC}^*(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event occurs, but a $\mathsf{testpwC}^{\#}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event occurs, Π_{C}^i will terminate and the values $\omega_{\mathsf{C}}^i|\mu_{\mathsf{C}}^i|\theta_{\mathsf{C}}^i$ have been set previously.
- No $\mathsf{testpwC}^*(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ and $\mathsf{testpwC}^{\#}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ events occur, but exactly one instance Π_{S}^j is paired with Π_{C}^i, in which case $\omega_{\mathsf{C}}^i|\mu_{\mathsf{C}}^i|\theta_{\mathsf{C}}^i = \omega_{\mathsf{S}}^j|\mu_{\mathsf{S}}^j|\theta_{\mathsf{S}}^j$.
- No $\mathsf{testpwC}^*(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event occurs, no $\mathsf{testpwC}^{\#}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event occurs, and no instance is paired with Π_{C}^i, then either the instance terminates or aborts. In this case, the probability of any instance terminating in this case is at most $\max(n_{\mathrm{Send}}/2^\lambda, n_{\mathrm{Send}}/|\mathcal{V}|)$, where $\mathcal{V}$ denotes the hash values' space of the underlying SPHF scheme Ω_Σ.

For any $\mathcal{H}$ query on $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}$:

1. It causes a $\mathsf{testpwC}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$, a $\mathsf{testpwS}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ or a $\mathsf{testpwExec}(\mathtt{C}, i, \mathtt{S}, j, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event to occur, in which case the output is the associated value of the corresponding event;
2. The output is randomly chosen, independent of anything that previously occurred, since this is a new query.

If an unpaired server instance Π_{S}^j never terminates without occurring a $\mathsf{testpwS}^{\#}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event, an unpaired client instance Π_{S}^j never terminates without occurring a $\mathsf{testpwC}^*(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event or a $\mathsf{testpwC}^{\#}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}_{\mathtt{S},\mathtt{C}}^*)$ event, then we have that $\mathbf{Expt}_2$ is consistent with $\mathbf{Expt}_1$.

Summarize the above, this claim holds □

Experiment $\mathbf{Expt}_3$: This experiment is same as $\mathbf{Expt}_2$ except that RO queries are answered as before but checking consistency against $\mathrm{Execute}$ queries is dropped.

Claim 29. $\mathsf{Adv}_{\mathcal{A},4}^{\mathrm{bpr}} \leq \mathsf{Adv}_{\mathcal{A},3}^{\mathrm{bpr}} + \mathsf{negl}(\lambda)$.

Proof. If $\mathcal{A}$ makes a correctpwExec event occur, it will notice the change. Next, we will construct an algorithm $\mathcal{B}$ that attempts to break the Type-I(i) intractability of the underlying homomorphic NIKE scheme Π by running $\mathcal{A}$ as a subroutine.

Given a challenge tuple $(\overline{\mathsf{pek}}, \overline{\mathsf{pek}}')$, $\mathcal{B}$ changes the way to answer $\mathrm{Execute}(\cdot)$ queries as follows:

- In an $\mathrm{Execute}(\mathtt{C}, i, \mathtt{S}, j)$ query: set $\mathsf{pek} = \overline{\mathsf{pek}} \cdot \mathsf{ek}_{\mathsf{C}}^i$, where $(\mathsf{ek}_{\mathsf{C}}^i, \mathsf{dk}_{\mathsf{C}}^i) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$; set $\mathsf{pek}' = \overline{\mathsf{pek}}' \cdot \mathsf{ek}_{\mathsf{S}}^j$, where $(\mathsf{ek}_{\mathsf{S}}^j, \mathsf{dk}_{\mathsf{S}}^j) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$.
- When $\mathcal{A}$ stops, for every $\mathcal{H}$ query on $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}$, where pek and pek' were generated in an $\mathrm{Execute}(\mathtt{C}, i, \mathtt{S}, j)$ query, do as follows:
  - Compute $\bar{\mathsf{k}}_1 = \textsc{Key}(\mathsf{dk}_{\mathsf{C}}^i, \mathsf{ek}_{\mathsf{S}}^j \cdot \hat{f}_0(\mathsf{pw}, \mathsf{pek}'))$ and $\bar{\mathsf{k}}_2 = \textsc{Key}(\mathsf{dk}_{\mathsf{S}}^j, \hat{f}_0(\mathsf{pw}, \mathsf{pek}))$;
  - Compute $\mathsf{k}^* = \mathsf{k}/(\bar{\mathsf{k}}_1 \cdot \bar{\mathsf{k}}_2)$.
  - Add $(\mathsf{pw}, \mathsf{k}^*)$ into the possible solution list.

Given $(\mathsf{pek}, \mathsf{pek}')$, for a password-guess pw, there exists an unique correct value:

$$\mathsf{k} \quad = \mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}) \cdot \mathsf{ek}_{\mathsf{C}}^i, \hat{f}_1(\mathsf{pw}, \mathsf{pek}') \cdot \mathsf{ek}_{\mathsf{S}}^j)$$

We can adjust the equation so that only $\mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}), \hat{f}_1(\mathsf{pw}, \mathsf{pek}'))$ on the right. If a correctpwExec event occurs, $\mathcal{B}$ adds the correct answer. Assume the Type-I(i) intractability of the underlying homomorphic NIKE scheme holds, correctpwExec only occurs with negligible probability. □

Experiment $\mathbf{Expt}_4$: This experiment is same as $\mathbf{Expt}_3$ except when the adversary guesses the correct password (i.e., $\mathcal{A}$ makes a correctpwSend event occur), halts and declare "success". Note that this involves the following changes:

- In a Send_2 query to a client instance $\Pi^j_{\mathtt{S}}$, if a $\mathsf{testpwC}^*(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}^*_{\mathtt{S},\mathtt{C}})$ event occurs and no Corrupt query has been made, halt and declare "success".
- In a $\mathcal{H}$ query, if a $\mathsf{testpwC}(\mathtt{C}, i, \mathtt{S}, \mathsf{pw}^*_{\mathtt{S},\mathtt{C}})$ or a $\mathsf{testpwS}(\mathtt{S}, j, \mathtt{C}, \mathsf{pw}^*_{\mathtt{S},\mathtt{C}})$ event occurs and no $\mathrm{Corrupt}$ query has been made, halt and declare "success".

Claim 30. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},4} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},3}$.

Proof. These changes introduce new ways for $\mathcal{A}$ to succeed, such that the $\mathcal{A}$'s advantage increases. □

Experiment $\mathbf{Expt}_5$: This experiment is same as $\mathbf{Expt}_4$ except that if $\mathcal{A}$ makes a pairedpwGuess event occur, halt and declare "loss". We suppose that when a query is made, the test for correctpwSend occurs after the test for pairedpwGuess.

Claim 31. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},5} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},4} + \mathsf{negl}(\lambda)$.

Proof. If $\mathcal{A}$ makes a pairedpwGuess event occur, it will notice the change. Next, we will construct an algorithm $\mathcal{B}$ that attempts to break the Type-I(i) Intractability of the underlying homomorphic NIKE scheme by running $\mathcal{A}$ as a subroutine. Given a challenge $(\overline{\mathsf{pek}}, \overline{\mathsf{pek}}')$, $\mathcal{B}$ chooses a random $d \xleftarrow{\$} [n_{se}]$, and changes the ways to answer $\mathrm{Send}(\cdot)$ queries as follows:

- In the d-th Send_0 query, say to a client instance $\Pi^i_{\mathtt{C}}$ with $\mathtt{C}$ as input, set $\mathsf{pek} = \overline{\mathsf{pek}}$.
- In a Send_1 query to a server instance $\Pi^j_{\mathtt{S}}$, where there was a $\mathrm{Send}_0(\mathtt{C}, i, \mathtt{S})$ query that generates pek, set $\mathsf{pek}' = \overline{\mathsf{pek}}' \cdot \mathsf{ek}^j_{\mathtt{S}}$ with $(\mathsf{ek}^j_{\mathtt{S}}, \mathsf{dk}^j_{\mathtt{S}}) \xleftarrow{\$} \mathrm{GEN}(\mathsf{pp})$.
- In a Send_2 query to $\Pi^i_{\mathtt{C}}$, if $\Pi^i_{\mathtt{C}}$ is unpaired, $\mathcal{B}$ outputs 0 and halts.
- In a Send_3 query to a server instance $\Pi^j_{\mathtt{S}}$, if $\Pi^j_{\mathtt{S}}$ was paired with $\Pi^i_{\mathtt{C}}$ after its Send_1 query, but is not now paired with $\Pi^i_{\mathtt{C}}$, no test for correctpwSend is made, and $\Pi^j_{\mathtt{S}}$ aborts.
- In a Send_4 query to $\Pi^i_{\mathtt{C}}$, if $\Pi^i_{\mathtt{C}}$ was paired with a server instance $\Pi^j_{\mathtt{S}}$ after its Send_2 query, but is not now paired with $\Pi^j_{\mathtt{S}}$, no test for correctpwSend is made, and $\Pi^i_{\mathtt{C}}$ aborts.
- When $\mathcal{A}$ stops, for every $\mathcal{H}$ query on $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}|\mathsf{pw}$, where com and pek' were respectively generated by the instances $\Pi^i_{\mathtt{C}}$ and $\Pi^j_{\mathtt{S}}$, and $\Pi^j_{\mathtt{S}}$ is paired with $\Pi^i_{\mathtt{C}}$ after the corresponding $\mathrm{Send}_1(\cdot)$ query, do as follows:
  - Compute $\bar{\mathsf{k}} = \mathrm{KEY}(\mathsf{ek}^j_{\mathtt{S}}, \mathsf{dk}^j_{\mathtt{S}}, \hat{f}_0(\mathsf{pw}, \mathsf{pek}))$;
  - Compute $\mathsf{k}^* = \mathsf{k} \cdot \bar{\mathsf{k}}$.
  - Add $(\mathsf{pw}, \mathsf{k}^*)$ into the possible solution list.

  Given $(\mathsf{pek}, \mathsf{pek}')$, for a password-guess pw, there exists an unique correct

$$\mathsf{k} = \mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}), \hat{f}_1(\mathsf{pw}, \mathsf{pek}') \cdot \mathsf{ek}^j_{\mathtt{S}}).$$

We can adjust this equation so that only $\mathbf{Key}(\hat{f}_0(\mathsf{pw}, \mathsf{pek}), \hat{f}_1(\mathsf{pw}, \mathsf{pek}'))$ remains on the right. If a pairedpwGuess event occurs, with probability $1/n_{\mathrm{Send}}$, $\mathcal{B}$ adds the correct answer. Assume the Type-I(i) intractability of the underlying homomorphic NIKE with associated functions holds, pairedpwGuess only occurs with negligible probability. □

Experiment $\mathbf{Expt}_6$: This experiment is same as $\mathbf{Expt}_5$ except that if $\mathcal{A}$ makes a doublepwS event occur, halt and declare "loss". We suppose that when a query is made, the test for correctpwSend or pairedpwGuess occurs after the test for doublepwS.

Claim 32. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},6} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},5} + \mathsf{negl}(\lambda)$.

Proof. If $\mathcal{A}$ makes a doublepwS event occur, it will notice the change. Next, we will construct an algorithm $\mathcal{B}$ that attempts to break the Type-I(ii) intractability of the underlying homomorphic NIKE scheme by running $\mathcal{A}$ as a subroutine. Given a challenge $\overline{\mathsf{pek}}$, $\mathcal{B}$ changes the way to answer $\mathrm{Send}(\cdot)$ queries as follows:

- In a $\mathrm{Send}_0(\mathtt{C}, i, \mathtt{S})$ query, set $\mathsf{pek} = \overline{\mathsf{pek}} \cdot \mathsf{ek}^i_{\mathtt{C}}$ with $(\mathsf{ek}^i_{\mathtt{C}}, \mathsf{dk}^i_{\mathtt{C}}) \xleftarrow{\$} \mathrm{GEN}(\mathsf{pp})$.

- Tests for correctpwSend (from $\mathbf{Expt}_4$) and pairedpwGuess (from $\mathbf{Expt}_5$) are not made.
- When $\mathcal{A}$ finishes, for every pair of $\mathcal{H}$ queries on values $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}_0|\mathsf{pw}_0$ and $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}_1|\mathsf{pw}_1$ with $\mathsf{pw}_0 \neq \mathsf{pw}_1$, there was a $\mathrm{Send}_0(\mathtt{C}, i, \mathtt{S})$ query generates pek and a $\mathrm{Send}_2(\mathtt{C}, i, (\mathsf{pek}', \cdot))$ query with pek' as input, if $\mathsf{pw}_0 \neq \mathsf{pw}_1$, do as follows:
  - Compute $\bar{\mathsf{k}}_0 = \textsc{Key}(\mathsf{dk}_{\mathtt{C}}^i, \hat{f}_1(\mathsf{pw}_0, \mathsf{pek}'))$ and $\bar{\mathsf{k}}_1 = \textsc{Key}(\mathsf{dk}_{\mathtt{C}}^i, \hat{f}_1(\mathsf{pw}_1, \mathsf{pek}'))$.
  - Compute $\mathsf{k}_0^* = \mathsf{k}_0/\bar{\mathsf{k}}_0$ and $\mathsf{k}_1^* = \mathsf{k}_1/\bar{\mathsf{k}}_1$.
  - Add $(\mathsf{pek}', (\mathsf{pw}_0, \mathsf{k}_0^*), (\mathsf{pw}_1, \mathsf{k}_1^*))$ into the possible solution list.

  Note that if the event in question occurs, the following two equations hold:

$$\begin{cases} \mathsf{k}_0 = \mathbf{Key}(\hat{f}_0(\mathsf{pw}_0, \mathsf{pek}) \cdot \mathsf{ek}_{\mathtt{C}}^i, \hat{f}_1(\mathsf{pw}_0, \mathsf{pek}')); \\ \mathsf{k}_1 = \mathbf{Key}(\hat{f}_0(\mathsf{pw}_1, \mathsf{pek}) \cdot \mathsf{ek}_{\mathtt{C}}^i, \hat{f}_1(\mathsf{pw}_1, \mathsf{pek}')). \end{cases}$$

If a doublepwS event occurs, $\mathcal{B}$ adds the correct answer to the list. Assume the Type-I(ii) intractability of the underlying homomorphic NIKE scheme holds, doublepwS only occurs with negligible probability. □

Experiment $\mathbf{Expt}_7$: This experiment is same as $\mathbf{Expt}_6$ except that if $\mathcal{A}$ makes a doublepwC event occur, halt and declare "loss". We suppose that when a query is made, the test for the events doublepwS, correctpwSend or pairedpwGuess occurs after the test for doublepwC.

Claim 33. $\mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},7} \leq \mathsf{Adv}^{\mathrm{bpr}}_{\mathcal{A},6} + \mathsf{negl}(\lambda)$.

Proof. If $\mathcal{A}$ makes a doublepwC event occur, it will notice the change. Next, we will construct an algorithm $\mathcal{B}$ that attempts to break the Intractability-I (ii) of the underlying homomorphic NIKE scheme by running $\mathcal{A}$ as a subroutine. Given a challenge $\overline{\mathsf{pek}}'$, $\mathcal{B}$ changes the way to answer $\mathrm{Send}(\cdot)$ queries as follows:

- In a $\mathrm{Send}_1(\mathtt{S}, j, \mathsf{in\_msg})$ query, set $\mathsf{pek}' = \overline{\mathsf{pek}'} \cdot \mathsf{ek}_{\mathtt{S}}^j$, where $(\mathsf{ek}_{\mathtt{S}}^j, \mathsf{dk}_{\mathtt{S}}^j) \xleftarrow{\$} \textsc{Gen}(\mathsf{pp})$.
- Tests for correctpwSend (from $\mathbf{Expt}_4$), pairedpwGuess (from $\mathbf{Expt}_5$) and doublepwS (from $\mathbf{Expt}_6$) are not made.
- When $\mathcal{A}$ finishes, for every pair of $\mathcal{H}$ queries on values $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}_0|\mathsf{pw}_0$ and $\mathtt{C}|\mathtt{S}|\mathsf{pek}|\mathsf{pek}'|\mathsf{k}_1|\mathsf{pw}_1$ with $\mathsf{pw}_0 \neq \mathsf{pw}_1$, there was a $\mathrm{Send}_1(\mathtt{S}, j, \mathsf{pek})$ query that generates pek', do as follows:
  - Compute $\bar{\mathsf{k}}_0 = \textsc{Key}(\mathsf{dk}_{\mathtt{S}}^j, \hat{f}_1(\mathsf{pw}_0, \mathsf{pek}))$ and $\bar{\mathsf{k}}_1 = \textsc{Key}(\mathsf{dk}_{\mathtt{S}}^j, \quad \hat{f}_1(\mathsf{pw}_1, \mathsf{pek}))$.
  - Compute $\mathsf{k}_0^* = \mathsf{k}_0/\bar{\mathsf{k}}_0$ and $\mathsf{k}_1^* = \mathsf{k}_1/\bar{\mathsf{k}}_1$.
  - Add $(\mathsf{pek}, (\mathsf{pw}_0, \mathsf{k}_0^*), (\mathsf{pw}_1, \mathsf{k}_1^*))$ into the possible solution list.

  Note that if the event in question occurs, the following two equations hold:

$$\begin{cases} \mathsf{k}_0 = \mathbf{Key}(\hat{f}_0(\mathsf{pw}_0, \mathsf{pek}), \hat{f}_1(\mathsf{pw}_0, \mathsf{pek}') \cdot \mathsf{ek}_{\mathtt{S}}^j); \\ \mathsf{k}_1 = \mathbf{Key}(\hat{f}_0(\mathsf{pw}_1, \mathsf{pek}), \hat{f}_1(\mathsf{pw}_1, \mathsf{pek}') \cdot \mathsf{ek}_{\mathtt{S}}^j). \end{cases}$$

If a doublepwC event occurs, $\mathcal{B}$ adds the correct answer to the list. Assume the Intractability-I (ii) of the underlying homomorphic NIKE scheme holds, doublepwC only occurs with negligible probability. □

Experiment $\mathbf{Expt}_8$: This experiment is same as $\mathbf{Expt}_7$ except that it uses an internal password oracle that holds all passwords and only accepts simple queries that test whether a given password is the correct password for a given users pair. The test for password-guesses (from $\mathbf{Expt}_4$) is changed so that whenever $\mathcal{A}$ guesses a password, a query is submitted to the oracle to determine if it is correct.

By inspection, $\mathbf{Expt}_7$ and $\mathbf{Expt}_8$ are indistinguishable.

Bounding the advantage in $\mathbf{Expt}_8$. The probability of $\mathcal{A}$ succeeding in $\mathbf{Expt}_8$ can be bounded:

$$\Pr[\mathcal{A} \text{ succ. in } \mathbf{Expt}_8] \leq \Pr[\text{correctpwSend}] + \Pr[\mathcal{A} \text{ succ. in } \mathbf{Expt}_8 | \neg\text{correctpwSend}] \Pr[\neg\text{correctpwSend}].$$

It is straightforward that $\Pr[\text{correctpwSend}] \leq n_{\mathrm{Send}}/|\mathcal{PW}|$.

If correctpwSend does not occur, then $\mathcal{A}$ succeed by making Test query to a fresh instance Π_{U}^{i}. Recall that Reveal queries are not allowed on the target instance and it's paired instance, and session id includes the messages pek and pek', no more than one server instance and one client instance will accept with the same session id. Thus, the output of Reveal queries is independent of the session key of Π_{U}^{i}. Then recall in $\mathbf{Expt}_2$, no unpaired client or server instance will terminate, and thus the target fresh instance must be paired. However, a $(\mathcal{H}, \cdot)$ query will never reveal the session key of Π_{U}^{i} if it is paired (from $\mathbf{Expt}_5$). Therefore, the view of the adversary is independent of the session key of Π_{U}^{i}, so that the probability $\Pr[\mathcal{A} \text{ succeeds} | \neg \mathsf{correctpwSend}] = 1/2$ holds. We can conclude that $\Pr[\mathcal{A} \text{ succeeds in } \mathbf{Expt}_8] \leq 1/2 + n_{\mathrm{Send}}/|\mathcal{PW}|$.

Taken all above together, we have that

$$\mathsf{Adv}_{\mathcal{A},\pi}^{\mathrm{bpr}} \leq n_{\mathrm{Send}}/|\mathcal{PW}| + \mathsf{negl}(\lambda).$$

□