Just How Secure is SRP, Really?

Jiayu Xu and Zhiyuan Zhao

Oregon State University, {xujiay,zhaozhiy}@oregonstate.edu

Abstract

An asymmetric Password-Authenticated Key Exchange (aPAKE) protocol allows a client and a server to agree upon a cryptographic key, with the only information shared in advance being a low-entropy password (memorized by the client) and a corresponding password file (stored in the server). The standard security definition for aPAKE is in the Universally Composable (UC) framework. Since aPAKE does not rely on a PKI, such protocols have received much attention in recent years due to their potential of replacing the traditional "password-over-TLS" approach for client-server authentication on the Internet.

One of the only two aPAKE protocols currently deployed in practice is Secure Remote Password (SRP) (Wu, NDSS 1998), which is used by millions of users on the Internet. A formal security analysis of SRP has long been elusive; the only security proof to date is the one by Dayanikli and Lehmann (CSF 2024) which is in a (somewhat non-standard) variant of UC called UC with angels. This framework allows the simulator access to an additional oracle that solves certain hard cryptographic problems.

In this work, we present a comprehensive new analysis of SRP, and clarify a number of ambiguities about its security:

1. We formally prove that SRP is UC-secure *if and only if* one computational problem in the field is hard and one decisional problem is easy. As the decisional problem is likely hard in the field, this strongly suggests that SRP is *not* UC-secure and hence justifies the usage of UC with angels in the security analysis;
2. On the other hand, we show that the "angel" given to the simulator in the Dayanikli–Lehmann analysis is stronger than necessary, and can be replaced by a weaker oracle;
3. Finally, we prove that UC-with-angels-security is still stronger than the game-based security for aPAKE, i.e., SRP is still game-based secure under some reasonable assumptions.

Overall, we pinpoint the *exact conditions* under which SRP can be proven secure, reducing its security to a number of underlying hardness problems. Along the way we also identify and bridge multiple gaps in the Dayanikli–Lehmann analysis — most notably, the concrete security bound — and apply novel proof techniques that might find application in other contexts.

[SRP] bears all the marks of a protocol invented in the prehistoric days of crypto. [...] [The design ideas of SRP] make any reasonable security proof somewhat challenging.

— Matthew Green, 2018 [Gre18]

Our paper gives the first security analysis of SRP in the universal composability (UC) framework. [...] Our proof is in the "Angel-based UC framework", a relaxation of the UC framework which gives all parties access to an oracle with super-polynomial power.

— Dennis Dayanikli and Anja Lehmann, 2024 [DL24]

In advanced settings such as the Angel-based security framework, the question of simulation plausibility is harder to answer as the power available to the simulator is less clear-cut.

— Brandon Broadnax et al., 2021 [BMM21]

1 Introduction

One fundamental principle of modern cryptography is that any public-key scheme must be accompanied by a security proof, which reduces the security of the scheme to some reasonable, hopefully well-studied assumptions. This is so widely accepted today that it might be hard to imagine that things were far different back in the 1990s and early 2000s, when it was standard to propose a scheme first and then someone else would give a security proof a few years later; as a typical example, the Schnorr signature scheme was first introduced in 1989 [Sch89], yet a formal security proof did not appear until 1996 [PS96]. Over the years, most of these "leftover" schemes from the 1980s and 1990s have been formally analyzed, with two notable exceptions: DSS/ECDSA and SRP. There have been a large number of works on the (limits of) formal security analysis of ECDSA; this work focuses on SRP.

SRP, what is it? *Secure Remote Password (SRP)* is a protocol for *asymmetric Password-Authenticated Key Exchange (aPAKE)*, which allows two parties — a client who holds a raw password and a server who holds a one-way mapping of the password (called the password file) — to jointly establish a cryptographic key. Crucially, the password is expected to have low entropy, and the protocol must be secure against a man-in-the-middle adversary; the latter property provides a significant advantage over the more commonly seen

"password-over-TLS" method of password authentication, which requires an authenticated channel and is vulnerable to various forms of "PKI failure" (e.g., due to phishing attacks).

SRP has a somewhat complicated history. The whole project began in 1997 [Wu97]. SRP-1, the original version, is insecure as an adversary who compromises the server can log in as a client, without performing an offline attack to find the password. This issue was fixed in SRP-2 and SRP-3, with SRP-3 being published in 1998 [Wu98]. However, SRP-3 was later found vulnerable to the so-called *two-for-one guessing attack*, where an adversarial client can test *two* different passwords in a session. SRP-6 and SRP-6a were introduced in 2002 to solve this problem [Wu02], with SRP-6a (henceforth SRP) being the stable and commonly used version.

See Figure 1 for a schematic presentation of SRP.

Broad applications, outdated design ideas. For a very long time, SRP was the *only* aPAKE protocol widely used in practice. It has been adopted by the 1Password password manager [Fil18], Proton Mail [Bar16], Apple's iCloud keychain escrow [App21], among others. In recent years, the status of SRP has been challenged by some next-generation aPAKE protocols such as OPAQUE [JKX18], which has been selected for standardization by the IETF [Cry20]; however, SRP still remains one of the most commonly seen aPAKE protocols in practice, and we do not see it being replaced by any other protocols in the near future.

Given its popularity, it might be surprising that SRP has been sharply criticized as "design rationale is not really clear" and "bears all the marks of a protocol invented in the prehistoric days of crypto" [Gre18]. There are two related reasons for this: First, SRP is defined over a *finite field*, with both addition and multiplication operations. This makes it difficult to implement SRP on elliptic curves. Second, exactly because SRP works in a field, its security cannot be reduced to standard Diffie–Hellman-type group assumptions, which do not involve addition.

The second point above also means that for people who wanted to give a formal security proof for SRP, they simply didn't know what assumption(s) to reduce to. The protocol sat there for over 20 years, used by millions of people every day but without a security proof — even after nearly all other contemporaneous protocols had been supplied with a formal analysis. This embarrassing situation was finally ended by the groundbreaking work of Dayanikli and Lehmann [DL24], of which we give an overview in Section 1.1.

Is SRP still relevant? Given the outdated design ideas of SRP and the lack of security reductions to standard assumptions, it is natural to ask whether SRP is still worth (formally) analyzing. Indeed, people have advocated as early as 2018 [Gre18] that SRP should not be used anymore and should be replaced by newer aPAKE protocols with better security properties, such as OPAQUE. Why do we still study SRP in 2026?

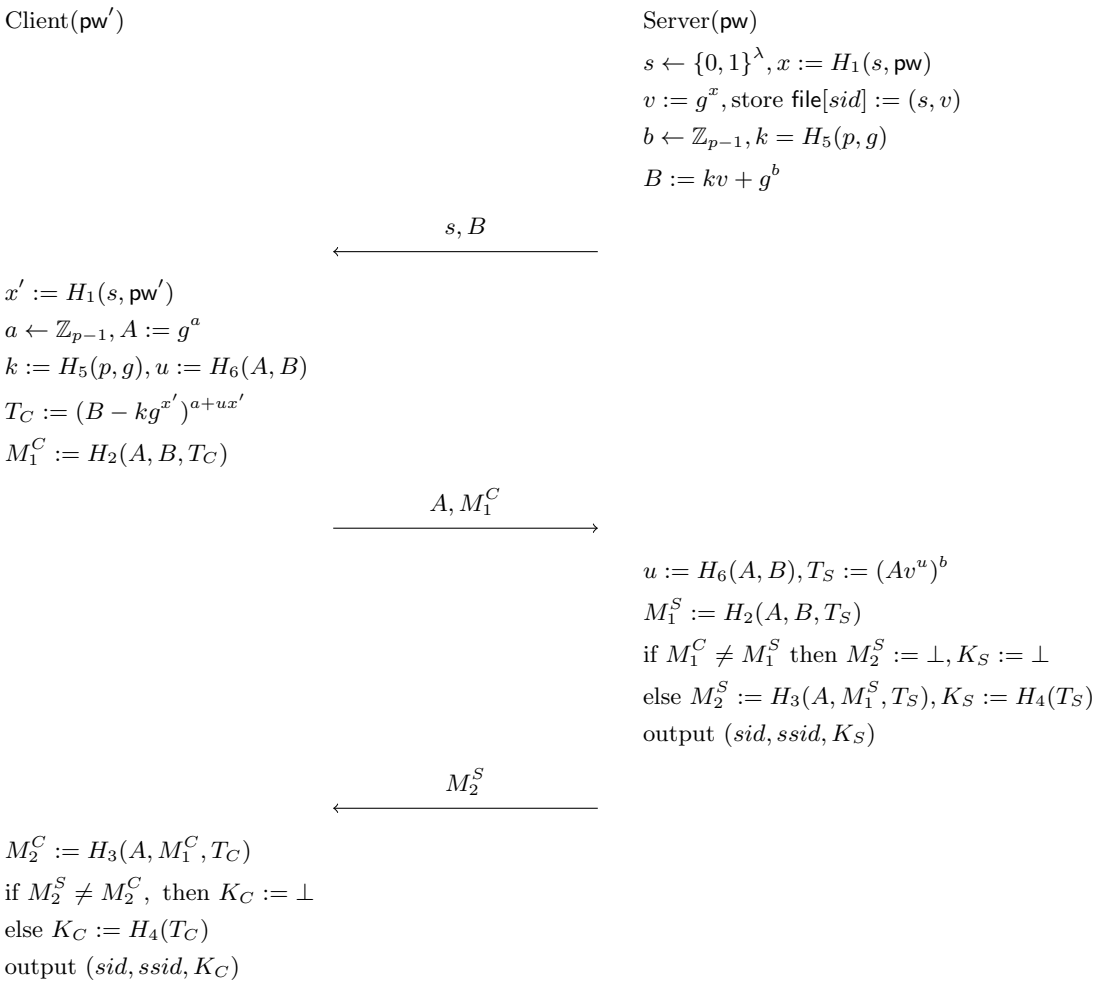

Figure 1: SRP. It works in finite field $\mathbb{F}_p$ of prime order p, and g is a generator of the cyclic group $\mathbb{F}_p^*$. $H_1 : \{0,1\}^* \to \mathbb{Z}_{p-1}$, $H_2 : \{0,1\}^* \to \mathbb{F}_p$, $H_3 : \{0,1\}^* \to \mathbb{F}_p$, $H_4 : \{0,1\}^* \to \{0,1\}^\lambda$, $H_5 : \{0,1\}^* \to \mathbb{F}_q$, $H_6 : \{0,1\}^* \to \mathbb{Z}_{p-1}$ are hash functions (modeled as ROs). This figure shows SRP-6a; in SRP-6 k is fixed to be 3

The primary reason is that despite all of its deficiencies, SRP remains widely used in real life, and as such, a formal security analysis is highly relevant in practical terms. Albrecht and Paterson [AP24] summarized this point well:

> *[I]f we consider science to be the process by which we get to understand the world, then a security analysis of a significant object in the world — say a protocol relied upon by millions of people — is then in and of itself a valid scientific result.*

We would go a step further and claim that such an analysis is *crucial* to the security and privacy of millions of users online.

Furthermore, while the design ideas of SRP are (admittedly) of little theoretical interest, some techniques in its *security proof* — in particular, creative usage of the forking lemma — are highly interesting and may find further applications in other contexts (see Section 1.2 for an overview).

1.1 Overview of the Dayanikli–Lehmann Analysis

Dayanikli and Lehmann [DL24] gave two formal security analyses of SRP. The first result deals with a simplified version of SRP (with scrambling parameters k and u in Figure 1 removed), which they showed to be secure against so-called semi-passive adversaries — adversaries that can only eavesdrop and compromise the server — assuming CDH is hard. Their proof is in the Universal Composability (UC) framework, which is the standard security definition for aPAKE [GMR06].[1] This is not a very surprising result, as it merely shows that the security of (simplified) SRP is on the same level of the unauthenticated hashed Diffie–Hellman key exchange plus resilience against server compromise.

The main contribution of Dayanikli and Lehmann is their second result, which states that SRP is secure against man-in-the-middle adversaries (that can also compromise the server). They introduced a new field assumption called *additive Simultaneous Diffie–Hellman (aSDH)*, which roughly says that given uniformly random field elements A, Δ, it is hard to come up with B, T_1, T_2 such that $T_1 = \mathbf{cdh}(B, A)$ and $T_2 = \mathbf{cdh}(B + \Delta, A)$. This accounts for the two-for-one guessing attack (i.e., if aSDH is hard then the two-for-one guessing attack is infeasible), and [DL24] gives some informal justifications of the necessity of aSDH.

However, another more sinister issue pops up while attempting to prove the UC-security of SRP. Referring to Figure 1, imagine the adversary, who assumes the role of the client, sends a message (A^*, M_1^{C*}) to the (honest) server. UC-security requires the simulator to extract a password guess (if any) from this message. The simulator can first extract a T from M_1^{C*} by observing the adversary's H_2 queries; the correct password guess pw_i, which the simulator

[1] The UC framework is especially suitable for the analysis of aPAKE, since it guarantees security under arbitrary composition — which is how aPAKE is commonly used — and it handles password reuse across different accounts, which is hard to model in the traditional game-based setting.

must extract, satisfies

$$\mathbf{ddh}(B - kg^{x_i}, A^* g^{ux_i}, T) = 1,$$

where $x_i = H_1(s, \mathsf{pw}_i)$. The simulator knows the discrete log of neither $A^* g^{ux_i}$ (the adversary chooses A^*) nor $B - kg^{x_i}$ (x_i must be uniformly random, so the simulator cannot know the discrete log of all $B - kg^{x_i}$ when there are multiple x_i), so it has no idea which x_i corresponds to the adversary's password guess!

To get around this issue, [DL24] resorts to the weaker *UC with angels* (a.k.a. angel-based UC) framework [PS04]. Somewhat contrivedly, this framework gives both the simulator and the real adversary an additional oracle (the "angel") that may solve some hard problems; [DL24] sets the "angel" to be a DDH oracle, which immediately resolves the extraction issue above. [DL24] proceeds to prove that SRP is UC-with-a-DDH-oracle-secure assuming *Gap* aSDH is hard.

At first glance, the Dayanikli–Lehmann analysis might look highly unsatisfactory. The original intention of UC with angels was to bypass some impossibility results in UC (e.g., UC commitments without trusted setup), and over the years the framework has mostly been used for theoretical purposes and rarely used to show the security of some protocols in real life — after all, what does it even mean in practice to say DDH is hard, but some imaginary "angel" allows the adversary to solve DDH?

On the other hand, from the above it seems that SRP is simply *not* UC-secure, and a UC with angels analysis might be the best we can do. This opens the door to a plethora of questions, none of which is answered in [DL24]:

1. Is it really impossible to prove the regular UC-security of SRP, under some reasonable assumptions?
2. If we work in the UC with angels framework, does the angel have to be a DDH oracle? Does a weaker oracle suffice?
3. Is the non-standard aSDH assumption really necessary? Do we need aSDH, or Gap aSDH?
4. Why do we stick to (some form of) UC in the first place — is it possible to prove the game-based security of SRP?
5. And finally, is the security proof in [DL24] correct?

1.2 Our Contributions

In this work, we present a comprehensive study of the security of SRP, answering *all of the questions above*. We summarize our contributions below:

1.2.1 Exact Conditions for the UC-Security of SRP

In Sections 4 and 5 we show that SRP is UC-secure *if and only if* the aSDH problem is hard and the *Multi-message Choose DDH Check (Multi-message CDC)* problem is "very easy". We formally define these assumptions in Section 3 but give a high-level summary here:

Hardness of aSDH. We confirm that aSDH needs to be hard to ensure the infeasibility of the two-for-one guessing attack. However, our version of aSDH assumption is slightly weaker than what's in [DL24] and precisely captures the security of SRP. The difference is that we require the adversary to output B *before* seeing A, whereas the B in [DL24] can depend on A. Furthermore, we show that the gap version of aSDH is not needed for the UC-security of SRP, which requires a more sophisticated security proof than [DL24].

The proof that hardness of aSDH is *necessary* for the UC-security of SRP is surprisingly difficult. Intuitively, solving aSDH allows the adversary to perform a two-for-one guessing attack, in which the adversary can test two passwords in a session. But if a PPT adversary can only solve aSDH with probability (say) 0.2, then it can only test 0.4 passwords per session in expectation, so it is not even clear whether this forms a real attack!

We get around this by proving an additional property of aSDH, which might be of independent interest. Using methods similar to the proof of CDH self-correction [Sho97, CKS08], we show that an aSDH solver can essentially check on its own if it has won. This allows an SRP attacker to discard wrong answers while trying to solve aSDH, hence boosting its chance of success. The concrete proof (in Section 5.2) involves careful considerations and interesting techniques.

Very easiness of Multi-message CDC. The next question is what exactly allows the simulator to extract a password guess in the scenario explained above, where the adversary plays the role of the client. We show that the necessary and sufficient condition is the "very easiness" of what we call the Choose DDH Check problem, in which the adversary must tell which x_i satisfies

$$\mathbf{ddh}(B - kg^{x_i}, A, T) = 1$$

for pseudorandom $B, k, x_1, \cdots, x_q$ chosen by the adversary and any A, T chosen by the game challenger.[2] By "very easiness", we mean that (1) the challenger's values A, T can follow *any* (efficiently samplable) distribution, and (2) the adversary must win with *overwhelming* probability. (For example, DDH is easy, but not very easy, in $\mathbb{Z}_p^*$.) This matches the requirements on a UC simulator.

Compared with the intuitive analysis in Section 1.1, we were able to remove the parameter u in the assumption, but did not manage to simplify it further. A DDH oracle would clearly allow the adversary to solve CDC; vice versa, it appears that the only way to solve CDC is via solving DDH, although we do not prove any results in this direction and leave it for future work. All in all, this provides strong evidence that SRP is highly unlikely to be UC-secure, and confirms that the overall methodology of [DL24] is sound.

Security in UC with angels. The above also easily implies the main result of [DL24], namely SRP is UC-secure with a DDH oracle. One caveat is that *Gap*

[2] In fact, since aPAKE is a multi-session protocol where different login sessions share the same password file, we need a *multi-session* version of CDC. We simplify it to what we call Multi-message CDC, where there is a single session but multiple B values.

aSDH becomes necessary here, since the *reduction* — not only the UC simulator — needs to answer the real adversary's DDH oracle queries correctly.

1.2.2 Game-Based Security of SRP

As mentioned above, one issue with UC with angels is that it is not entirely clear what security in this framework implies in real life. We partially answer this question by considering the relation between security in UC with angels and the *game-based* security of aPAKE [BP13], which does not model security under (say) password reuse but does provide a basic level of security guarantee. Concretely, we prove two results in Section 6:

- In Section 6.2 we show a general result that security of aPAKE in UC with *any* angels implies game-based security. Since SRP is UC-secure with a DDH oracle under Gap aSDH, this implies that SRP is game-based secure under Gap aSDH. To the best of our knowledge, this result that connects UC with angels security and game-based security is the first of its kind, and we believe it is of independent interest.
- In Section 6.3 we give a specific result on SRP, namely SRP is game-based secure under aSDH (and the gap version is unnecessary).

Overall, we provide a "sanity check" of the validity of UC with angels (in the context of aPAKE), and also confirm the game-based security of SRP under minimal assumptions.

1.2.3 Gaps in the Dayanikli–Lehmann Analysis

One would imagine that our UC-security proof for SRP (under hardness of aSDH and very easiness of CDC) is very similar to the UC-with-angels-security proof in [DL24]. While this is true at a high level, we observe that *the proof in [DL24] does not go through*, due to multiple subtle gaps which we elaborate below. These gaps result in their concrete security bound for SRP drastically deviating from the correct one, among other issues.

(Mis)use of the forking lemma. The analysis of SRP is a rare example of a UC proof that uses the forking lemma. To begin with, it might be surprising that the forking lemma can be used in a UC proof at all, since a UC simulator must perform straightline extraction and cannot rewind the adversary. However, this does not prevent a *reduction* from rewinding the adversary, and indeed, the forking lemma is used in one of the hybrids where we reduce to (a variant of) CDH.

We defer an explanation of why we need the forking lemma to our security proof; here we just point out that

- *Our application of the forking lemma is much subtler than usual*, which is overlooked in [DL24] and is a technical contribution of ours. At a high level, this is because we want to reduce to CDH rather than DL or Gap

CDH; the critical difference is that the we cannot check if a CDH solver wins or not, yet *when using the forking lemma to construct a reduction, we must check if the "bad event" happens* (see, e.g., the third-to-last line of algorithm A in [BN06, p.13]). To get around this, we have to reduce to a variant of CDH with additional oracles, which turns out to be 2-Query Oracle Diffie–Hellman (ODH) — which is in turn equivalent to CDH. We believe that our "adding an additional oracle" trick is an interesting proof technique on its own, and might find further forking lemma applications that reduce to CDH, or perhaps *any* computational assumption that does not allow for checking if the adversary wins or not. Unfortunately, [DL24] (specifically, Game G_5 in their proof) completely misses the details of the forking lemma.

- Furthermore, the concrete security bound in the forking lemma hybrid in [DL24] — which is the loosest point in the entire security proof — is far from correct. Very roughly, the probability of the "bad event" in their proof is $\ell^2 \cdot \varepsilon$ for ℓ RO queries and the reduction's advantage ε, whereas the correct bound should be $\ell^{\frac{3}{2}} \cdot \sqrt{\varepsilon}$. The lesson here is worth highlighting:

  > *The forking lemma incurs a quadratic security loss, meaning that the reduction's advantage is roughly the square of the probability of the "bad event" (times any loss caused by the reduction's guesses). A concrete bound without such a loss is guaranteed to be incorrect.*

  We stress that for a widely used protocol like SRP, a concrete security bound is essential to a formal security analysis: it is crucial to the choice of the security parameter, which in turn affects the efficiency of the protocol.

Other improvements over [DL24]. Apart from the misuse of the forking lemma, there are a number of other issues in the concrete security bound in [DL24]:

1. To begin with, their concrete security analysis is incomplete: one of their hybrids (Game G_8) merely claims that the environment's distinguishing advantage is negligible, without giving an upper bound. (Somehow their theorem statement still gives an overall security bound. It seems that they simply treat the distinguishing advantage in this game as 0, which is not the case.)

2. *Number of sessions:* Since multiple sessions — i.e., client logins — in SRP share the server's password file as a common parameter, one cannot simply assume there is only a single session and claim that security extends to the multi-session case due to composability; instead, a multi-session analysis in the *Joint UC (JUC) framework* [CR03] must be performed.[3]

[3] Technically speaking, every server password file represents a session, and every login attempt by the client represents a sub-session.

In particular, the reductions have to make a guess on in which session the reduction needs to embed its challenges, losing a factor of ℓ_{SS}, the number of sessions.
Unfortunately, it is unclear whether [DL24] considers the multi-session setting or not. One of their games (Game G_9) includes the ℓ_{SS} factor in their *statistical* analysis; however, none of their *reductions* consider the multi-session case, and the reductions are described as if they somehow "know" which session to embed their challenges in. Since ℓ_{SS} might be very large in practice, missing this factor means underestimating the environment's distinguishing advantage significantly.[4]

3. [DL24] also overlooks a number of (statistical) "bad events" that would allow the environment to distinguish, although the errors here are relatively minor.

Apart from concrete security bounds, parts of the hybrid argument in [DL24] do not go through, due to incorrect order of hybrids. In a hybrid argument, a reduction generally cannot simulate the game honestly as the challenger does (due to losing some information while embedding its challenges); as such, we might need to add some "bridging" games for the reduction to go through. In some reductions, [DL24] puts the "bridging" game *after* where it is needed, causing one of the most important and subtle reductions in the proof — the aSDH reduction — not being able to simulate the game.

In this work, we present a much more detailed and careful security analysis, with an overall upper bound of the environment's distinguishing advantage at the end of the proof. Our concrete security bound, as well as the sequence of games, are drastically different from the proof in [DL24].

We stress that while all of the issues above can be fixed, they require heavy lifting and illustrate various "hidden" subtleties in the security analysis of SRP. In our UC-security proof of SRP (Section 4), we present detailed comparisons between our proof and the one in [DL24].

Recap. While it is widely acknowledged that a formal security analysis of SRP is difficult, it has never been articulated just *how* difficult it is. By pinpointing the precise conditions under which SRP is UC-secure (or UC-with-angels-secure), presenting a game-based proof for SRP, and giving the exact security bound for SRP, we hope to "measure" the difficulty of a formal security analysis for SRP from multiple angles, and add some solid technical inputs to the broader discussion of whether SRP should still be used.

Finally, in Section 7 we recapitulate our work and present some philosophical discussion on whether SRP should continue to be used today.

[4] This conceptual mistake is very common in multi-session protocols; as another example, the UC security analysis of OPAQUE [JKX18, Section 5.3] also suffers from this issue.

2 The UC aPAKE Functionality

Throughout this work, we use λ to denote the security parameter. The UC aPAKE functionality is shown in Figure 2. For an explanation, see e.g., [DL24, Section 2].

Differences with the standard functionality. Our UC aPAKE functionality follows [DL24], and slightly deviates from the standard functionality in [GMR06]. In the standard functionality, if authentication fails (due to either mismatching passwords or a failed online attack) its output key is set to random; in our functionality it is set to $\perp$. This models *explicit authentication* and matches the behavior of SRP.

2.1 Forking Lemma

We recall the forking lemma from [BN06].

Theorem 1 (Forking Lemma)**.** *Let $q \in \mathbb{N}$ and $\mathbf{H}$ be a finite set. Let $\mathcal{W}$ be any randomized algorithm that on input X and $h_1, \cdots, h_q \in \mathbf{H}$, outputs an index $I \in [0, q]$ and a side output σ, and let $\mathcal{D}$ be the distribution of $\mathcal{W}$'s input. Define $\mathcal{W}$'s* accepting probability *as*

$$\mathbf{Acc}_{\mathcal{W}} := \Pr\left[I \neq 0 : \begin{array}{c} X \leftarrow \mathcal{D} \\ h_1, \cdots, h_q \leftarrow \mathbf{H} \\ (I, \sigma) \leftarrow \mathcal{W}(X, h_1, \cdots, h_q) \end{array} \right].$$

Now define the forker $\mathcal{F}^{\mathcal{W}}(X)$ *as follows:*

1. *Pick random coins ρ for $\mathcal{W}$.*
2. *Sample $h_1, \cdots, h_q \leftarrow \mathbf{H}$.*
3. *Run $(I, \sigma) \leftarrow \mathcal{W}(X, h_1, \cdots, h_q; \rho)$.*
4. *If $I = 0$ then return $(0, \perp, \perp)$.*
5. *Sample $h'_I, \cdots, h'_q \leftarrow \mathbf{H}$.*
6. *Run $(I', \sigma') \leftarrow \mathcal{W}(X, h_1, \cdots, h_{I-1}, h'_I, \cdots, h'_q; \rho)$.*
7. *If $I = I' \wedge h_I \neq h'_I$ then return $(1, \sigma, \sigma')$.*
8. *Else return $(0, \perp, \perp)$.*

Define $\mathcal{F}^{\mathcal{W}}$'s accepting probability as

$$\mathbf{Frk}_{\mathcal{F}^{\mathcal{W}}} := \Pr\left[b = 1 : \begin{array}{c} X \leftarrow \mathcal{D} \\ (b, \star, \star) \leftarrow \mathcal{F}^{\mathcal{W}}(X) \end{array} \right].$$

Password Registration

On (`StorePwdFile`, *sid*, pw) from S, create record $\langle$`file`, *sid*, S, pw$\rangle$ marked `fresh`.

Stealing Password Data

1. On (`StealPwdFile`, *sid*, S) from $\mathcal{A}$: If no record $\langle$`file`, *sid*, S, pw$\rangle$ exists, return "no password file." Otherwise, mark the record as `compromised`. If a record $\langle$`offline`, *sid*, S, pw$\rangle$ exists, send pw to $\mathcal{A}$.
2. On (`OfflineTestPwd`, *sid*, S, pw*) from $\mathcal{A}$:
   (a) If a record $\langle$`file`, *sid*, S, pw$\rangle$ marked as `compromised` exists and pw* = pw, return "correct guess." Otherwise, return "wrong guess."
   (b) Otherwise, record $\langle$`offline`, *sid*, S, pw$^*\rangle$.

Password Authentication

1. On (`CltSession`, *sid*, *ssid*, S, pw$'$) from C: If no record $\langle ssid, C, \ldots\rangle$ exists, create a record $\langle ssid, C, S, sid,$ pw$', \text{client}\rangle$ marked as `fresh` and send (`CltSession`, *sid*, *ssid*, C, S) to $\mathcal{A}$.
2. On (`SvrSession`, *sid*, *ssid*, C) from S: If no record $\langle ssid, S, \ldots\rangle$ exists, retrieve the record $\langle$`file`, *sid*, S, pw$\rangle$. If it exists, create a record $\langle ssid, S, C, sid,$ pw, server$\rangle$ marked as `fresh`, then send (`SvrSession`, *sid*, *ssid*, S, C) to $\mathcal{A}$.

Active Session Attacks

1. On (`TestPwd`, *sid*, *ssid*, P, pw*) from $\mathcal{A}$: If a record $\langle ssid, P, P', sid,$ pw, role$\rangle$ marked as `fresh` exists:
   (a) If pw* = pw, mark it as `compromised` and return "correct guess".
   (b) Otherwise, mark it as `interrupted` and return "wrong guess".
2. On (`Impersonate`, *sid*, *ssid*, C, S) from $\mathcal{A}$: If a record $\langle ssid, C, S, sid,$ pw, client$\rangle$ marked as `fresh` exists:
   (a) If a record $\langle$`file`, *sid*, S, pw$\rangle$ is marked as `compromised`, mark the session as `compromised` and return "correct guess".
   (b) Otherwise, mark the session as `interrupted` and return "wrong guess".

Key Generation and Authentication

1. On (`NewKey`, *sid*, *ssid*, P, K^*) from $\mathcal{A}$: If a record $\langle ssid, P, P', sid,$ pw, role$\rangle$ not marked as `completed` exists:
   (a) If the record is `compromised`, set $K \leftarrow K^*$.
   (b) Otherwise, if role = client and the record is `fresh`, and a matching server record exists with K', set $K \leftarrow K'$.
   (c) Otherwise, if role = server and the record is `fresh`, pick $K \leftarrow \{0,1\}^\lambda$.
   (d) Otherwise, set $K \leftarrow \perp$.
2. Send $(ssid, K)$ to P and mark the record as `completed`.

Figure 2: The UC aPAKE functionality $\mathcal{F}_{\text{aPAKE}}$

Then

$$\mathbf{Acc}_{\mathcal{W}} \leq \sqrt{q \cdot \mathbf{Frk}_{\mathcal{F}^{\mathcal{W}}}} + \frac{q}{|\mathbf{H}|}.$$

3 Background: Cryptographic Assumptions

In this section, we define the three assumptions needed in the security analysis of SRP: CDH, aSDH, and Choose DDH Check (CDC). CDH is a standard group assumption. aSDH is a field assumption first defined in [DL24], but we present a slightly different version that exactly matches the security requirement of SRP. CDC is a new field assumption that precisely captures what's needed for the simulator to go through; we require it to be "very easy", which is a weaker condition than giving the simulator a DDH oracle (as in [DL24]).

Notations. Let $\mathcal{G}$ be a PPT algorithm that on input 1^λ outputs $(\mathbb{F}_p, p, g)$, where $\mathbb{F}_p$ is the field of prime order p, and g is an arbitrary generator of the multiplicative cyclic group $\mathbb{F}_p^*$ (whose order is $p-1$). We require that $p-1$ have at least one big prime factor, e.g., $p = 2q+1$ where q is also a prime. Below we abuse notations and say a computational problem is hard/easy in $\mathbb{F}_p$, rather than $\mathcal{G}$.

For any $A \in \mathbb{F}_p^*$, define $\mathbf{dl}(A)$ to be $a \in \mathbb{Z}_{p-1}$ such that $g^a = A$. (Note that the zero element of $\mathbb{F}_p$ is not a group element and its $\mathbf{dl}$ is not well-defined.) For any $A, B \in \mathbb{F}_p^*$, define $\mathbf{cdh}(A, B) = A^{\mathbf{dl}(B)} = B^{\mathbf{dl}(A)}$. For any $A, B, C \in \mathbb{F}_p^*$, define $\mathbf{ddh}(A, B, C)$ to be 1 if $C = \mathbf{cdh}(A, B)$ and 0 otherwise.

For an algorithm $\mathcal{A}$ in a security game, we use $\mathbf{Adv}_{\mathcal{A}}^{\text{Game}}$ to denote its advantage.

3.1 Computational Diffie–Hellman (CDH) and Variants

Definition 1 (Computational Diffie–Hellman (CDH))**.** *The **CDH** game is defined as follows:*

1. *The game challenger samples $a, b \leftarrow \mathbb{Z}_{p-1}$ and outputs $A := g^a, B := g^b$.*
2. *$\mathcal{A}$ outputs $C \in \mathbb{F}_p^*$, and wins if $C = \mathbf{cdh}(A, B)$.*

We say the CDH problem is hard in $\mathbb{F}_p$ if for any PPT $\mathcal{A}$, the probability that $\mathcal{A}$ wins the above game is negligible.

Two-query Oracle Diffie–Hellman (2ODH). The following computational problem is needed in our forking lemma argument:

Definition 2 (Two-query Oracle Diffie–Hellman (2ODH))**.** *The **2ODH** problem is the same as the CDH problem, except that in the game $\mathcal{A}$ is equipped with two additional oracles:*

- $\mathcal{H} : \mathbb{F}_p^* \to \{0,1\}^\lambda$*, an RO;*[5]
- $\mathcal{O} : \mathbb{F}_p^* \to \{0,1\}^\lambda$*, which on input X returns $\mathcal{H}(X^{\mathbf{dl}(B)})$. $\mathcal{O}$ can be queried only twice.*

There are two differences between our formulation of 2ODH and what's generally called the Oracle Diffie–Hellman (ODH) problem [ABR01]: first, our 2ODH is a computational problem where the adversary needs to output the CDH value g^{ab}, whereas ODH is a decisional problem where the adversary needs to distinguish $\mathcal{H}(g^{ab})$ from random[6]; second, in 2ODH the adversary can only query $\mathcal{O}$ twice, whereas in ODH there can be arbitrarily many queries.

2ODH is equivalent to CDH, but with a $\binom{\ell}{2}$ reduction loss for ℓ queries to $\mathcal{H}$. This is because the adversary can query $\mathcal{O}(g^{r_1}), \mathcal{O}(g^{r_2})$ for arbitrary $r_1 \neq r_2$, and check if the results are equal to $\mathcal{H}(B^{r_1}), \mathcal{H}(B^{r_2})$; the reduction has to guess which two of the adversary's $\mathcal{H}$ queries are $\mathcal{H}(B^{r_1}), \mathcal{H}(B^{r_2})$ and program $\mathcal{O}$ accordingly.

3.2 Additive Simultaneous Diffie–Hellman (aSDH)

Two-for-One Guessing.

Definition 3 (q-for-One Guessing)**.** *The* **q-*for-One Guessing*** *game is defined as follows:*

1. *The game challenger outputs $k \leftarrow \mathbb{F}_p^*, x_1, \cdots, x_q \leftarrow \mathbb{Z}_{p-1}$ (subject to the condition that $x_1, \cdots, x_q$ are all distinct).*
2. *$\mathcal{A}$ outputs $B \in \mathbb{F}_p$.*
3. *The game challenger further outputs $A \leftarrow \mathbb{F}_p^*$.*
4. *$\mathcal{A}$ outputs $(i, T_i), (j, T_j) \in [q] \times \mathbb{F}_p^*$ for $i < j$, and wins if $T_i = \mathbf{cdh}(B - kg^{x_i}, A)$ and $T_j = \mathbf{cdh}(B - kg^{x_j}, A)$.*

We say the q-for-One Guessing problem is hard in $\mathbb{F}_p$ if for any PPT $\mathcal{A}$, the probability that $\mathcal{A}$ wins the above game is negligible.

[5] Technically, $\mathcal{H}$ excludes collisions, i.e., when it samples an output, it always excludes existing outputs from previous queries. This will simplify our application to SRP.

[6] To prevent the adversary from trivially winning, in ODH one has to forbid the $\mathcal{O}(A)$ query. There is no such requirement in 2ODH.

Definition 4 (Two-for-One Guessing)**.** *The* ***Two-for-One Guessing*** *problem is the same as the q-for-One Guessing problem, except that q is fixed to be* 2.[7]

It is not hard to see that the q-for-One-Guessing and Two-for-One Guessing problems are equivalent:

Theorem 2. *The q-for-One-Guessing problem is hard in* $\mathbb{F}_p$ *if and only if the Two-for-One-Guessing problem is hard in* $\mathbb{F}_p$*.*

Proof. "$\Rightarrow$": This is immediate.

"$\Leftarrow$": For any PPT adversary $\mathcal{A}$ in the q-for-One Guessing game, we construct an adversary $\mathcal{A}'$ in the Two-for-One Guessing game:

1. On k, x, x' from its challenger, $\mathcal{A}'$ samples two indices $i < j \leftarrow [q]$, sets $x_i := x, x_j := x'$, and samples $x_\ell \leftarrow \mathbb{Z}_{p-1}^*$ for all $\ell \notin \{i, j\}$ (subject to the condition that $x_1, \cdots, x_q$ are all distinct). $\mathcal{A}'$ invokes $\mathcal{A}$ on $(k, x_1, \cdots, x_q)$.
2. When $\mathcal{A}$ outputs B, $\mathcal{A}'$ outputs the same value.
3. On A from its challenger, $\mathcal{A}'$ sends the same value to $\mathcal{A}$.
4. When $\mathcal{A}$ outputs $(i^*, T_{i^*}), (j^*, T_{j^*})$, if $(i^*, j^*) \neq (i, j)$, $\mathcal{A}'$ aborts. Otherwise $\mathcal{A}'$ outputs (T_i, T_j).

The key observation is that *the positions of i and j are independent of $\mathcal{A}$'s view*, so $\mathcal{A}$'s output indices (i^*, j^*) are independent of (i, j). Thus, $\Pr[(i^*, j^*) = (i, j)] = 1/\binom{q}{2}$. Furthermore, if $(i^*, j^*) = (i, j)$ and $\mathcal{A}$ wins, then $T_i = \mathbf{cdh}(B - kg^{x_i}, A)$ and $T_j = \mathbf{cdh}(B - kg^{x_j}, A)$, so $\mathcal{A}'$ also wins. We have

$$\mathbf{Adv}_{\mathcal{A}'} = \frac{1}{\binom{q}{2}} \cdot \mathbf{Adv}_{\mathcal{A}},$$

and since the Two-for-One-Guessing problem is hard, $\mathbf{Adv}_{\mathcal{A}'}$ is negligible, which implies that $\mathbf{Adv}_{\mathcal{A}}$ is also negligible. This shows that the q-for-One-Guessing problem is hard. □

Equivalence between Two-for-One Guessing and aSDH. The essence of the Two-for-One Guessing problem is that the adversary needs to compute $T_1 = \mathbf{cdh}(B_1, A)$ and $T_2 = \mathbf{cdh}(B_2, A)$, where A is uniformly random, the adversary has one degree of "freedom" while deciding B_1 and B_2, but $B_1 - B_2 = k \cdot (g^{x_1} - g^{x_2})$ is uniformly random to the adversary. This yields the following simplified form:

Definition 5 (Additive Simultaneous Diffie–Hellman (aSDH))**.** *The* ***aSDH*** *game is defined as follows:*

[7] Since there are only two indices, it is fixed that $i = 1$ and $j = 2$, and we may omit them in $\mathcal{A}$'s final output.

1. *The game challenger outputs* $\Delta \leftarrow \mathbb{F}_p^*$.
2. $\mathcal{A}$ *outputs* $B \in \mathbb{F}_p$.
3. *The game challenger further outputs* $A \leftarrow \mathbb{F}_p^*$.
4. $\mathcal{A}$ *outputs* $T_1, T_2 \in \mathbb{F}_p$, *and wins if* $T_1 = \mathbf{cdh}(B, A)$ *and* $T_2 = \mathbf{cdh}(B + \Delta, A)$.

We say aSDH is hard in $\mathbb{F}_p$ *if for any PPT* $\mathcal{A}$, *the probability that* $\mathcal{A}$ *wins the above game is negligible.*

Theorem 3. *The aSDH problem is hard in* $\mathbb{F}_p$ *if and only if the Two-for-One-Guessing problem is hard in* $\mathbb{F}_p$.

Proof. "⇒": For any PPT adversary $\mathcal{A}$ in the Two-for-One-Guessing game, we construct an adversary $\mathcal{A}'$ in the aSDH game:

1. On Δ from its challenger, $\mathcal{A}'$ samples $x_1, x_2 \leftarrow \mathbb{Z}_{p-1}^*$ (subject to the condition that x_1, x_2 are distinct), computes $k := \Delta \cdot (g^{x_1} - g^{x_2})^{-1}$, and invokes $\mathcal{A}$ on k, x_1, x_2.[8]
2. When $\mathcal{A}$ outputs B, $\mathcal{A}'$ computes $B' := B - kg^{x_1}$ and outputs B'.
3. On A from its challenger, $\mathcal{A}'$ sends the same value to $\mathcal{A}$.
4. When $\mathcal{A}$ outputs T_1, T_2, $\mathcal{A}'$ outputs the same values.

If $\mathcal{A}$ wins, then

$$T_1 = \mathbf{cdh}(B - kg^{x_1}, A) = \mathbf{cdh}(B', A),$$
$$T_2 = \mathbf{cdh}(B - kg^{x_2}, A) = \mathbf{cdh}(B' + \Delta, A),$$

so $\mathcal{A}'$ also wins. We have

$$\mathbf{Adv}_{\mathcal{A}'} \geq \mathbf{Adv}_{\mathcal{A}},$$

and since the aSDH problem is hard, $\mathbf{Adv}_{\mathcal{A}'}$ is negligible, which implies that $\mathbf{Adv}_{\mathcal{A}}$ is also negligible. This shows that the Two-for-One Guessing problem is hard.

"⇐": For any PPT adversary $\mathcal{A}$ in the aSDH game, we construct an adversary $\mathcal{A}'$ in the Two-for-One Guessing game:

1. On k, x_1, x_2 from its challenger, $\mathcal{A}'$ computes $\Delta := k \cdot (g^{x_1} - g^{x_2})$ and invokes $\mathcal{A}$ on Δ.

[8] Here we can see why in the definition of Two-for-One Guessing we require $x_1 \neq x_2$: otherwise k might not be well-defined, and we would need to rule out the negligible probability that $x_1 = x_2$. Ruling out this possibility in the security game yields a cleaner proof.

2. When $\mathcal{A}$ outputs B, $\mathcal{A}'$ computes $B' := B + kg^{x_1}$ and outputs B'.

3. On A from its challenger, $\mathcal{A}'$ sends the same value to $\mathcal{A}$.

4. When $\mathcal{A}$ outputs T_1, T_2, $\mathcal{A}'$ outputs the same values.

If $\mathcal{A}$ wins, then

$$T_1 = \mathbf{cdh}(B, A) = \mathbf{cdh}(B' - kg^{x_1}, A),$$
$$T_2 = \mathbf{cdh}(B + \Delta, A) = \mathbf{cdh}(B' - kg^{x_2}, A),$$

so $\mathcal{A}'$ also wins. We have

$$\mathbf{Adv}_{\mathcal{A}'} \geq \mathbf{Adv}_{\mathcal{A}},$$

and since the Two-for-One-Guessing problem is hard, $\mathbf{Adv}_{\mathcal{A}'}$ is negligible, which implies that $\mathbf{Adv}_{\mathcal{A}}$ is also negligible. This shows that the aSDH problem is hard. □

aSDH is easier than CDH. Finally, it is not hard to see that the aSDH problem is easier than the CDH problem:

Theorem 4. *If the aSDH problem is hard in $\mathbb{F}_p$, then the CDH problem is hard in $\mathbb{F}_p$.*

Proof. For any PPT adversary $\mathcal{A}$ in the CDH game, we construct an adversary $\mathcal{A}'$ in the aSDH game:

1. On Δ from its challenger, $\mathcal{A}'$ samples $b \leftarrow \mathbb{Z}_{p-1}$ and outputs $B := g^b$.

2. On A from its challenger, $\mathcal{A}'$ invokes $\mathcal{A}$ on $(A, B + \Delta)$.

3. When $\mathcal{A}$ outputs C, $\mathcal{A}'$ computes $T_1 := A^b$ and outputs (T_1, C).

Note that A and Δ are independently random in $\mathbb{F}_p^*$, so $B+\Delta$ is also random in $\mathbb{F}_p^*$ and independent of A, i.e., $\mathcal{A}$'s input is a correctly formed CDH challenge. If $\mathcal{A}$ wins, then

$$C = \mathbf{cdh}(A, B + \Delta),$$

and furthermore $T_1 = A^b = \mathbf{cdh}(A, B)$, so $\mathcal{A}'$ also wins. We have

$$\mathbf{Adv}_{\mathcal{A}'} \geq \mathbf{Adv}_{\mathcal{A}},$$

and since the aSDH problem is hard, $\mathbf{Adv}_{\mathcal{A}'}$ is negligible, which implies that $\mathbf{Adv}_{\mathcal{A}}$ is also negligible. This shows that the CDH problem is hard. □

3.3 Choose DDH Check (CDC)

Definition 6 (Choose DDH Check with RO (CDC-RO))**.** *The **CDC-RO** game is defined as follows:*

1. SIM *outputs* $(B, k, x_1, \cdots, x_q) \in \mathbb{F}_p \times \mathbb{F}_p^* \times \mathbb{Z}_{p-1}^q$. *The distribution of* $(B, k, x_1, \cdots, x_q)$ *must be indistinguishable from uniform.*

2. *The game challenger outputs* $(A, T) \in (\mathbb{F}_p^*)^2$, *which follows any (efficiently samplable) distribution.*

3. SIM *outputs an index* $i \in [0, q]$ *(or aborts), and wins if the output is correct. The correct output is defined as follows: if there is a single* $i \in [q]$ *such that* $\mathbf{ddh}(B - kg^{x_i}, Ag^{ux_i}, T) = 1$, *then the correct output is* i*; if there is no such* i, *then the correct output is* 0*; if there are multiple such* i, *then there is no correct output. Here,* $u = H(A, B)$ *where* $H : \mathbb{F}_p^* \times \mathbb{F}_p \to \mathbb{Z}_{p-1}$ *is an RO simulated by* SIM.

We say the CDC-RO problem is very easy in $\mathbb{F}_p$ *if there is a PPT* SIM *such that for any PPT game challenger, the probability that* SIM *wins the above game is overwhelming.*

Roughly speaking, the challenger corresponds to the UC environment in SRP and SIM corresponds to the UC simulator (hence its name); the game itself is roughly an abstraction of the password extraction process in SRP, where the simulator, given an adversarial protocol message (A, T) and candidate password guesses $x_1, \cdots, x_q$, must decide which one is the "correct" password guess (if any).

Simplification of CDC-RO. We now show that CDC-RO can be simplified to the following form, where the g^{ux_i} term is removed:

Definition 7 (Choose DDH Check (CDC))**.** *The* ***CDC*** *problem is the same as the CDC-RO problem, except that in step 3, the criterion for determining the correct input* $\mathbf{ddh}(B - kg^{x_i}, Ag^{ux_i}, T) = 1$ *is replaced by* $\mathbf{ddh}(B - kg^{x_i}, A, T) = 1$. *(The RO* H *is not used anymore.)*

Theorem 5. *The CDC problem is very easy in* $\mathbb{F}_p$ *if and only if the CDC-RO problem is very easy in* $\mathbb{F}_p$.

Proof. "⇒": Since CDC is very easy, there exists a PPT adversary SIM that wins the CDC game with probability $\mathbf{Adv}_{\mathsf{SIM}} = 1 - \varepsilon$, where ε is negligible. We construct a $\mathsf{SIM}_{\mathrm{RO}}$ in the CDC-RO game:

0. $\mathsf{SIM}_{\mathrm{RO}}$ answers H queries (either by itself or by SIM) via lazy sampling.

1. $\mathsf{SIM}_{\mathrm{RO}}$ fixes SIM's random coins and invokes SIM q times. When SIM outputs $(B, k, x_1, \cdots, x_q)$[9], $\mathsf{SIM}_{\mathrm{RO}}$ outputs the same values.

[9]Since SIM's random coins are fixed, its q invocations will result in the same $(B, k, x_1, \cdots, x_q)$ values.

2. On (A, T) from its challenger, $\mathsf{SIM}_{\mathrm{RO}}$ computes $u := H(A, B)$ and runs SIM on $(A, T \cdot (B - kg^{x_i})^{-ux_i})$ for $i \in [q]$.

3. $\mathsf{SIM}_{\mathrm{RO}}$ examines SIM's outputs in its q executions. If there is a single $i \in [q]$ such that SIM outputs i in its i-th execution, then $\mathsf{SIM}_{\mathrm{RO}}$ outputs i. If there are multiple such i, then $\mathsf{SIM}_{\mathrm{RO}}$ outputs $\mathsf{MultiIndices}$ and aborts. If there is no such i, $\mathsf{SIM}_{\mathrm{RO}}$ outputs 0.

Since SIM wins the CDC game with probability $1 - \varepsilon$, it wins *all q times* with probability at least $1 - q\varepsilon$. Below we assume SIM wins all q times, and analyze $\mathsf{SIM}_{\mathrm{RO}}$'s winning probability.

First, we argue that $\Pr[\mathsf{MultiIndices}]$ is negligible. $\mathsf{MultiIndices}$ happens if there exist $i \neq j$ such that

$$T = (B - kg^{x_i})^{a+ux_i} = (B - kg^{x_j})^{a+ux_j}.$$

For any fixed tuple (A, x_i, x_j) where $x_i \neq x_j$, let $a = \mathbf{dl}(A)$. $\mathsf{MultiIndices}$ implies (taking discrete log on both sides)

$$(a + ux_i)\mathbf{dl}(B - kg^{x_i}) = (a + ux_j)\mathbf{dl}(B - kg^{x_j}),$$

or equivalently,

$$u(x_i\mathbf{dl}(B - kg^{x_i}) - x_j\mathbf{dl}(B - kg^{x_j})) + a(\mathbf{dl}(B - kg^{x_i}) - \mathbf{dl}(B - kg^{x_j})) = 0.$$

There are two possibilities:

- If this is a linear equation of u, then the probability that u happens to be its solution is $1/(p-1)$.
- If this is not a linear equation of u, i.e., the coefficient of u is 0, then the equation holds only if $a(\mathbf{dl}(B - kg^{x_i}) - \mathbf{dl}(B - kg^{x_j})) = 0$. But this is impossible, since $a \neq 0$ and $\mathbf{dl}(B - kg^{x_i}) \neq \mathbf{dl}(B - kg^{x_j})$.

We conclude that the probability of $\mathsf{MultiIndices}$ is up to $1/(p-1)$ for a fixed tuple of (A, x_i, x_j). Assuming there are ℓ queries to H, there are $\binom{q}{2}\ell$ candidate tuples of (A, x_i, x_j), so

$$\Pr[\mathsf{MultiIndices}] \leq \binom{q}{2} \cdot \frac{\ell}{p-1}.$$

Next, assume $\mathsf{MultiIndices}$ does not happen, i.e., there is at most one i such that $T = (B - kg^{x_i})^{a+ux_i}$. Note that this is equivalent to $\mathbf{ddh}(B - kg^{x_i}, Ag^{ux_i}, T) = 1$, which is exactly $\mathsf{SIM}_{\mathrm{RO}}$'s winning condition. If there is a single such i, then $\mathsf{SIM}_{\mathrm{RO}}$ outputs i, so $\mathsf{SIM}_{\mathrm{RO}}$ wins. If there is no such i, then $\mathsf{SIM}_{\mathrm{RO}}$ outputs 0, so $\mathsf{SIM}_{\mathrm{RO}}$ also wins.

We conclude that if SIM wins all q times and the event $\mathsf{MultiIndices}$ does not happen, then $\mathsf{SIM}_{\mathrm{RO}}$ also wins. So

$$\mathbf{Adv}_{\mathsf{SIM}_{\mathrm{RO}}} \geq (1 - q\varepsilon)\left(1 - \binom{q}{2} \cdot \frac{\ell}{p-1}\right),$$

which is overwhelming. Therefore, CDC-RO is very easy.

"⇐": Since CDC-RO is very easy, there exists a PPT adversary $\mathsf{SIM}_{\mathrm{RO}}$ that wins the CDC-RO game with probability $1-\varepsilon$, where ε is negligible. We construct a SIM in the CDC game:

1. SIM fixes $\mathsf{SIM}_{\mathrm{RO}}$'s random coins and invokes $\mathsf{SIM}_{\mathrm{RO}}$ q times. When $\mathsf{SIM}_{\mathrm{RO}}$ outputs $(B, k, x_1, \cdots, x_q)$, SIM outputs the same values.
2. On (A, T) from its challenger, SIM computes $u := H(A, B)$ and runs $\mathsf{SIM}_{\mathrm{RO}}$ on $(A, T \cdot (B - kg^{x_i})^{ux_i})$ for $i \in [q]$.
3. SIM examines $\mathsf{SIM}_{\mathrm{RO}}$'s outputs in its q executions. If there is a single $i \in [q]$ such that $\mathsf{SIM}_{\mathrm{RO}}$ outputs i in its i-th execution, then SIM outputs i. If there are multiple such i, then SIM outputs $\mathsf{MultiIndices}$ and aborts. If there is no such i, SIM outputs 0.

Since $\mathsf{SIM}_{\mathrm{RO}}$ wins the CDC-RO game with probability $1-\varepsilon$, it wins *all* q *times* with probability at least $1-q\varepsilon$. Below we assume $\mathsf{SIM}_{\mathrm{RO}}$ wins all q times, and analyze SIM's winning probability.

$\mathsf{MultiIndices}$ happens if there exist $i \neq j$ such that

$$T = (B - kg^{x_i})^a = (B - kg^{x_j})^a,$$

which is impossible. Then there is at most one i such that $T = (B - kg^{x_i})^a$, i.e., $\mathbf{ddh}(B - kg^{x_i}, A, T) = 1$, which is exactly SIM's winning condition. If there is a single such i, then SIM outputs i, so SIM wins. If there is no such i, then SIM outputs 0, so SIM also wins.

We conclude that if $\mathsf{SIM}_{\mathrm{RO}}$ wins all q times, then SIM also wins. So

$$\mathbf{Adv}_{\mathsf{SIM}} \geq 1 - q\varepsilon,$$

which is overwhelming. Therefore, CDC-RO is very easy. □

The multi-session version. The CDC problem as defined above is not good for use in SRP, as SRP is a multi-session protocol where all sessions share the same parameters $(k, x_1, \cdots, x_q)$ but have their own B, and the simulator has to extract the adversary's password guess in each session. Therefore, we need the following multi-session version of CDC:

Definition 8 (ℓ-session CDC-RO)**.** *The* ℓ***-session CDC-RO*** *game is defined as follows:*

1. SIM *outputs* $(B_1, \cdots, B_\ell, k, x_1, \cdots, x_q) \in \mathbb{F}_p^\ell \times \mathbb{F}_p^* \times \mathbb{Z}_{p-1}^q$. *The distribution of* $(B_1, \cdots, B_\ell, k, x_1, \cdots, x_q)$ *must be indistinguishable from uniform. Then for each* $sid \in [\ell]$ *(and in any order), the game proceeds as follows:*

2. *The game challenger outputs* (sid, A_{sid}, T_{sid}) *for* $(A_{sid}, T_{sid}) \in (\mathbb{F}_p^*)^2$, *which follows any (efficiently samplable) distribution. (Below we assume sid is implicit and write A for* A_{sid}, *and so on.)*

3. SIM *outputs* (sid, i) *for* $i \in [0, q]$ *(or aborts), and loses if the output is incorrect. The correct output is defined as follows: if there is a single* $i \in [q]$ *such that* $\mathbf{ddh}(B - kg^{x_i}, Ag^{ux_i}, T) = 1$, *then the correct output is i; if there is no such i, then the correct output is* 0*; if there are multiple such i, then there is no correct output. Here,* $u = H(A, B)$ *where* $H : \mathbb{F}_p^* \times \mathbb{F}_p \to \mathbb{Z}_{p-1}$ *is an RO simulated by* SIM.

4. *When steps 2 and 3 are done for all* $sid \in [\ell]$, *if* SIM *has not lost, then* SIM *wins.*

We say the ℓ-session CDC-RO problem is very easy in $\mathbb{F}_p$ *if there is a PPT* SIM *such that for any PPT game challenger, the probability that* SIM *wins the above game is overwhelming.*

Definition 9 (ℓ-session CDC)**.** *The* ***ℓ-session CDC*** *problem is the same as the ℓ-session CDC-RO problem, except that in step 3, the criterion for determining the correct input* $\mathbf{ddh}(B - kg^{x_i}, Ag^{ux_i}, T) = 1$ *is replaced by* $\mathbf{ddh}(B - kg^{x_i}, A, T) = 1$.

The following theorem is the multi-session version of Theorem 5. The proof is also similar and is omitted:

Theorem 6. *For any ℓ, the ℓ-session problem CDC is very easy in* $\mathbb{F}_p$ *if and only if the ℓ-session CDC-RO problem is very easy in* $\mathbb{F}_p$.

Simplifying Multi-session CDC to Multi-message CDC. It turns out that the Multi-session CDC problem can be simplified to the following form, with only one session but multiple B values:

Definition 10 (ℓ-message CDC)**.** *The* ***ℓ-message CDC*** *game is defined as follows:*

1. SIM *outputs* $(B_1, \cdots, B_\ell, k, x_1, \cdots, x_q) \in \mathbb{F}_p^\ell \times \mathbb{F}_p^* \times \mathbb{Z}_{p-1}^q$. *The distribution of* $(B_1, \cdots, B_\ell, k, x_1, \cdots, x_q)$ *must be indistinguishable from uniform.*

2. *The game challenger outputs* $(j, A, T) \in [\ell] \times (\mathbb{F}_p^*)^2$, *which follows any (efficiently samplable) distribution.*

3. SIM *outputs an index* $i \in [0, q]$ *(or aborts), and wins if the output is correct. The correct output is defined as follows: if there is a single* $i \in [q]$ *such that* $\mathbf{ddh}(B_j - kg^{x_i}, A, T) = 1$*, then the correct output is* i*; if no such* i *exists, then the correct output is* 0*; if there are multiple* i *such that* $\mathbf{ddh}(B_j - kg^{x_i}, A, T) = 1$*, then there is no correct output.*

We say the ℓ-message CDC problem is very easy in $\mathbb{F}_p$ if there is a PPT SIM such that for any PPT game challenger, the probability that SIM wins the above game is overwhelming.

Theorem 7. *For any ℓ, the ℓ-message CDC problem is very easy in $\mathbb{F}_p$ if and only if the ℓ-session CDC problem is very easy in $\mathbb{F}_p$.*

Proof. "$\Rightarrow$": Since ℓ-message CDC is very easy, there exists a PPT adversary SIM that wins the ℓ-message CDC game with probability $1 - \varepsilon$, where ε is negligible. We construct a SIM' in the ℓ-session CDC game:

1. SIM' fixes SIM's random coins and invokes SIM q times. When SIM outputs $(B_1, \cdots, B_\ell, k, x_1, \cdots, x_q)$, SIM' outputs the same values.
2. For each $sid \in [\ell]$, on (sid, A_{sid}, T_{sid}) from its challenger, SIM' runs SIM on (sid, A_{sid}, T_{sid}).
3. When SIM outputs i for (sid, A_{sid}, T_{sid}), SIM' outputs (sid, i).

It is straightforward to see that if SIM wins all q times, then SIM' also wins. So

$$\mathbf{Adv}_{\mathsf{SIM}'} \geq 1 - q\varepsilon,$$

which is overwhelming. Therefore, ℓ-session CDC is very easy.

"$\Leftarrow$": This is straightforward. □

4 UC-Security of SRP

In this section, we show that the SRP protocol is UC-secure assuming the hardness of aSDH and the very easiness of CDC. Our main theorem is:

Theorem 8. *The SRP protocol UC-realizes $\mathcal{F}_{aPAKE}$ if the following assumptions hold in $\mathbb{F}_p$:*

1. *The aSDH problem is hard;*
2. *The ℓ_{SS}-message CDC problem is very easy, where ℓ_{SS} is the number of sub-sessions.*

By Theorems 2, 3 and 6, we may instead assume that the q-for-One Guessing problem is hard (for some appropriate q that we will define later), and the ℓ_{SS}-session CDC-RO problem is very easy. By Theorem 4 we may additionally assume that the CDH problem is hard.

The rest of this section is dedicated to the proof of Theorem 8. In Section 4.1 we show the simulator, and in Section 4.2 we show that our simulator generates a view indistinguishable from the real world. Comparison with the proof in [DL24] is shown in red.

4.1 The Simulator

We present our simulator SIM for the SRP protocol in Figures 3 and 4. SIM interacts with the ideal functionality $\mathcal{F}_{\text{aPAKE}}$ and simulates the real-world protocol execution for any environment $\mathcal{Z}$. Following standard UC framework conventions, we assume a dummy adversary $\mathcal{A}$ that is totally controlled by $\mathcal{Z}$ and simply forwards messages between $\mathcal{Z}$ and the protocol parties.

Comments on the simulator. In step 6(b), SIM needs to find x_i such that $\mathbf{ddh}(B - kg^{x_i}, A^* g^{u_S x_i}, T_S) = 1$. This is the only step in the entire proof where we require multi-session CDC to be very easy. To fit the exact syntax of ℓ_{SS}-session CDC-RO (Definition 8), SIM should invoke the ℓ_{SS}-session CDC-RO solver at the beginning of its simulation, and use the solver's output as k and $x_1, \cdots, x_q$. Then in each sub-session, in step 4, SIM would use the solver's output as B, and in step 6(b), SIM would send (A^*, T_S) to the solver and let the solver find x_i. Our description above is a simplified and slightly informal version where SIM solves ℓ_{SS}-session CDC-RO on its own. In particular, since the ℓ_{SS}-session CDC-RO game requires $(k, B, x_1, \cdots, x_q)$ to be pseudorandom anyway, we let SIM sample them uniformly at random.

By contrast, in step 8(a) SIM can check if $\mathbf{ddh}(B^* - kv, Av^{u_C}, T) = 1$ without solving multi-session CDC: SIM knows $\log A = a$ and $\log v = x$, so the condition is equivalent to $T = (B^* - kv)^{a + u_C x}$. Similarly, SIM can check if $\mathbf{ddh}(B^* - kg^{x_j}, Ag^{u_C x_j}, T) = 1$ in step 8(b).

The simulator in [DL24] handles the M_2^{S*} message slightly differently: it does not distinguish cases 7(a) and 7(c), and sends $(\texttt{NewKey}, sid, ssid, C, \perp)$ to $\mathcal{F}$ (without $(\texttt{TestPwd}, sid, ssid, C, \perp)$) in both cases. This is problematic due to the following. Suppose C and S have the same password, $\mathcal{A}$ forwards the first two messages $(s^*, B^*) = (s, B)$ and $(A^*, M_1^{C*}) = (A, M_1^C)$, but modifies the last message $M_2^{S*} \neq M_2^S$. Then in the real world C will output $\perp$. In the ideal world, since $\mathcal{A}$ forwards the first two messages, SIM *previously entered step 6(a) and no H_3 value is defined; thus,* SIM *will not enter step 7(b) (in particular, no* `TestPwd` *command will be sent). Once* SIM *enters step 7(c), C's session record remains* `fresh`*, so $\mathcal{F}$ will let C output the same session key as S. To fix this,* SIM *must send a "dummy" $(\texttt{TestPwd}, sid, ssid, C, \perp)$ command, so that C's session will be* `interrupted`*, causing C to output $\perp$.*

Password file registration

Compute $k := H_5(p, g)$.
On the first message (no matter the message name) that involves sid, sample $s \leftarrow \{0,1\}^\lambda$ and store $\langle \texttt{PWfile}, sid, s, \perp, \perp \rangle$.

1. On $(\texttt{StealPwdFile}, sid, S)$ from $\mathcal{A}$ to S, send $(\texttt{StealPwdFile}, sid, S)$ to $\mathcal{F}$.
   (a) If $\mathcal{F}$ returns "no password file", then send "no password file" to $\mathcal{A}$.
   (b) Else, if $\mathcal{F}$ returns pw:
      (i) If there is a record $\langle \texttt{PWfile}, sid, s, x, v \rangle$ and $H_1(s, \mathsf{pw})$ has been defined as x, then compute $v := g^x$.
      (ii) Else (i.e., if there is a record $\langle \texttt{PWfile}, sid, s, \perp, \perp \rangle$), sample $x \leftarrow \mathbb{Z}_{p-1}$, define $H_1(s, \mathsf{pw}) := x$, compute $v := g^x$, and change the record to $\langle \texttt{PWfile}, sid, s, x, v \rangle$.

      Either way, send (s, v) to $\mathcal{A}$.
   (c) Else (i.e., if $\mathcal{F}$ returns nothing), sample $x \leftarrow \mathbb{Z}_{p-1}$, compute $v := g^x$, and change the record $\langle \texttt{PWfile}, sid, s, \perp, \perp \rangle$ to $\langle \texttt{PWfile}, sid, s, x, v \rangle$. Then send (s, v) to $\mathcal{A}$.

   In all cases above, if v is generated so that $B = kv$, abort.

Offline attacks

2. On (fresh) $H_1(s, \mathsf{pw}^*)$ query from $\mathcal{A}$, send $(\texttt{OfflineTestPwd}, sid, S, \mathsf{pw}^*)$ to the $\mathcal{F}$.
   (a) If $\mathcal{F}$ returns "correct guess", then there must have been a `StealPwdFile` message, so there is a $\langle \texttt{PWfile}, sid, s, x, v \rangle$ record. Program $H_1(s, \mathsf{pw}^*) := x$.
   (b) If $\mathcal{F}$ returns "wrong guess" or nothing, then sample $H_1(s, \mathsf{pw}^*) \leftarrow \mathbb{Z}_p^*$.

   Finally, send $H_1(s, \mathsf{pw}^*)$ to $\mathcal{A}$.

RO queries

3. Answer all $H_3, \cdots, H_6$ queries from either $\mathcal{A}$ or SIM via lazy sampling. (We will describe how H_2 queries are answered later.) If there is a collision in H_1, or a collision in H_2, H_3, H_6 with the same $ssid$, abort.

Initial protocol messages

4. On $(\texttt{SvrSession}, sid, ssid, S, C)$ from $\mathcal{F}$, retrieve record $\langle \texttt{PWfile}, sid, s, \star, \star \rangle$ and sample $B \leftarrow \mathbb{F}_p$.
   (a) If the record is $\langle \texttt{PWfile}, sid, s, x, v \rangle$ (note that this implies that there has been a `StealPwdFile` message), and $B = kv$, abort.
   (b) Else, send (s, B) from S to C.
5. On $(\texttt{CltSession}, sid, ssid, C, S)$ from $\mathcal{F}$ and (s^*, B^*) from $\mathcal{A}$ to C, sample $a \leftarrow \mathbb{Z}_{p-1}$ and $M_1^C \leftarrow \mathbb{F}_p$, compute $A := g^a$, and send (A, M_1^C) from C to S.

Figure 3: Simulator SIM for SRP (part 1)

Active session attacks

6. On (A^*, M_1^{C*}) from $\mathcal{A}$ to S,
   (a) If $(s^*, B^*) = (s, B) \wedge (A^*, M_1^{C*}) = (A, M_1^C)$, then send $(\texttt{NewKey}, ssid, C, \perp)$ to $\mathcal{F}$.
      (i) If $\mathcal{F}$ returns $(ssid, \perp)$, then send $M_2^S := \perp$ from S to C. // C and S's passwords differ
      (ii) Else, send $M_2^S \leftarrow \mathbb{F}_p$ from S to C.
   (b) Else, look for T_S such that $H_2(A^*, B, T_S)$ has been defined as M_1^{C*}.
      (i) If there is no such T_S, send $(\texttt{TestPwd}, sid, ssid, S, \perp)$ and then $(\texttt{NewKey}, sid, ssid, S, \perp)$ to $\mathcal{F}$. Send $M_2^S := \perp$ from S to C.

   Then compute $u_S := H_6(A^*, B)$, and among all $x_i = H_1(s, \mathsf{pw}_i)$, find the one that satisfies $\mathbf{ddh}(B - kg^{x_i}, A^* g^{u_S x_i}, T_S) = 1$.
      (ii) If there are multiple such x_i, output MultiIndicesSvr and abort.
      (iii) If there is a single such x_i, send $(\texttt{TestPwd}, sid, ssid, S, \mathsf{pw}_i)$ to $\mathcal{F}$.
      If $\mathcal{F}$ returns "correct guess", compute $K_S := H_4(T_S)$ and send $(\texttt{NewKey}, sid, ssid, S, K_S)$ to $\mathcal{F}$. Send $M_2^S := H_3(A^*, M_1^{S*}, T_S)$ from S to C.
      If $\mathcal{F}$ returns "wrong guess", send $(\texttt{NewKey}, sid, ssid, S, \perp)$ to $\mathcal{F}$. Send $M_2^S := \perp$ from S to C.
      (iv) If there is no such x_i, send $(\texttt{TestPwd}, sid, ssid, S, \perp)$ and then $(\texttt{NewKey}, sid, ssid, S, \perp)$ to $\mathcal{F}$. Send $M_2^S := \perp$ from S to C.

7. On M_2^{S*} from $\mathcal{A}$ to C,
   (a) If $(s^*, B^*) = (s, B) \wedge (A^*, M_1^{C*}) = (A, M_1^C) \wedge M_2^{S*} = M_2^S$, then send $(\texttt{NewKey}, sid, ssid, C, \perp)$ to $\mathcal{F}$. // $\mathcal{A}$ passive
   (b) Else, if there is a T_C such that $H_3(A, M_1^C, T_C) = M_2^{S*}$ and $H_2(A, B^*, T_C) = M_1^C$, then compute $K_C := H_4(T_C)$ and send $(\texttt{NewKey}, sid, ssid, C, K_C)$ to $\mathcal{F}$. // This may require SIM to query $H_2(A, B^*, T_C)$, which is described later
   (c) Else, send $(\texttt{TestPwd}, sid, ssid, C, \perp)$ and then $(\texttt{NewKey}, sid, ssid, C, \perp)$ to $\mathcal{F}$.

8. On (fresh) $H_2(A', B', T)$ query from either $\mathcal{A}$ or SIM, if $A' = A \wedge B' = B^*$ does not hold or $(s, B) = (s^*, B^*)$, answer with lazy sampling. Else (so the query is $H_2(A, B^*, T)$), compute $u_C := H_6(A, B^*)$ and:
   (a) If there is a record $\langle \texttt{PWfile}, sid, s, x, v \rangle$, $s^* = s$, and $\mathbf{ddh}(B^* - kv, Av^{u_C}, T) = 1$, then send $(\texttt{Impersonate}, sid, ssid, C, S)$ to $\mathcal{F}$.
      (i) If $\mathcal{F}$ returns "correct guess", program $H_2(A, B^*, T) := M_1^C$.
      (ii) If $\mathcal{F}$ returns "wrong guess", sample $H_2(A, B^*, T) \leftarrow \mathbb{F}_p \setminus \{M_1^C\}$.
   (b) Else, for all x_i which is the result of an $H_1(s^*, \mathsf{pw}_i)$ query, check if $\mathbf{ddh}(B^* - kg^{x_i}, Ag^{u_C x_i}, T) = 1$.
      (i) If there are multiple such x_i, or if a $(\texttt{TestPwd}, sid, ssid, C, \star)$ command has already been sent to $\mathcal{F}$ in some previous H_2 query, output MultiIndicesClt and abort. // two-for-one guessing attack
      (ii) Else, if there is a single such x_i, send $(\texttt{TestPwd}, sid, ssid, C, \mathsf{pw}_i)$ to $\mathcal{F}$.
      If $\mathcal{F}$ returns "correct guess", program $H_2(A, B^*, T) := M_1^C$.
      If $\mathcal{F}$ returns "wrong guess", sample $H_2(A, B^*, T) \leftarrow \mathbb{F}_p \setminus \{M_1^C\}$.
      (iii) If there is no such x_i, sample $H_2(A, B^*, T) \leftarrow \mathbb{F}_p \setminus \{M_1^C\}$.

   Finally, send $H_2(A, B^*, T)$ to $\mathcal{A}$.

Figure 4: Simulator SIM for SRP (part 2). See comments below on how boxed operations can be performed

4.2 Hybrid Argument

We now show that the simulator SIM generates a view of $\mathcal{Z}$ indistinguishable from $\mathcal{Z}$'s real view.

Hybrid 0: This is the real world, where the adversary $\mathcal{A}$ interacts with the SRP protocol parties C and S. We assume $\mathcal{A}$'s protocol messages have a star after them: for example, the first S-to-C message is (s, B) and $\mathcal{A}$ may modify it to (s^*, B^*). See Figure 5 for the real world with $\mathcal{A}$.

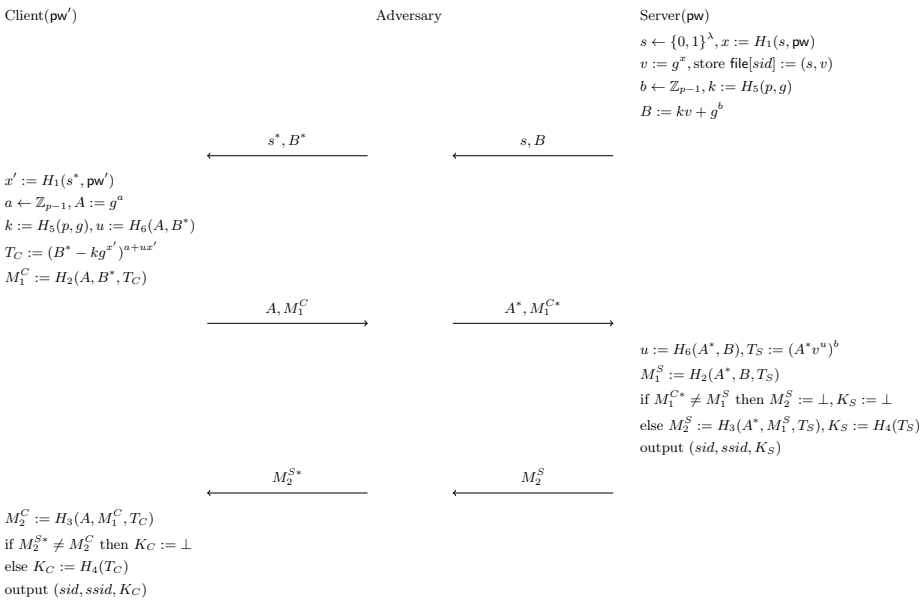

Figure 5: SRP protocol — real world with adversary

Hybrid 1 (exclude RO collisions): If there is a collision in H_1, or a collision in H_2, H_3, H_6 with the same *ssid*, the game aborts.

We use ℓ_{H_i} to denote the number of queries to H_i by $\mathcal{A}$. By the birthday bound, we have

$$\mathbf{Dist}_{\mathcal{Z}}^{0,1} \leq \frac{(\ell_{H_1}+2)^2}{2(p-1)} + \frac{(\ell_{H_2}+2)^2}{2p} + \frac{(\ell_{H_3}+2)^2}{2p} + \frac{(\ell_{H_6}+2)^2}{2(p-1)},$$

where the "+2" is due to honest parties' queries to these ROs.

This hybrid is Game G_2 in [DL24], where they exclude all RO collisions. In fact it is unnecessary to exclude H_5 collisions, since the protocol only uses a single H_5 value: $k = H_5(p, g)$. Close scrutiny shows that we also don't need to exclude H_4 collisions.

Hybrid 2 (random protocol messages and session keys for same password and passive $\mathcal{A}$): In the case of $\mathsf{pw} = \mathsf{pw}'$, we modify the game as follows:

1. If $(s^*, B^*) = (s, B)$, sample $M_1^C \leftarrow \mathbb{F}_p$;
2. If $(s^*, B^*) = (s, B) \wedge (A^*, M_1^{C*}) = (A, M_1^C)$, sample $M_2^S \leftarrow \mathbb{F}_p$ and $K_S \leftarrow \{0,1\}^\lambda$;
3. If $(s^*, B^*) = (s, B) \wedge (A^*, M_1^{C*}) = (A, M_1^C) \wedge M_2^{S*} = M_2^S$ (i.e., $\mathcal{A}$ is passive), set $K_C := K_S$.

Recall that in hybrid 1, the relevant values are defined as

$$A := g^a,$$
$$M_1^C := H_2(A, B, T), M_2^S := H_3(A, M_1^C, T),$$
$$K_C := K_S := H_4(T),$$

where $T = T_C = T_S = \mathbf{cdh}(B - kg^x, Ag^{ux})$ for $x = H_1(s, \mathsf{pw})$ and $u = H_6(A, B)$. Let Query be the event that $\mathcal{A}$ queries $H_2(\star, \star, T)$, $H_3(\star, \star, T)$, or $H_4(T)$; clearly hybrids 1 and 2 are identical unless and until Query happens. We now reduce Query to the CDH problem.

The reduction $\mathcal{R}_{\text{CDH1}}$, given input (A', B'), samples two random indices $i \leftarrow [\ell_{SS}]$ (where ℓ_{SS} is the number of sub-sessions) and $j \leftarrow [\ell_{H_2} + \ell_{H_3} + \ell_{H_4}]$ as a guess that in the i-th sub-session, $\mathcal{A}$'s j-th query to H_2, H_3 or H_4 triggers Query.[10] Then $\mathcal{R}_{\text{CDH1}}$ invokes the environment $\mathcal{Z}$ and runs the code of hybrid 2, except that in the i-th sub-session, $\mathcal{R}_{\text{CDH1}}$ sets $A := A'$ and $B := B' + kg^x$. Upon receiving $\mathcal{A}$'s j-th query to H_2, H_3 or H_4 (say the query is $H_2(\star, \star, T)$, $H_3(\star, \star, T)$, or $H_4(T)$), $\mathcal{R}_{\text{CDH1}}$ outputs $T' := T \cdot (B')^{-ux}$ and halts.

Assume that Query happens, and $\mathcal{R}_{\text{CDH1}}$'s guess is correct. Then it is straightforward to see that $\mathcal{R}_{\text{CDH1}}$ simulates hybrid 2 (which is the same as hybrid 1) unless and until Query happens. Furthermore, by the definition of A and B we have

$$T = \mathbf{cdh}(B - kg^x, Ag^{ux}) = \mathbf{cdh}(B', A'g^{ux}),$$

so $\mathbf{cdh}(B', A') = T \cdot (B')^{-ux} = T'$, i.e., $\mathcal{R}_{\text{CDH1}}$ wins the CDH game. We conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{1,2} \leq \ell_{SS}(\ell_{H_2} + \ell_{H_3} + \ell_{H_4})\mathbf{Adv}^{\text{CDH}}_{\mathcal{R}_{\text{CDH1}}}.$$

The security proof in [DL24] does not describe how their reduction "knows" in which sub-session A and B should be computed using the CDH challenge A' and B'. In fact the reduction must make a guess over all sub-sessions, losing a factor of ℓ_{SS}. The same issue appears in all subsequent reductions.

Hybrid 3 (reject for different password and passive $\mathcal{A}$): In the case of $\mathsf{pw} \neq \mathsf{pw}'$, we modify the game as follows:

1. If $(s^*, B^*) = (s, B)$, sample $M_1^C \leftarrow \mathbb{F}_p$;

[10] $\ell_{H_2} + \ell_{H_3} + \ell_{H_4}$ can be replaced by $\max_{i \in [\ell_{SS}]}(\ell_{H_2^i} + \ell_{H_3^i} + \ell_{H_4^i})$, where $\ell_{H_2^i}$ is the number of H_2 queries in the i-th sub-session, and so on.

2. If $(s^*, B^*) = (s, B) \wedge (A^*, M_1^{C*}) = (A, M_1^C)$, set $M_2^S := \perp$ and $K_S := \perp$;

3. If $(s^*, B^*) = (s, B) \wedge (A^*, M_1^{C*}) = (A, M_1^C) \wedge M_2^{S*} = M_2^S$ (i.e., $\mathcal{A}$ is passive), set $K_C := \perp$.

First, we make the change in item (2) (without any other change), i.e., we argue that if $\mathsf{pw} \neq \mathsf{pw}'$ and $(s^*, B^*) = (s, B) \wedge (A^*, M_1^{C*}) = (A, M_1^C)$, then $M_2^S = \perp$ and $K_S = \perp$ with overwhelming probability in hybrid 2. Since $\mathsf{pw} \neq \mathsf{pw}'$ and we have excluded collisions in H_1, $x \neq x'$. We claim that

$$T_C = \mathbf{cdh}(B - kg^{x'}, Ag^{ux'}) \text{ and } T_S = \mathbf{cdh}(B - kg^{x}, Ag^{ux})$$

are different with overwhelming probability. Otherwise $T_C = T_S$, which implies that (taking discrete log on both sides)

$$(a + ux)\mathbf{dl}(B - kg^x) = (a + ux')\mathbf{dl}(B - kg^{x'}),$$

or equivalently,

$$a(\mathbf{dl}(B - kg^x) - \mathbf{dl}(B - kg^{x'})) + (ux\mathbf{dl}(B - kg^x) - ux'\mathbf{dl}(B - kg^{x'})) = 0,$$

which is a linear equation of a. Since a is uniformly random, the probability that a happens to be the solution of the equation is $1/(p-1)$. We conclude that the probability that $T_C = T_S$ is $1/(p-1)$ for a single sub-session; thus, $\Pr[T_C = T_S] \leq \ell_{SS}/(p-1)$.

If $T_C \neq T_S$, then $M_1^C \neq M_1^S$ (since we have excluded collisions in H_2) and S will set $M_2^S = \perp$ and $K_S = \perp$. So the distinguishing advantage between hybrids 2 and 3 in item (2) is no greater than $\ell_{SS}/(p-1)$.

Next, we make the change in item (3). This clearly makes no external difference, since C will set $K_C = \perp$ as long as $M_2^S = \perp$.

The argument above is missing in [DL24], which is not entirely trivial.

We finally make the change in item (1). The reduction to the CDH problem, $\mathcal{R}_{\text{CDH2}}$, is the same as the reduction under hybrid 2, except that $\mathcal{R}_{\text{CDH2}}$ runs the code of hybrid 3 and only needs to observe $\mathcal{A}$'s H_2 queries. The argument why $\mathcal{R}_{\text{CDH2}}$ "works" is identical to the argument under hybrid 2. We conclude that

$$\mathbf{Dist}_{\mathcal{Z}}^{2,3} \leq \ell_{SS}\left(\ell_{H_2} \cdot \mathbf{Adv}^{\text{CDH}}_{\mathcal{R}_{\text{CDH2}}} + \frac{1}{p-1}\right).$$

Hybrids 2 and 3 combined correspond to Game G_3 in [DL24]. We believe having two hybrids and two separate reductions to CDH makes the argument easier to follow.

Hybrid 4 (random client message and matching H_2): If $(s^*, B^*) \neq (s, B)$, the game samples $M_1^C \leftarrow \mathbb{F}_p$ (note that if $(s^*, B^*) = (s, B)$, this change was already made in hybrids 2 and 3). Furthermore, we modify the behavior of H_2 oracle to match the new definition of M_1^C: C does not query $H_2(A, B^*, T_C)$ anymore, and later when $\mathcal{A}$ or S queries $H_2(A, B^*, T_C)$, the game programs $H_2(A, B^*, T_C) :=$

M_1^C. However, if $\mathcal{A}$ queries $H_2(A, B^*, T_C)$ before seeing M_1^C, the game outputs PredictA and aborts.

In hybrid 3, C sets $M_1^C := H_2(A, B^*, T_C)$. In hybrid 4, we sample $M_1^C \leftarrow \mathbb{F}_p$ first, and later when $H_2(A, B^*, T_C)$ is queried, we program the result as M_1^C. Clearly this makes no difference. The only remaining case is PredictA. Note that A is sent together with M_1^C, so in this case A is uniformly random in $\mathcal{A}$'s view when $\mathcal{A}$ makes the H_2 query. But then the probability that $\mathcal{A}$ can make an $H_2(A, \star, \star)$ query, i.e., $\mathcal{A}$ guesses A correctly, is $1/(p-1)$ per query and $\ell_{H_2}/(p-1)$ overall.[11] We have

$$\mathbf{Dist}_{\mathcal{Z}}^{3,4} \leq \Pr[\mathsf{PredictA}] \leq \frac{\ell_{H_2}}{p-1}.$$

Hybrid 5 (change client's session key as in SIM*):* On M_2^{S*} from $\mathcal{A}$ to C, if $\mathcal{A}$ is passive, the game defines K_C as in hybrids 2 and 3. Otherwise, we define K_C as follows: If there exists a T such that $H_3(A, M_1^C, T) = M_2^{S*}$, the game makes an $H_2(A, B^*, T)$ query (if there has been no such query). If the result is M_1^C and $T = T_C$, the game sets $K_C := H_4(T)$; otherwise, the game sets $K_C := \perp$.

Suppose $\mathcal{A}$ queries $H_3(A, M_1^C, T)$ and sends the result M_2^{S*} to C. Then

- If $T = T_C$: then in hybrid 4, C's check $H_3(A, M_1^C, T_C) \stackrel{?}{=} M_2^{S*}$ passes; in hybrid 5, by the definition of H_2, the game programs $H_2(A, B^*, T) := M_1^C$, so C's check also passes.
- If $T \neq T_C$: then in hybrid 4, C's check $H_3(A, M_1^C, T_C) \stackrel{?}{=} M_2^{S*}$ fails; and the check in hybrid 5 obviously fails.

We conclude that whether K_C is $H_4(T_C)$ or $\perp$ is exactly the same in hybrids 4 and 5. The only remaining case is that $\mathcal{A}$ guesses $M_2^{S*} = H_3(A, M_1^C, T)$ without making an $H_3(A, M_1^C, T)$ query. The probability of this happening is $1/p$ per sub-session and ℓ_{SS}/p in total. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{4,5} \leq \frac{\ell_{SS}}{p}.$$

Hybrids 4 and 5 combined correspond to Game G_8 in [DL24]. (See hybrid 6 below for why we move it earlier.) [DL24] merely claims that the probability of the "bad event" is negligible, without calculating the exact distinguishing advantage.

Hybrid 6 (rule out two-for-one guessing attack): If $\mathcal{A}$ makes two queries $H_2(A, B^*, T_1)$, $H_2(A, B^*, T_2)$ (T_1 and T_2 may be equal or different) such that

$$T_1 = \mathbf{cdh}(Ag^{u_C x_i}, B^* - kg^{x_i}), T_2 = \mathbf{cdh}(Ag^{u_C x_j}, B^* - kg^{x_j}),$$

[11] Note that the number of sub-sessions does not matter here: although there is an independent random A in each sub-session, while making H_2 queries $\mathcal{A}$ must specify the sub-session ID, i.e., for each $H_2(ssid, A', \star, \star)$ query $\mathcal{A}$ triggers PredictA only if the A *in the specific sub-session* $ssid$ is equal to A', which gives $\mathcal{A}$ a $1/(p-1)$ advantage per query.

where $u_C = H_6(A, B^*)$ and $x_i \neq x_j$ are two $H_1(s^*, \star)$ query outputs by $\mathcal{A}$, the game outputs MultiIndicesClt and aborts.

We reduce MultiIndicesClt to the $(\ell_{H_1} + 1)$-for-One Guessing problem via a reduction $\mathcal{R}_{(\ell_{H_1}+1)\text{-for-1}}$. Below we simply let $\ell = \ell_{H_1}+1$. At a high level, $\mathcal{R}_{\ell\text{-for-1}}$ uses its challenge to embed the protocol message A, and then the "bad event" MultiIndicesClt becomes the winning condition of ℓ-for-One Guessing (with some gaps that can be addressed by $\mathcal{R}_{\ell\text{-for-1}}$). However, this means that $\mathcal{R}_{\ell\text{-for-1}}$ does not know a, which is needed for computing T_C in C's algorithm! We carefully design our reduction (see step 7 below for details) so that the game can still be simulated without knowledge of a (with some advantage loss).

1. On input $(k, x_1, \cdots, x_{\ell_{H_1}+1})$, $\mathcal{R}_{\ell\text{-for-1}}$ sets $H_5(p, g) := k$ and invokes $\mathcal{Z}$. $\mathcal{R}_{\ell\text{-for-1}}$ uses $x_1, \cdots, x_{\ell_{H_1}+1}$ to answer H_1 queries (by $\mathcal{A}$ or by $\mathcal{R}_{\ell\text{-for-1}}$ itself).

2. When S registers its password file, $\mathcal{R}_{\ell\text{-for-1}}$ samples $s \leftarrow \{0,1\}^\lambda$ and computes $v := g^{H_1(s,\mathsf{pw})}$.

3. $\mathcal{R}_{\ell\text{-for-1}}$ samples an index $m \leftarrow [\ell_{SS}]$ (as a guess that MultiIndicesClt happens in the m-th sub-session). In all other sub-sessions, $\mathcal{R}_{\ell\text{-for-1}}$ simply runs the code of hybrid 4. In the m-th sub-session, when S's sub-session begins, $\mathcal{R}_{\ell\text{-for-1}}$ samples $b \leftarrow \mathbb{Z}_{p-1}$, computes $B := kv + g^b$, and sends (s, B) to $\mathcal{A}$.

4. On $\mathcal{A}$'s message (s^*, B^*) to C, $\mathcal{R}_{\ell\text{-for-1}}$ outputs B^* to its challenger and receives A. $\mathcal{R}_{\ell\text{-for-1}}$ then samples $M_1^C \leftarrow \mathbb{F}_p$ and sends (A, M_1^C) to $\mathcal{A}$ as C's response.

5. On $\mathcal{A}$'s message (A^*, M_1^{C*}) to S, $\mathcal{R}_{\ell\text{-for-1}}$ computes M_2^S, K_S as in hybrid 4, lets S output K_S, and sends M_2^S to $\mathcal{A}$.

6. On $\mathcal{A}$'s message M_2^{S*} to C, $\mathcal{R}_{\ell\text{-for-1}}$ lets C output K_C, defined as follows:

   (a) If $\mathsf{pw} = \mathsf{pw}'$ and $\mathcal{A}$ is passive, $\mathcal{R}_{\ell\text{-for-1}}$ sets $K_C := K_S$.

   (b) If $\mathsf{pw} \neq \mathsf{pw}'$ and $\mathcal{A}$ is passive, $\mathcal{R}_{\ell\text{-for-1}}$ sets $K_C := \perp$.

   (c) Else, if there is an $H_3(A, M_1^C, T)$ query whose result is M_2^{S*}, $\mathcal{R}_{\ell\text{-for-1}}$ makes an $H_2(A, B^*, T)$ query. If the result is M_1^C, $\mathcal{R}_{\ell\text{-for-1}}$ sets $K_C := H_4(T)$.

   (d) Else, $\mathcal{R}_{\ell\text{-for-1}}$ sets $K_C := \perp$.

7. For RO queries, we have described how to answer H_1 and H_5 queries in step 1. $\mathcal{R}_{\ell\text{-for-1}}$ answers H_3, H_4, H_6 queries via lazy sampling (but excluding collisions).
   For $H_2(A', B', T)$ queries, $\mathcal{R}_{\ell\text{-for-1}}$ also answers via lazy sampling unless $A' = A \wedge B' = B^*$. The most difficult case is $H_2(A, B^*, T)$ queries. In hybrid 4, the game checks if $T = T_C$, and if so, it programs $H_2(A, B^*, T) := M_1^C$ (and lazy samples if not). But computing T_C requires knowledge of

a, which $\mathcal{R}_{\ell\text{-for-1}}$ does not know since it used its challenge to embed A (in step 4)!
To get around this, first recall that there are ℓ_{H_2} H_2 queries by $\mathcal{A}$ and up to one H_2 query by $\mathcal{R}_{\ell\text{-for-1}}$ (in step 6(c)). $\mathcal{R}_{\ell\text{-for-1}}$ samples an index $h \leftarrow [0, \ell_{H_2}+1]$ as a guess that the h-th H_2 query is $H_2(A, B^*, T_C)$ (where 0 denotes no $H_2(A, B^*, T_C)$ query). If $h \neq 0$ then $\mathcal{R}_{\ell\text{-for-1}}$ programs the result of the h-th H_2 query as M_1^C, and answers all other H_2 queries via lazy sampling; if $h = 0$ then $\mathcal{R}_{\ell\text{-for-1}}$ answers all H_2 queries via lazy sampling.[12]

8. After $\mathcal{Z}$ finishes, $\mathcal{R}_{\ell\text{-for-1}}$ randomly picks two of $\mathcal{A}$'s $H_2(A, B^*, T)$ queries and rename T there as T_1, T_2. T_1, T_2 may be the same or different.
Furthermore, $\mathcal{R}_{\ell\text{-for-1}}$ samples $i < j \leftarrow [\ell_{H_1}]$, computes $u_C := H_6(A, B^*)$ and $T_1' = T_1 \cdot (B^* - kg^{x_i})^{-u_C x_i}, T_2' = T_2 \cdot (B^* - kg^{x_j})^{-u_C x_j}$, and outputs $(i, T_1'), (j, T_2')$ to its challenger.

Assume $\mathcal{R}_{\ell\text{-for-1}}$'s guesses are all correct; in particular, if $h \neq 0$ then the h-th H_2 query is indeed $H_2(A, B^*, T_C)$, and if $h = 0$ then there is indeed no $H_2(A, B^*, T_C)$ query. Then as long as there is an $H_2(A, B^*, T_C)$ query, the result is programmed as M_1^C in step 7, which matches hybrid 5. Furthermore, we claim that $\mathcal{R}_{\ell\text{-for-1}}$ also simulates hybrid 5 correctly in step 6:

- Step 6(a) matches hybrid 2;
- Step 6(b) matches hybrid 3;
- Otherwise hybrid 5 requires the game to find an $H_3(A, M_1^C, T)$ query whose result is M_2^{S*}, make an $H_2(A, B^*, T)$ query, and set $K_C := H_4(T)$ if the result is M_1^C and $T = T_C$. This matches step 6(c)(d), except that the $T = T_C$ check is omitted. $\mathcal{R}_{\ell\text{-for-1}}$ can safely skip the $T = T_C$ check since if $H_2(A, B^*, T) = M_1^C$, it is guaranteed that $T = T_C$.

We conclude that $\mathcal{R}_{\ell\text{-for-1}}$ simulates hybrid 5 correctly.

Next, we claim that $\mathcal{R}_{\ell\text{-for-1}}$ wins the ℓ-for-One Guessing game as long as $\mathsf{MultiIndicesClt}$ happens and $\mathcal{R}_{\ell\text{-for-1}}$'s guesses are all correct. If so, then T_1, T_2 indeed trigger $\mathsf{MultiIndicesClt}$, and the corresponding H_1 query results are indeed x_i and x_j, i.e.,

$$T_1 = \mathbf{cdh}(Ag^{u_C x_i}, B^* - kg^{x_i}), T_2 = \mathbf{cdh}(Ag^{u_C x_j}, B^* - kg^{x_j}),$$

then by the definitions of T_1', T_2' in step 8, we have

$$T_1' = \mathbf{cdh}(A, B^* - kg^{x_i}), T_2' = \mathbf{cdh}(A, B^* - kg^{x_j}),$$

so $\mathcal{R}_{\ell\text{-for-1}}$ indeed wins the ℓ-for-One Guessing game.

[12]Note that in this way $\mathcal{R}_{\ell\text{-for-1}}$ never needs to compute T_C, which in turn means that $x' = H_1(s^*, \mathsf{pw}')$ also doesn't need to be computed. So, the only H_1 query $\mathcal{R}_{\ell\text{-for-1}}$ makes is the one in step 2. This is why we can reduce to $(\ell_{H_1}+1)$-for-One Guessing, rather than $(\ell_{H_1}+2)$-for-One Guessing.

$\mathcal{R}_{\ell\text{-for-1}}$ guesses $m \leftarrow [\ell_{SS}]$ in step 3, $h \leftarrow [0, \ell_{H_2}+1]$ in step 6, two H_2 queries in step 8, and $i < j \leftarrow [\ell_{H_1}]$ in step 8. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{5,6} \leq \ell_{SS}\binom{\ell_{H_1}}{2}\ell_{H_2}^2(\ell_{H_2}+2)\mathbf{Adv}_{\mathcal{R}_{\ell\text{-for-1}}}^{\ell\text{-for-One Guessing}}.$$

This hybrid corresponds to Game G_6 in [DL24]. Note that our reduction always samples M_1^C at random (in step 4); this goes through because we already changed M_1^C to random in hybrid 4. If we put hybrid 4 after hybrid 6, then the reduction must compute M_1^C as $H_2(A, B^*, T_C)$, which cannot be done because the reduction cannot compute T_C. This is exactly the mistake of [DL24]: their Game G_8 (which corresponds to our hybrid 4) comes after Game G_6, causing the reduction not being able to simulate the game.

One might ask why [DL24] does not notice this issue. The reason is that they do not describe how their reduction simulates the game at all, except that the reduction uses its challenge to embed A. As such, they overlook the fact that the "secret information" a, which is not known to the reduction, is used while computing T_C.

On the tightness of the reduction. The concrete security bound in hybrid 6 looks extremely bad, especially since ℓ_{SS}, ℓ_{H_1} and ℓ_{H_2} are all likely to be large. If we reduce to the Gap ℓ-for-One Guessing (or equivalently, Gap aSDH) problem instead, then the $\binom{\ell_{H_1}}{2}\ell_{H_2}^2(\ell_{H_2}+2)$ factor can be removed, resulting in a reasonable security bound: in step 7 the reduction can scan over all H_2 queries and use the DDH oracle to check which query input is T_C, and in step 8 the reduction can use the DDH oracle to check which two of the $H_2(A, B^*, T)$ queries and which two x_i, x_j values trigger $\mathsf{MultiIndicesClt}$. This is the approach taken in [DL24]; however, (apart from the mistake above) [DL24] overlooks the guess over all sub-sessions.

Hybrid 7 (impersonation and password guessing detection): In this game, we further change the behavior of H_2 when the query input is $H_2(A, B^*, T)$, where A and B^* are as defined in the game, and T is an arbitrary element of $\mathbb{F}_p$. On (fresh) $H_2(A, B^*, T)$ query, the game computes $u_C := H_6(A, B^*)$ and does the following:

1. If S has been compromised (recall that S's password file is (s, v)), $s^* = s$, and $\mathbf{ddh}(B^* - kv, Av^{u_C}, T) = 1$ (or equivalently, $T = \mathbf{cdh}(B^* - kv, Av^{u_C})$), then
   (a) If $\mathsf{pw} = \mathsf{pw}'$, program $H_2(A, B^*, T) := M_1^C$.
   (b) If $\mathsf{pw} \neq \mathsf{pw}'$, sample $H_2(A, B^*, T) \leftarrow \mathbb{F}_p \setminus \{M_1^C\}$.
2. Else, for all x_i which is the result of an $H_1(s^*, \mathsf{pw}_i)$ query by $\mathcal{A}$, check if $\mathbf{ddh}(B^* - kg^{x_i}, Ag^{u_C x_i}, T) = 1$ (or equivalently, $T = \mathbf{cdh}(B^* - kg^{x_i}, Ag^{u_C x_i})$).
   (a) If there are multiple such x_i, or if there was a previous $H_2(A, B^*, T')$ query whose output was M_1^C, output $\mathsf{MultipleIndicesClt}$ and abort.

(b) Else, if there is a single such x_i, and the corresponding $\mathsf{pw}_i = \mathsf{pw}'$, program $H_2(A, B^*, T) := M_1^C$.

(c) Else, sample $H_2(A, B^*, T) \leftarrow \mathbb{F}_p \setminus \{M_1^C\}$.

It is easy to see that hybrids 6 and 7 have no difference regarding the definition of the "bad event" MultipleIndicesClt. Below we assume that MultipleIndicesClt does not happen.

In hybrid 6, we answer H_2 queries via lazy sampling, except that $H_2(A, B^*, T_C)$ is programmed as M_1^C. In hybrid 7, we instead program $H_2(A, B^*, T)$ as M_1^C in cases 1(a) and 2(b) and lazy sample otherwise. Therefore, we only need to show that $T = T_C$ in hybrid 7 if and only if cases 1(a) or 2(b) happens (except with negligible probability). Consider any fixed $H_2(A, B^*, T)$ query:

1. In case 1, we have $s^* = s$ and $T = \mathbf{cdh}(B^* - kv, Av^{u_C}) = \mathbf{cdh}(B^* - kg^x, Ag^{u_C x})$.

   (a) If $\mathsf{pw} = \mathsf{pw}'$, then $x = H_1(s, \mathsf{pw}) = H_1(s^*, \mathsf{pw}') = x'$, so $T = \mathbf{cdh}(B^* - kg^{x'}, Ag^{u_C x'}) = T_C$.

   (b) If $\mathsf{pw} \neq \mathsf{pw}'$, then $x \neq x'$, so by the same argument under hybrid 3, $T \neq \mathbf{cdh}(B^* - kg^{x'}, Ag^{u_C x'}) = T_C$ except with probability $1/(p-1)$. (Note that in this case we exclude M_1^C while sampling $H_2(A, B^*, T)$, so that $H_2(A, B^*, T)$ and $H_2(A, B^*, T_C)$ do not form a collision.)

2. In case 2,

   (b) If there is a single pw_i such that $T = \mathbf{cdh}(B^* - kg^{x_i}, Ag^{u_C x_i})$ (where $x_i = H_1(s^*, \mathsf{pw}_i)$) and $\mathsf{pw}' = \mathsf{pw}_i$, then $x' = x_i$, so $T = \mathbf{cdh}(B^* - kg^{x'}, Ag^{u_C x'}) = T_C$.

   (c) Else, either (i) there exists some pw_i such that $T = \mathbf{cdh}(B^* - kg^{x_i}, Ag^{u_C x_i})$ but $\mathsf{pw}' \neq \mathsf{pw}_i$ (and $x' \neq x_i$), so by the same argument under hybrid 3, $T \neq \mathbf{cdh}(B^* - kg^{x'}, Ag^{u_C x'}) = T_C$ except with probability $1/(p-1)$ for any fixed pw_i, so the overall probability that there exists a pw_i such that the corresponding $T = T_C$ is $\ell_{H_1}/(p-1)$; or (ii) there is no such pw_i, i.e., $\mathcal{A}$ does not query $H_1(s^*, \mathsf{pw}')$ but computes $T = T_C$. In this case x' is uniformly random in $\mathcal{A}$'s view, so the probability that $\mathcal{A}$ guesses T_C correctly is again $1/(p-1)$.

We conclude that

$$\text{hybrid 7 programs } H_2(A, B^*, T) \text{ as } M_1^C \Leftrightarrow \text{case 1(a) or 2(b)} \Leftrightarrow T = T_C$$

(except with probability up to $\ell_{H_1}/(p-1)$ for a single H_2 query in cases 1(b) and 2(c), and the overall probability is $\ell_{H_1}\ell_{H_2}/(p-1)$), which matches the behavior of hybrid 6. We get

$$\mathbf{Dist}_{\mathcal{Z}}^{6,7} \leq \frac{\ell_{H_1}\ell_{H_2}}{p-1}.$$

This hybrid appears missing in [DL24].

Hybrid 8 (rule out $\mathcal{A}$ sending correct M_1^{C*} without RO query): On message (A^*, M_1^{C*}) from $\mathcal{A}$ to S with $(s^*, B^*) \neq (s, B) \vee (A^*, M_1^{C*}) \neq (A, M_1^C)$ (i.e., $\mathcal{A}$ is not passive in the first two messages), the game looks for a T such that $H_2(A^*, B, T) = M_1^{C*}$. If no such T exists, the game sets $M_2^S :=\perp$ and $K_S :=\perp$.

The only way for $\mathcal{A}$ to distinguish between hybrids 7 and 8 is to guess the value of $H_2(A^*, B, T)$ without making this query, whose probability is $1/p$ in each sub-session. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{7,8} \leq \frac{\ell_{SS}}{p}.$$

This hybrid appears missing in [DL24]. Note that this is not covered by hybrid 9 below; a separate statistical argument must be performed, which [DL24] does not do.

Hybrid 9 (rule out client impersonation without successful offline attack): In this hybrid we continue considering the scenario when $\mathcal{A}$ sends (A^*, M_1^{C*}) to S and $\mathcal{A}$ is not passive in the first two messages. If $\mathcal{A}$ never queried $H_1(s, \mathsf{pw})$ before, the game sets $M_2^S := \perp$ and $K_S := \perp$.

Recall that S computes $T_S := (A^* v^{u_S})^b = \mathbf{cdh}(B - kv, A^* v^{u_S})$ and checks if $M_1^{C*} = H_2(A^*, B, T_S)$. Let $\mathsf{CltImpersonation}$ be the event that $\mathcal{A}$ does not query $H_1(s, \mathsf{pw})$ but queries $H_2(A^*, B, T_S)$ and sends the result to S as M_1^{C*}. Clearly, hybrids 8 and 9 are identical unless $\mathsf{CltImpersonation}$ happens. We reduce $\mathsf{CltImpersonation}$ to 2ODH using the forking lemma (Theorem 1).

Why is forking needed? Let us first explain at a high level why we need the forking lemma here. $\mathsf{CltImpersonation}$ amounts to $\mathcal{A}$ solving CDH on instance $(B - kv, A^* v^{u_S})$. The problem is that the reduction cannot simply embed its CDH challenge (B', A') as $(B - kv, A^* v^{u_S})$, since $u_S = H_6(A, B)$ — which requires the reduction to program $H_6(A^*, B)$ as $\mathbf{dl}_v(A'/A^*)$, which it cannot compute. Instead, the reduction has to embed its CDH challenge as $(B - kv, v)$; but then the reduction cannot compute $\mathbf{cdh}(B - kv, v)$ even if $\mathcal{A}$ queries H_2 on $\mathbf{cdh}(B - kv, A^* v^{u_S})$, since A^* is controlled by $\mathcal{A}$ and the reduction doesn't know $\mathbf{dl}(A^*)$. However, if $\mathcal{A}$ queries H_2 twice on $\mathbf{cdh}(B - kv, A^* v^{u_S})$ for different values of u_S — say they are $T = \mathbf{cdh}(B - kv, A^* v^u)$ and $T' = \mathbf{cdh}(B - kv, A^* v^{u'})$ — then the reduction can compute $\mathbf{cdh}(B - kv, v)$ as $(T/T')^{\frac{1}{u-u'}}$. We now proceed to the formal argument.

The wrapper. First, we define a wrapper $\mathcal{W}$ that embeds its challenge (X, B') as $(v, B - kv)$:

1. Let $\ell = \ell_{H_6} + 2$. On inputs (X, B') and $(u_1, \cdots, u_\ell)$, $\mathcal{W}$ samples $i \leftarrow [\ell_{SS}]$ (as a guess that $\mathsf{CltImpersonation}$ happens in the i-th sub-session). Then $\mathcal{W}$ samples $s \leftarrow \{0,1\}^\lambda$ and sets $v := X$. In all sub-sessions but the i-th, $\mathcal{W}$ simply runs the code of hybrid 9. Below we only describe $\mathcal{W}$'s behavior in the i-th sub-session.

2. $\mathcal{W}$ computes $B := B' + kv$ and sends (s, B) to $\mathcal{A}$.

3. On $\mathcal{A}$'s message (s^*, B^*) to C, $\mathcal{W}$ samples $A \leftarrow \mathbb{F}_p^*$ and $M_1^C \leftarrow \mathbb{F}_p$, and sends (A, M_1^C) to $\mathcal{A}$.

4. On $\mathcal{A}$'s message (A^*, M_1^{C*}) to S, if $(s^*, B^*) = (s, B) \wedge (A^*, M_1^{C*}) = (A, M_1^C)$, then $\mathcal{W}$ knows that CltImpersonation does not happen in this sub-session (i.e., its guess is wrong), so it outputs $(0, \perp)$ and halts. Otherwise $\mathcal{W}$ sets $M_2^S := \perp$ and $K_S := \perp$, sends M_2^S to $\mathcal{A}$, and lets S output K_S.

5. On $\mathcal{A}$'s message M_2^{S*} to C, $\mathcal{W}$ computes K_C as in hybrid 5 and lets C output K_C.

6. For RO queries, $\mathcal{W}$ answers H_3, H_4, H_5 queries via lazy sampling. If $\mathcal{A}$ queries $H_1(s, \mathsf{pw})$, then $\mathcal{W}$ knows that CltImpersonation does not happen, so it outputs $(0, \perp)$ and halts. $\mathcal{W}$ answers all other H_1 queries via lazy sampling.
   $\mathcal{W}$ answers H_6 queries for the i-th sub-session (by $\mathcal{A}$ or by $\mathcal{W}$ itself) using its inputs $u_1, \cdots, u_\ell$,[13] and identifies the smallest index I such that the I-th H_6 query is $H_6(A^*, B)$.
   $H_2(A, B^*, T)$ queries pose an issue: $\mathcal{W}$ is supposed to answer these queries by running the code in hybrid 7. In case 1 there, $\mathcal{W}$ needs to check if $\mathbf{ddh}(B^* - kv, Av^{u_C}, T) = 1$. The code in hybrid 7 can check this efficiently, because it knows $\mathbf{dl}(v) = H_1(s, \mathsf{pw})$; whereas $\mathcal{W}$ embeds its challenge X as v, so it doesn't know $\mathbf{dl}(v)$ and cannot complete this check!
   To get around this, we have to resort to the same trick as in hybrid 6: $\mathcal{W}$ samples an index $h \leftarrow [0, \ell_{H_2} + 1]$ as a guess that the h-th H_2 query is $H_2(A, B^*, T_C)$ (where 0 denotes no $H_2(A, B^*, T_C)$ query). If $h \neq 0$ then $\mathcal{W}$ programs the result of the h-th H_2 query as M_1^C, and answers all other $H_2(A, B^*, T)$ queries via lazy sampling; if $h = 0$ then $\mathcal{W}$ answers all $H_2(A, B^*, T)$ queries via lazy sampling. (Additional care must be taken while answering H_2 queries; see step 7 and the subsequent paragraph below.)

7. When $\mathcal{A}$ finishes, $\mathcal{W}$ checks if $\mathcal{A}$ triggers CltImpersonation or not, and outputs (I, σ) or $(0, \perp)$ accordingly. This must be done very carefully, and is the reason why we have to reduce to 2ODH rather than CDH; we describe how $\mathcal{W}$ can achieve this in the next paragraph.

How does the wrapper determine its output? To begin with, $\mathcal{W}$ must check if $\mathcal{A}$ triggers CltImpersonation or not; in particular, if CltImpersonation does not happen, then $\mathcal{W}$ must not accept, i.e., $\mathcal{W}$ must not introduce "false positives" (otherwise the reduction $\mathcal{R}$ might not win even if the forker $\mathcal{F}^{\mathcal{W}}$ accepts, which makes it difficult to analyze $\mathcal{R}$'s winning probability). However, CltImpersonation happens if $\mathcal{A}$ queries $H_2(A^*, B, T_S)$ for $T_S = \mathbf{cdh}(B -$

[13] Note that $\mathcal{W}$ might make two H_6 queries itself, one on the C side and one on the S side. That's why we set $\ell = \ell_{H_6} + 2$.

$kv, A^* v^{u_S}) = \mathbf{cdh}(B', A^* X^{u_S})$, where X and B' are $\mathcal{W}$'s challenge; as such, $\mathcal{W}$ cannot perform this check!

We get around this issue by reducing to 2ODH rather than CDH, which gives $\mathcal{W}$ access to two oracles:

- $\mathcal{H} : \mathbb{F}_p^* \to \{0,1\}^\lambda$, an RO (with collisions excluded);
- $\mathcal{O} : \mathbb{F}_p^* \to \{0,1\}^\lambda$, which on input M returns $\mathcal{H}(M^{b'})$ for $b' = \mathbf{dl}(B')$. $\mathcal{O}$ can be queried only once.

(Note that these two oracles are exactly what's in the 2ODH game in Definition 2.)

$\mathcal{W}$ uses the following method to determine its output:

- $\mathcal{W}$ answers $H_2(A^*, B, T)$ queries with $\mathcal{H}(T)$. (Other H_2 queries are answered as in step 6 above or via lazy sampling.)
- When $\mathcal{A}$ finishes, $\mathcal{W}$ looks for a T such that $H_2(A^*, B, T) = M_1^{C*}$. If there is no such T, then $\mathsf{CltImpersonation}$ does not happen, so $\mathcal{W}$ outputs $(0, \perp)$. Otherwise $\mathcal{W}$ queries $\mathcal{O}(A^* X^{u_S})$. If the result is equal to M_1^{C*}, i.e.,

  $$M_1^{C*} = H_2(A^*, B, T) = \mathcal{H}(T) \text{ and } \mathcal{O}(A^* X^{u_S}) = \mathcal{H}((A^* X^{u_S})^{b'})$$

  are equal, then $\mathcal{W}$ knows that $T = \mathbf{cdh}(B', A^* X^{u_S})$, i.e., $\mathsf{CltImpersonation}$ happens. In this case $\mathcal{W}$ outputs $(I, (T, u_I))$. Otherwise $\mathcal{W}$ outputs $(0, \perp)$.

If $\mathsf{CltImpersonation}$ happens and $\mathcal{W}$'s guesses are correct, then $\mathcal{W}$ outputs $(I, (T, u_I))$. We have

$$\Pr[\mathsf{CltImpersonation}] \leq \ell_{SS}(\ell_{H_2} + 2) \cdot \mathbf{Acc}_{\mathcal{W}}.$$

The forker. The wrapper $\mathcal{W}$ immediately yields the forker $\mathcal{F}^{\mathcal{W}}$ (see Theorem 1 for the definition of $\mathcal{F}^{\mathcal{W}}$), whose accepting probability satisfies

$$\mathbf{Acc}_{\mathcal{W}} \leq \sqrt{\ell \cdot \mathbf{Frk}_{\mathcal{F}^{\mathcal{W}}}} + \frac{\ell}{p},$$

so

$$\Pr[\mathsf{CltImpersonation}] \leq \ell_{SS}(\ell_{H_2} + 2) \left(\sqrt{\ell \cdot \mathbf{Frk}_{\mathcal{F}^{\mathcal{W}}}} + \frac{\ell}{p} \right).$$

The reduction. Finally, we construct the reduction $\mathcal{R}_{\text{2ODH}}$ to 2ODH:

1. On input (X, B'), $\mathcal{R}_{\text{2ODH}}$ invokes $\mathcal{F}^{\mathcal{W}}$ on the same input. $\mathcal{R}_{\text{2ODH}}$ answers $\mathcal{F}^{\mathcal{W}}$'s $\mathcal{H}$ and $\mathcal{O}$ queries by querying its own oracles. ($\mathcal{F}^{\mathcal{W}}$ runs $\mathcal{W}$ twice, and in each run $\mathcal{W}$ makes at most one query to $\mathcal{O}$; therefore, $\mathcal{R}_{\text{2ODH}}$ makes up to two queries to $\mathcal{O}$.)
2. On $\mathcal{F}^{\mathcal{W}}$'s output $(b, \star, \star)$, if $b = 0$, $\mathcal{R}_{\text{2ODH}}$ aborts.

3. Else, $\mathcal{R}_{\text{2ODH}}$ parses $\mathcal{F}^{\mathcal{W}}$'s output as $(1,(T,u_I),(T',u'_I))$. Note that the definition of $\mathcal{F}^{\mathcal{W}}$ guarantees that $u_I \neq u'_I$ as long as $\mathcal{F}^{\mathcal{W}}$ accepts. $\mathcal{R}_{\text{2ODH}}$ outputs $(T/T')^{\frac{1}{u_I-u'_I}}$.

We claim that if $\mathcal{F}^{\mathcal{W}}$ accepts, then $\mathcal{R}_{\text{2ODH}}$ solves the 2ODH challenge. Suppose $\mathcal{F}^{\mathcal{W}}$ accepts; by the definition of $\mathcal{F}^{\mathcal{W}}$, $\mathcal{W}$ accepts in both runs. We have designed $\mathcal{W}$ so that if $\mathcal{W}$ accepts, it is guaranteed that CltImpersonation happens, and $\mathcal{W}$'s output (T,u_I) satisfies $T = \mathbf{cdh}(B-kv, A^*u_I)$. Note that A^* and B in $\mathcal{W}$'s two runs must be the same, since $\mathcal{W}$'s random coins and inputs up to u_I are fixed; thus, the two runs are identical until after the I-th H_6 query, whose input is (A^*,B). This means that $\mathcal{W}$'s two runs yield two $\mathbf{cdh}(B-kv, A^*u_I)$ values for the same A^* and B but different u_I, which implies

$$\frac{T}{T'} = \frac{\mathbf{cdh}(B-kv, A^*v^{u_I})}{\mathbf{cdh}(B-kv, A^*v^{u'_I})} = \frac{(A^*v^{u_I})^{\mathbf{dl}(B-kv)}}{(A^*v^{u'_I})^{\mathbf{dl}(B-kv)}} = v^{(u_I-u'_I)\mathbf{dl}(B-kv)},$$

so $\mathcal{R}_{\text{2ODH}}$'s output

$$\left(\frac{T}{T'}\right)^{\frac{1}{u_I-u'_I}} = v^{\mathbf{dl}(B-kv)} = \mathbf{cdh}(v, B-kv).$$

Since $\mathcal{W}$ embeds its challenge (X,B') as $(v,B-kv)$, $\mathcal{R}_{\text{2ODH}}$'s output is correct, i.e., $\mathcal{R}_{\text{2ODH}}$ wins the 2ODH game. We have

$$\mathbf{Frk}_{\mathcal{F}^{\mathcal{W}}} = \mathbf{Adv}^{\text{2ODH}}_{\mathcal{R}_{\text{2ODH}}}.$$

Combining the above, we conclude that (plugging in $\ell = \ell_{H_6}+2$)

$$\mathbf{Dist}^{8,9}_{\mathcal{Z}} \leq \Pr[\mathsf{CltImpersonation}] \leq \ell_{SS}(\ell_{H_2}+2)\left(\sqrt{(\ell_{H_6}+2)\mathbf{Adv}^{\text{2ODH}}_{\mathcal{R}_{\text{2ODH}}}} + \frac{\ell_{H_6}+2}{p}\right).$$

This hybrid corresponds to Game G_5 in [DL24]. The argument there contains multiple gaps:

1. *It defines the "bad event" slightly differently, only considering the case that S is compromised. In fact, the possibility that $\mathcal{A}$ does not compromise S but sends a valid M_1^{C*} must be ruled out as well, either via a separate statistical argument or simply combined with the "compromising S" case. This case is overlooked in [DL24].*

2. *It overlooks the guess among all sub-sessions, missing the ℓ_{SS} factor.*

3. *It completely misses the details of the forking lemma. [DL24] says*

   > *Assume that there is an adversary who can compute $T_1 = \mathbf{cdh}_g(Av^{u_1}, B-kv)$ for $u_1 = H_6(A,B)$. By the Forking Lemma [BN06], the adversary on the same random tape can also compute $T_2 = \mathbf{cdh}_g(Av^{u_2}, B-kv)$ for different random oracle output $u_2 = H_6(A,B)$.*

This is not what the Bellare–Neven forking lemma states; rather, the forking lemma merely relates the accepting probabilities of the wrapper and the forker, neither of which is the adversary. As such, the first step of applying the forking lemma is to construct a wrapper and a forker, which [DL24] does not mention at all.

4. *As we have seen in our proof, the construction of the wrapper and the forker is not merely some "boilerplate" work; rather, the wrapper must check if the "bad event" happens or not, which requires interesting proof techniques that are missing in [DL24].*[14]

5. *The misuse of the forking lemma also causes the concrete security bound to be far from the correct one. Since the adversary is run twice, it is clear that the reduction's advantage should be roughly proportional to the square of the probability of the "bad event". However, this quadratic loss is not present in [DL24]! Instead, their security bound is*

$$\ell_{H_2}^2 \cdot \mathbf{Adv}_{\mathcal{R}}^{\mathrm{CDH}},$$

which severely underestimates the adversary's advantage.

Hybrid 10 (use CDC solver to check client-to-server message): Since ℓ_{SS}-session CDC-RO is easy, there is a PPT algorithm SIM' that wins that game with overwhelming probability. In this hybrid, we make the following changes to hybrid 9:

1. The game invokes SIM' at the very beginning (setting $q = \ell_{H_1} + 1$), and uses SIM''s output as k and $x_1, \ldots, x_{\ell_{H_1}+1}$. Then in the j-th sub-session, the game proceeds as follows:

2. The game does not sample b, and instead uses SIM''s output B_j as B. However, if $B_j = kv$, the game outputs $\mathsf{NoValidb}$ and aborts.

3. When $\mathcal{A}$ sends (A^*, M_1^{C*}) to S, if $(s^*, B^*) = (s, B) \wedge (A^*, M_1^{C*}) = (A, M_1^C)$, the game proceeds as in hybrid 9 (whose behavior in this case is defined in hybrids 2 and 3).
   Else, the game looks for a T such that $H_2(A^*, B, T)$ has been defined as M_1^{C*}. If there is no such T, the game aborts (this behavior matches hybrid 8). Else, the game sends (j, A^*, T) to SIM'.

   (a) If SIM' aborts, the game outputs $\mathsf{MultiIndicesSvr}$ and aborts.

   (b) If SIM' outputs $i \in [\ell_{H_1} + 1]$, the game checks if $x_i = x$.

      (i) If so, the game computes $M_2^S := H_3(A^*, M_1^{C*}, T)$ and $K_S := H_4(T)$.

[14] A more "standard" and routine application of the forking lemma can be done if we reduce to the Gap CDH assumption, where the wrapper can check if the "bad event" happens by querying the DDH oracle. However, [DL24] explicitly mentions that their reduction is to CDH, not Gap CDH.

(ii) If not, the game sets $M_2^S := \perp$ and $K_S := \perp$.

(c) If SIM' outputs 0, the game sets $M_2^S := \perp$ and $K_S := \perp$.

4. The game uses SIM' to answer H_6 queries.

We first show that NoValidb happens with negligible probability. By the definition of SIM', the (joint) distribution of $(B_1, \cdots, B_{\ell_{SS}}, k, v = g^x)$ is indistinguishable from random. If it were truly random, then the probability that there is a $j \in [\ell_{SS}]$ such that $B_j = kv$ would be ℓ_{SS}/p. Thus,

$$\Pr[\mathsf{NoValidb}] \leq (1 - \mathbf{Adv}^{\ell_{SS}\text{-session CDC-RO}}_{\mathsf{SIM}'}) + \frac{\ell_{SS}}{p}.$$

Next, we show that if NoValidb does not happen and SIM' wins, then hybrid 10 is identical to hybrid 9. To begin with, if SIM' wins then it does not abort, so MultiIndicesSvr cannot happen. So, hybrid 10 sets $M_2^S := H_3(A^*, M_1^{C*}, T)$ and $K_S := H_4(T)$ if $M_1^{C*} = H_2(A^*, B, T)$, where T satisfies

$$\mathbf{ddh}(B - kg^{x_i}, A^* g^{u_S x_i}, T) = 1$$

for a sole x_i and $x_i = x$; and sets $M_2^S := \perp$ and $K_S := \perp$ otherwise. In other words, hybrid 10 sets $M_2^S := H_3(A^*, M_1^{C*}, T)$ and $K_S := H_4(T)$ if and only if $M_1^{C*} = H_2(A^*, B, T)$ for

$$T = \mathbf{cdh}(B - kg^x, A^* g^{u_S x}).$$

In contrast, hybrid 9 sets $M_2^S := H_3(A^*, M_1^{C*}, T)$ and $K_S := H_4(T)$ if and only if $M_1^{C*} = H_2(A^*, B, T_S)$, where $T_S = (A^* v^{u_S})^b$ for $v = g^x$ and $b = \mathbf{dl}(B - kg^{x_i})$, i.e.,

$$T_S = \mathbf{cdh}(B - kg^x, A^* g^{u_S x}).$$

Thus, these two conditions are equivalent, and hybrids 9 and 10 are identical.

We conclude that

$$\begin{aligned}\mathbf{Dist}^{9,10}_{\mathcal{Z}} &\leq (1 - \mathbf{Adv}^{\ell_{SS}\text{-session CDC-RO}}_{\mathsf{SIM}'}) + \Pr[\mathsf{NoValidb}] \\ &\leq 2(1 - \mathbf{Adv}^{\ell_{SS}\text{-session CDC-RO}}_{\mathsf{SIM}'}) + \frac{\ell_{SS}}{p}.\end{aligned}$$

The "bad event" MultiIndicesSvr is covered by our assumption that (the multi-session version of) CDC is very easy. The proof of Theorem 5 essentially contains a statistical argument why $\Pr[\mathsf{MultiIndicesSvr}]$ *is negligible; the bound there is*

$$\Pr[\mathsf{MultiIndicesSvr}] \leq \binom{\ell_{H_1}}{2} \cdot \frac{\ell_{H_6}}{p-1}.$$

Since the simulator in [DL24] has access to a DDH oracle, their argument is much simpler than ours. However, they still need to argue that $\Pr[\mathsf{MultiIndicesSvr}]$ *is negligible, and the bound in their Game G_4 misses the ℓ_{H_6} factor.*

Hybrid 11 (password guess extraction from client-to-server message): In step 3(b) of hybrid 10, the game checks if $x_i = x$. In this hybrid, the game finds the $H_1(s, \mathsf{pw}_i)$ query whose result is x_i, and checks if $\mathsf{pw}_i = \mathsf{pw}$ instead.

Recall that $x = H_1(s, \mathsf{pw})$. Since we have excluded collisions in H_1, $x_i = x$ is equivalent to $\mathsf{pw}_i = \mathsf{pw}$. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{10,11} = 0.$$

Hybrid 12 (program RO output on successful offline attack): In this hybrid, we let the password file be created lazily — that is, the game computes $v := g^x$ only upon a `StealPwdFile` command from $\mathcal{A}$. Here, x is defined as follows:

- If $H_1(s, \mathsf{pw})$ has been queried, the game sets $x := H_1(s, \mathsf{pw})$.
- Else, the game uses SIM''s output as x, and programs $H_1(s, \mathsf{pw}) := x$ when such a query is made. (If x collides with another H_1 query, the game aborts.)

At any time, if $B = kv$, the game outputs NoValidb and aborts.

Delaying the definition of v means that if NoValidb happens, hybrids 11 and 12 might be distinguishable due to different time of abort. The other difference between hybrids 11 and 12 is whether x is set to $H_1(s, \mathsf{pw})$, or x is sampled pseudorandomly at first and then $H_1(s, \mathsf{pw})$ is programmed to x. This makes no external difference. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{11,12} \leq \Pr[\mathsf{NoValidb}] \leq (1 - \mathbf{Adv}_{\mathsf{SIM}'}^{\ell_{SS}\text{-session CDC-RO}}) + \frac{\ell_{SS}}{p}.$$

This hybrid corresponds to Game G_9 in [DL24].

We can see that hybrid 12 is identical to the ideal world, with the game challenger split into the aPAKE functionality $\mathcal{F}$ and the simulator SIM. Summing up the distinguishing advantages among all hybrids, the distinguishing advantage between the real world and the ideal world is upper-bounded by

$$\begin{aligned}
&\ell_{SS}(\ell_{H_2} + \ell_{H_3} + \ell_{H_4})\mathbf{Adv}_{\mathcal{R}_{\text{CDH1}}}^{\text{CDH}} + \ell_{SS}\ell_{H_2} \cdot \mathbf{Adv}_{\mathcal{R}_{\text{CDH2}}}^{\text{CDH}} \\
&+\ell_{SS}(\ell_{H_2} + 2)\sqrt{(\ell_{H_6} + 2)\mathbf{Adv}_{\mathcal{R}_{\text{2ODH}}}^{\text{2ODH}}} \\
&+\ell_{SS}\binom{\ell_{H_1}}{2}\ell_{H_2}^2(\ell_{H_2} + 2)\mathbf{Adv}_{\mathcal{R}_{\ell\text{-for-1}}}^{\ell\text{-for-One Guessing}} \\
&+3(1 - \mathbf{Adv}_{\mathsf{SIM}'}^{\ell_{SS}\text{-session CDC-RO}}) \\
&+\frac{(\ell_{H_1} + 2)^2 + (\ell_{H_2} + 2)^2 + (\ell_{H_3} + 2)^2 + (\ell_{H_6} + 2)^2}{2(p-1)} \\
&+\frac{\ell_{SS}(\ell_{H_2} + 2)(\ell_{H_6} + 2) + \ell_{H_1}\ell_{H_2} + 4\ell_{SS}}{p-1}.
\end{aligned}$$

This completes the proof.

Comparing our concrete security bound with the one in [DL24, Theorem 2], the main points they miss are (1) the application of the forking lemma, especially the square root of the 2ODH reduction's advantage, and (2) the ℓ_{SS} factor in all reductions. There are various other minor gaps in [DL24], but the above two are the most significant. (Part of the security bound is not directly comparable, as they reduce to Gap CDH and Gap aSDH, whereas we reduce to the non-gap versions which result in worse bounds.)

4.3 UC-Security with Angels

Recall that in our entire proof of Theorem 8, the assumption that Multi-message CDC is very easy is *only* used in step 6(b) of the simulator. This suggests that even if Multi-message CDC is hard, the simulation still goes through if we give the simulator an oracle that solves DDH (note that Multi-message CDC can be trivially solved given a DDH oracle).

One caveat is that in UC with angels, the real adversary is also given the DDH oracle. This means that the reductions to CDH and aSDH must simulate the DDH oracle to the real adversary, which cannot be done unless these reductions can also access the DDH oracle. Thus, we need to reduce to the gap version of aSDH. This immediately yields the main result in [DL24]:

Corollary 1 ([DL24, Theorem 2])**.** *Let* DDH *be an oracle that given* $(A, B, C) \in \mathbb{F}_p$*, outputs* $\mathbf{ddh}(A, B, C)$*. The SRP protocol UC-realizes* $\mathcal{F}_{aPAKE}$ *with* DDH *if the Gap aSDH problem is hard in* $\mathbb{F}_p$*.*

Furthermore, this result is not optimal: the DDH oracle can be replaced by a Multi-message CDC oracle, which is slightly weaker.

5 UC-Insecurity of SRP

In this section, we consider the opposite direction to Section 4 and show that the conditions there are also *necessary* for the UC-security of SRP. The main theorem is:

Theorem 9. *If the SRP protocol UC-realizes* $\mathcal{F}_{aPAKE}$*, then all of the following assumptions hold in* $\mathbb{F}_p$*:*

1. *The CDH problem is hard;*
2. *The aSDH problem is hard;*
3. *The Multi-message CDC problem is very easy.*

Below we consider the three conditions one by one. The theorem follows immediately from Lemma 2 in Section 5.1, Lemma 5 in Section 5.2, and Lemma 6 in Section 5.3.[15]

[15] Since hardness of aSDH implies hardness of CDH (Theorem 4), item (2) in fact implies item (1) and we don't need Section 5.1. We choose to include it because Section 5.1 shows that if CDH is easy then SRP is insecure even against a *passive* adversary, whereas the attack assuming aSDH is easy requires an active adversary.

5.1 Necessity of Hardness of CDH

It is easy to see that if CDH is easy, then even a passive adversary can break SRP. Since negative results in UC are far less common than UC-security results, we present this UC-insecurity proof in detail.

We need the following definition and lemma, which are [CHK+05, Lemma A.1] (which deals with symmetric PAKE) extended to the context of aPAKE:

Definition 11. *Fix a (not necessarily PPT) simulator* SIM *that, for any PPT environment, generates a view indistinguishable from the real-world view. For any PPT environment $\mathcal{Z}$, let* SpuriousGuess($\mathcal{Z}$) *be the event that $\mathcal{Z}$ initiates a sub-session between C and S in which:*

- *$\mathcal{Z}$ instructs the adversary to be passive in this sub-session (i.e., $\mathcal{A}$ forwards all protocol messages without modification);*
- SIM *has not sent an* `OfflineTestPwd` *that resulted in "correct guess", nor has it sent a* `TestPwd` *command in a previous sub-session that resulted in "correct guess"; but*
- SIM *sends a* `TestPwd` *command for this sub-session.*

The criteria for a "spurious guess" are more complicated than that for PAKE: since aPAKE has a multi-session functionality and allows for offline attacks, we must rule out the possibility that the simulator knows the server's password via a successful offline attack or in a previous sub-session.

Lemma 1. *For any aPAKE protocol that UC-realizes $\mathcal{F}_{aPAKE}$ and any PPT environment $\mathcal{Z}$,* Pr[SpuriousGuess($\mathcal{Z}$)] *is negligible.*[16]

The proof is deferred to Appendix A.

Lemma 2. *If the CDH problem is easy, then the SRP protocol does not UC-realize $\mathcal{F}_{aPAKE}$.*

Proof. Since CDH is easy, there exists a PPT $\mathcal{A}_{\text{CDH}}$ that wins the CDH game with non-negligible probability. Consider the following environment $\mathcal{Z}$ against SRP:

1. $\mathcal{Z}$ chooses an arbitrary password `pw` and sends (`StorePwdFile`, sid, `pw`) to S.
2. $\mathcal{Z}$ initiates the two protocol parties' sub-sessions by sending (`CltSession`, $sid, ssid, S$, `pw`) to C and (`SvrSession`, $sid, ssid$) to S.
3. $\mathcal{Z}$ instructs the SRP man-in-the-middle adversary $\mathcal{A}$ to be passive in this sub-session. At the end of this sub-session, $\mathcal{Z}$ receives S's output K_S.

[16] Technically speaking we need the aPAKE protocol to be correct, namely if the two parties' passwords match and there is no adversary, then they arrive at the same key. This correctness property is explicitly used in the proof of Lemma 1, and one can easily check that SRP is correct. This has been extensively discussed in the context of (symmetric) PAKE; see [RX23] for more discussion.

4. $\mathcal{Z}$ queries $k := H_5(p, g)$, $u := H_6(A, B)$ and $x := H_1(s, \mathsf{pw})$, and invokes $\mathcal{A}_{\text{CDH}}$ on $(Ag^{ux}, B - kg^x)$.

5. When $\mathcal{A}_{\text{CDH}}$ outputs T', $\mathcal{Z}$ queries $K'_S := H_4(T')$. If $K'_S = K_S$, $\mathcal{Z}$ outputs 1; otherwise, it outputs 0.

In the real world, if $\mathcal{A}_{\text{CDH}}$ wins the CDH game then $T' = \mathbf{cdh}(Ag^{ux}, B - kg^x) = T$, so $K'_S = H_4(T') = H_4(T) = K_S$ and $\mathcal{Z}$ outputs 1. We have

$$\Pr[\mathcal{Z} \text{ outputs 1 in the real world}] \geq \mathbf{Adv}^{\text{CDH}}_{\mathcal{A}_{\text{CDH}}}.$$

In the ideal world, fix any PPT simulator SIM. By Lemma 1, the probability SIM ever sends a `TestPwd` command to $\mathcal{F}_{\text{aPAKE}}$ (before the sub-session completes), $\Pr[\mathsf{SpuriousGuess}(\mathcal{Z})]$, is negligible.[17] As long as SIM does not send this command, the S sub-session will be `fresh` in $\mathcal{F}_{\text{aPAKE}}$, and $\mathcal{F}_{\text{aPAKE}}$ will let S output a uniformly random K_S (independent of the rest of the game), so the probability that $K_S = K'_S$ is $1/2^\lambda$. We have

$$\Pr[\mathcal{Z} \text{ outputs 1 in the ideal world}] \leq \Pr[\mathsf{SpuriousGuess}(\mathcal{Z})] + \frac{1}{2^\lambda}.$$

Combined, we get

$$\mathbf{Dist}_{\mathcal{Z}} \geq \mathbf{Adv}^{\text{CDH}}_{\mathcal{A}_{\text{CDH}}} - \Pr[\mathsf{SpuriousGuess}(\mathcal{Z})] - \frac{1}{2^\lambda},$$

which is non-negligible. □

One caveat in the proof is that the CDH game requires $\mathcal{A}_{\text{CDH}}$'s inputs to be uniformly random. This is the case when $\mathcal{Z}$ invokes $\mathcal{A}_{\text{CDH}}$ on $(Ag^{ux}, B - kg^x)$, if A, B, k, u, x are all uniformly random; however, these values are in fact simulated by SIM, so they might only be pseudorandom. (If they are not even pseudorandom then another environment can easily detect that SIM is "cheating" while simulating these values.) This means that the $\mathbf{Adv}^{\text{CDH}}_{\mathcal{A}_{\text{CDH}}}$ term in the proof should, in fact, be replaced by $\mathbf{Adv}^{\text{CDH}}_{\mathcal{A}_{\text{CDH}}} - \mathsf{negl}$. A similar issue appears in Section 5.2, which we do not elaborate there.

5.2 Necessity of Hardness of aSDH

The necessity of the hardness of aSDH (or equivalently, Two-for-One Guessing) is surprisingly difficult. As a warmup, we first show that Two-for-One Guessing needs to be "moderately hard" for SRP to be UC-secure:

Lemma 3. *If there exists a PPT $\mathcal{A}_{\text{2-for-1}}$ that wins the Two-for-One Guessing game with probability non-negligibly greater than 1/2, then the SRP protocol does not UC-realize $\mathcal{F}_{aPAKE}$.*

[17] Note that $\mathcal{Z}$ never instructs the adversary to compromise S, so any `OfflineTestPwd` command from SIM will return "no password guess". Furthermore, $\mathcal{Z}$ only initiates one sub-session. Therefore, the conditions of Lemma 1 are satisfied.

Proof. Let $\mathcal{A}_{\text{2-for-1}}$ be the Two-for-One Guessing solver in the lemma statement. Consider the following environment $\mathcal{Z}$ against SRP:

1. $\mathcal{Z}$ chooses two arbitrary passwords $\mathsf{pw}_1 \neq \mathsf{pw}_2$.
2. $\mathcal{Z}$ queries $k = H_5(p, g)$, samples $s^* \leftarrow \{0,1\}^\lambda$, and queries $x_1 := H_1(s^*, \mathsf{pw}_1)$, $x_2 := H_1(s^*, \mathsf{pw}_2)$.
3. $\mathcal{Z}$ samples $i \leftarrow \{1, 2\}$ and initiates C's sub-session by sending ($\texttt{CltSession}$, $sid, ssid, S, \mathsf{pw}_i$) to C. ($\mathcal{Z}$ does not need to initiate S's sub-session as S is completely disregarded in the attack.)
4. $\mathcal{Z}$ invokes $\mathcal{A}_{\text{2-for-1}}$ on (k, x_1, x_2).
5. When $\mathcal{A}_{\text{2-for-1}}$ outputs B^*, $\mathcal{Z}$ instructs the SRP man-in-the-middle adversary $\mathcal{A}$ to send (s^*, B^*) to C.
6. When C sends (A, M_1^C) to $\mathcal{A}$, $\mathcal{Z}$ sends A to $\mathcal{A}_{\text{2-for-1}}$.
7. When $\mathcal{A}_{\text{2-for-1}}$ outputs (T_1, T_2), $\mathcal{Z}$ queries $u := H_6(A, B^*)$, computes $T_C^{(1)} := T_1 \cdot (B^* - kg^{x_1})^{ux_1}, T_C^{(2)} := T_2 \cdot (B^* - kg^{x_2})^{ux_2}$, and then queries $M_1^{(1)} := H_2(A, B^*, T_C^{(1)}), M_1^{(2)} := H_2(A, B^*, T_C^{(2)})$. If $M_1^{(i)} \neq M_1^C$, then $\mathcal{Z}$ outputs 0 and halts.
8. If $M_1^{(i)} = M_1^C$, $\mathcal{Z}$ queries $M_2^{S*} := H_3(A, M_1^{(i)}, T_C^{(i)})$ and $K'_C := H_4(T_C^{(i)})$, and instructs $\mathcal{A}$ to send M_2^{S*} to C.
9. When $\mathcal{Z}$ receives C's output K_C, if $K'_C = K_C$, $\mathcal{Z}$ outputs 1; otherwise, it outputs 0.

In the real world, if $\mathcal{A}_{\text{2-for-1}}$ wins the Two-for-One Guessing game then $T_i = \mathbf{cdh}(B^* - kg^{x_i}, A)$, so

$$T_C^{(i)} = T_i \cdot (B^* - kg^{x_i})^{ux_i}) = (B^* - kg^{x_i})^{a+ux_i} = T_C$$

and thus

$$M_1^{(i)} = H_2(A, B^*, T_C^{(i)}) = H_2(A, B^*, T_C) = M_1^C,$$
$$K'_C = H_4(T_C^{(i)}) = H_4(T_C) = K_C,$$

so both of $\mathcal{Z}$'s checks (steps 7 and 9) pass and $\mathcal{Z}$ outputs 1. We have

$$\Pr[\mathcal{Z} \text{ outputs 1 in the real world}] \geq \mathbf{Adv}^{\text{Two-for-One Guessing}}_{\mathcal{A}_{\text{2-for-1}}}.$$

In the ideal world, fix any PPT simulator SIM. Let $\mathsf{CorrectGuess}$ be the event that SIM sends ($\texttt{TestPwd}, sid, ssid, C, \mathsf{pw}_i$) to $\mathcal{F}_{\text{aPAKE}}$ before the sub-session completes.[18] Note that *i is independent of* SIM*'s view*; therefore, $\Pr[\mathsf{CorrectGuess}] \leq$

[18] Here we assume w.l.o.g. that SIM only sends one ($\texttt{TestPwd}, sid, ssid, C, \star$) command, as subsequent such commands will be ignored by $\mathcal{F}_{\text{aPAKE}}$.

1/2. If CorrectGuess does not happen, the C sub-session will be `fresh` or `interrupted` in $\mathcal{F}_{\text{aPAKE}}$, and $\mathcal{F}_{\text{aPAKE}}$ will let C output a uniformly random K_C (independent of the rest of the game) or $\perp$, so the probability that $K_C = K'_C$ is $1/2^\lambda$. We have

$$\Pr[\mathcal{Z} \text{ outputs 1 in the ideal world}] \leq \Pr[\mathsf{CorrectGuess}] + \frac{1}{2^\lambda} \leq \frac{1}{2} + \frac{1}{2^\lambda}.$$

Combined, we get

$$\mathbf{Dist}_{\mathcal{Z}} \geq \mathbf{Adv}^{\text{2-for-1 Guessing}}_{\mathcal{A}_{\text{2-for-1}}} - \frac{1}{2} - \frac{1}{2^\lambda},$$

which is non-negligible since $\mathbf{Adv}^{\text{2-for-1 Guessing}}_{\mathcal{A}_{\text{2-for-1}}}$ is non-negligibly greater than 1/2. □

What if the best Two-for-One Guessing solver only has advantage, say, 0.2? Then the Two-for-One Guessing adversary against SRP only has a 0.2 chance to test two passwords per sub-session, so on average it can only test 0.4 passwords per sub-session — worse than a regular online attack where the adversary can test one password per sub-session! As such, it is not even clear whether Two-for-One Guessing constitutes an attack in this case.

On the other hand, if DDH is very easy, then intuitively the SRP adversary can check if the Two-for-One Guessing solver wins, and only make the corresponding H_2 queries if so (i.e., discard the useless T values if the Two-for-One Guessing solver loses). In this way the adversary can still test two passwords per sub-session and this is a "real" attack.

Lemma 4. *If the Two-for-One Guessing problem is easy and the DDH problem is very easy, then the SRP protocol does not UC-realize $\mathcal{F}_{aPAKE}$.*

Proof. We only provide a brief sketch. The environment $\mathcal{Z}$ is the same as in Lemma 3, except that in step 7, when $\mathcal{A}_{\text{2-for-1}}$ outputs (T_1, T_2), $\mathcal{Z}$ checks if $\mathcal{A}_{\text{2-for-1}}$ wins by running the DDH distinguisher, and if not, $\mathcal{Z}$ outputs 0 and halts. (If $\mathcal{A}_{\text{2-for-1}}$ wins, $\mathcal{Z}$ proceeds as before.) In the real world the probability that $\mathcal{Z}$ outputs 1 is still lower-bounded by $\mathbf{Adv}^{\text{Two-for-One Guessing}}_{\mathcal{A}_{\text{2-for-1}}}$; in the ideal world,

$$\begin{aligned}
\Pr[\mathcal{Z} \text{ outputs 1}] &= \Pr[\mathcal{Z} \text{ outputs 1} \wedge \mathcal{A}_{\text{2-for-1}} \text{ wins}] + \Pr[\mathcal{Z} \text{ outputs 1} \wedge \mathcal{A}_{\text{2-for-1}} \text{ loses}] \\
&= \Pr[\mathcal{Z} \text{ outputs 1} \wedge \mathcal{A}_{\text{2-for-1}} \text{ wins}] \\
&= \Pr[\mathcal{Z} \text{ outputs 1} \mid \mathcal{A}_{\text{2-for-1}} \text{ wins}] \cdot \mathbf{Adv}^{\text{2-for-1 Guessing}}_{\mathcal{A}_{\text{2-for-1}}} \\
&\leq \left(\frac{1}{2} + \frac{1}{2^\lambda}\right) \cdot \mathbf{Adv}^{\text{2-for-1 Guessing}}_{\mathcal{A}_{\text{2-for-1}}}
\end{aligned}$$

(where the last inequality is due to the same argument as in Lemma 3 — note that the analysis there does not depend on whether $\mathcal{A}_{\text{2-for-1}}$ wins or not), so

$$\mathbf{Dist}_{\mathcal{Z}} \geq \left(\frac{1}{2} - \frac{1}{2^\lambda}\right) \mathbf{Adv}^{\text{2-for-1 Guessing}}_{\mathcal{A}_{\text{2-for-1}}},$$

which is non-negligible. □

The same argument can be made if the Two-for-One Guessing solver can tell whether itself wins or not (with overwhelming accuracy). The last piece of the whole proof is to convert any Two-for-One Guessing solver into such a "good" solver. This requires some clever tricks, and we defer the final proof to Appendix B:

Lemma 5. *If the Two-for-One Guessing problem is easy, then the SRP protocol does not UC-realize $\mathcal{F}_{aPAKE}$.*

5.3 Necessity of Very Easiness of Multi-Session CDC

In this section we show that if SRP is UC-secure, then Multi-session CDC must be very easy. Our main result is:

Lemma 6. *If the SRP protocol UC-realizes $\mathcal{F}_{aPAKE}$, then the ℓ_{SS}-session CDC problem is very easy (for any polynomial ℓ_{SS}).*

The proof is a straightforward extension of the proof in the single-session case; below we focus on the single-session case and explain briefly at the end how to generate to the multi-session case.

Non-adaptive CDC-RO. The following non-adaptive form of CDC-RO, where the "testing index" i must be chosen at the beginning of the game, greatly facilitates the proof:

Definition 12 (Non-adaptive CDC-RO (NA-CDC-RO))**.** *The **NA-CDC-RO** game is defined as follows:*

0. *The game challenger picks an index $i \in [0, q]$. (i is not sent to* SIM.*)*
1. SIM *outputs $(B, k, x_1, \cdots, x_q) \in \mathbb{F}_p \times \mathbb{F}_p^* \times \mathbb{Z}_{p-1}^q$. The distribution of $(B, k, x_1, \cdots, x_q)$ must be indistinguishable from uniform.*
2. *The game challenger outputs $(A, T) \in (\mathbb{F}_p^*)^2$, with the following constraints: if $i \in [q]$ then i is the only index such that $\mathbf{ddh}(B - kg^{x_i}, Ag^{ux_i}, T) = 1$, and if $i = 0$ then $\mathbf{ddh}(B - kg^{x_{i'}}, Ag^{ux_{i'}}, T) = 0$ for all $i' \in [q]$. Here, $u = H(A, B)$ where $H : \mathbb{F}_p^* \times \mathbb{F}_p \to \mathbb{Z}_{p-1}$ is an RO simulated by* SIM.
3. SIM *outputs an index in $[0, q]$, and wins if its output is i.*

We say the NA-CDC-RO problem is very easy in $\mathbb{F}_p$ if there is a PPT SIM *such that for any PPT game challenger, the probability that* SIM *wins the above game is overwhelming.*

Lemma 7. *If the NA-CDC-RO problem is very easy, then the CDC-RO problem is very easy.*

The proof is simple and is deferred to Appendix C.

Proof in the single-session case.

Lemma 8. *If SRP UC-realizes $\mathcal{F}_{aPAKE}$, then the NA-CDC-RO problem is very easy.*

Proof. Since SRP UC-realizes $\mathcal{F}_{\text{aPAKE}}$, there exists a PPT simulator SIM that generates an indistinguishable view for any PPT environment $\mathcal{Z}$. The high-level idea is that when facing an adversary that assumes the role of C, SIM must send a `TestPwd` command to $\mathcal{F}_{\text{aPAKE}}$ on password guess pw^*, which is extracted from C's message (A, M_1^{C*}); this "core" of SIM is essentially an NA-CDC-RO solver. (The formal proof is a bit technical, though: it involves SIM and $\mathcal{Z}$ in the SRP UC game, and the solver SIM' and the challenger **Ch** in the NA-CDC-RO game. We will use SIM to construct SIM', and use **Ch** to construct $\mathcal{Z}$ — in such a way that if SIM' doesn't work for **Ch**, then SIM doesn't work for $\mathcal{Z}$.)

Concretely, recall that SIM directly communicates with two parties, $\mathcal{F}_{\text{aPAKE}}$ and $\mathcal{Z}$. We construct an adversary SIM' in the NA-CDC-RO game:

1. SIM' (as $\mathcal{Z}$) queries $k := H_5(p, g)$ to SIM. Then SIM' chooses arbitrary but distinct passwords $\mathsf{pw}_1, \cdots, \mathsf{pw}_q$ and (as $\mathcal{F}_{\text{aPAKE}}$) initiates S's sub-session by sending $(\texttt{SvrSession}, sid, ssid)$ to SIM.

2. When SIM outputs the first protocol message (s, B), SIM' (as $\mathcal{Z}$) queries $x_1 := H_1(s, \mathsf{pw}_1), \cdots, x_q := H_1(s, \mathsf{pw}_q)$ to SIM.

3. SIM' sends $(B, k, x_1, \cdots, x_q)$ to its game challenger.

4. When the game challenger outputs (A, T), SIM' (as $\mathcal{Z}$) queries $M_1^{C*} := H_2(A, B, T)$ to SIM and sends (A, M_1^{C*}) to SIM from C to S.

5. When SIM sends $(\texttt{TestPwd}, sid, ssid, \mathsf{pw}^*)$,

   - If $\mathsf{pw}^* = \mathsf{pw}_{i^*}$ for some $i^* \in [q]$, then SIM' outputs i^*. (Note that SIM' can halt here and there is no need to send "correct/wrong guess" to SIM.)
   - Else, SIM' outputs 0.

   If no such command is sent, SIM' also outputs 0.

Let **Ch** be any PPT challenger in the NA-CDC-RO game. The lemma immediately follows from the two claims below:

Claim 1. *If $i = 0$, i.e., there is no $i' \in [q]$ such that $\mathbf{ddh}(B - kg^{x_{i'}}, Ag^{ux_{i'}}, T) = 1$, then SIM' wins with overwhelming probability.*

Let $\mathsf{SpuriousIndex}$ be the event that $i = 0$ but SIM sends $(\texttt{TestPwd}, sid, ssid, \mathsf{pw}_{i^*})$ for some $i^* \in [q]$; we only need to prove that $\Pr[\mathsf{SpuriousIndex}]$ is negligible. Consider the following environment $\mathcal{Z}_0$:

0. $\mathcal{Z}_0$ runs **Ch**'s code and generates index $i = 0$.[19]

1. $\mathcal{Z}_0$ queries $k := H_5(p, g)$. Then $\mathcal{Z}_0$ chooses arbitrary but distinct passwords $\mathsf{pw}_1, \cdots, \mathsf{pw}_q$, samples $i' \leftarrow [q]$, and sends $(\texttt{StorePwdfile}, sid, \mathsf{pw}_{i'})$ to S. After that, $\mathcal{Z}_0$ initiates S's sub-session by sending $(\texttt{SvrSession}, sid, ssid)$ to S.

2. When SIM outputs the first protocol message (s, B), $\mathcal{Z}_0$ queries $x_1 := H_1(s, \mathsf{pw}_1), \cdots, x_q := H_1(s, \mathsf{pw}_q)$ to SIM.

3. $\mathcal{Z}_0$ runs **Ch**'s code on $(B, k, x_1, \cdots, x_q)$ and obtain (A, T).

4. $\mathcal{Z}_0$ queries $M_1^{C*} := H_2(A, B, T)$ and sends (A, M_1^{C*}) from C to S.

5. When S outputs K_S, if $K_S = \perp$, $\mathcal{Z}_0$ outputs 1; otherwise, it outputs 0.

In the real world, S computes

$$T_S = (Av^u)^b = (Ag^{ux_i})^b = \mathbf{cdh}(B - kg^{x_i}, Ag^{ux_i}) \neq T,$$

so

$$M_1^S = H_2(A, B, T_S) \neq H_2(A, B, T) = M_1^{C*}$$

and S's check fails (except in the case of an H_2 collision, which happens with probability $1/2^\lambda$). Then S outputs $K_S = \perp$, causing $\mathcal{Z}$ to output 1. We have

$$\Pr[\mathcal{Z}_0 \text{ outputs 1 in the real world}] = 1 - \frac{1}{2^\lambda}.$$

In the ideal world, first observe that the combined behavior of $\mathcal{Z}_0$ and $\mathcal{F}_{\text{aPAKE}}$ is identical to the behavior of SIM', so event $\mathsf{SpuriousIndex}$ is still well-defined (and has the same probability) in $\mathcal{Z}_0$. Also note that i' *is independent of* SIM*'s view*; therefore, if $\mathsf{SpuriousIndex}$ happens, then with probability $1/q$, $i^* = i'$, i.e., SIM "accidently" hits the correct password. This will trigger $\mathcal{F}_{\text{aPAKE}}$ to mark the S sub-session `compromised`, in which case S will not output $\perp$ (and $\mathcal{Z}_0$ will output 0). We have

$$\Pr[\mathcal{Z}_0 \text{ outputs 0 in the ideal world}] \geq \frac{1}{q} \Pr[\mathsf{SpuriousIndex}],$$

i.e.,

$$\Pr[\mathcal{Z}_0 \text{ outputs 1 in the ideal world}] \leq 1 - \frac{1}{q} \Pr[\mathsf{SpuriousIndex}].$$

Combined, we get

$$\mathbf{Dist}_{\mathcal{Z}_0} \geq \frac{1}{q} \Pr[\mathsf{SpuriousIndex}] - \frac{1}{2^\lambda},$$

and since $\mathbf{Dist}_{\mathcal{Z}_0}$ is negligible, $\Pr[\mathsf{SpuriousIndex}]$ must be negligible.

[19] If **Ch** does not generate index $i = 0$, then $\mathcal{Z}_0$ runs **Ch**'s code polynomial times so that it generates $i = 0$ with overwhelming probability. This works unless **Ch** generates $i = 0$ with negligible probability — in which case we do not need to consider the $i = 0$ case in the first place (i.e., Claim 2 below suffices).

Claim 2. *If $i \in [q]$, i.e.,* $\mathbf{ddh}(B - kg^{x_i}, Ag^{ux_i}, T) = 1$, *then* SIM' *wins with overwhelming probability.*

Let $\mathsf{CorrectIndex}$ be the event that SIM sends $(\texttt{TestPwd}, sid, ssid, \mathsf{pw}_i)$ to $\mathcal{F}_{\text{aPAKE}}$; we only need to prove that $\Pr[\mathsf{CorrectIndex}]$ is overwhelming. Consider the following environment $\mathcal{Z}$:

0. $\mathcal{Z}$ runs **Ch**'s code and generates index $i \neq 0$.
1. $\mathcal{Z}$ follows steps 1–4 of $\mathcal{Z}_0$ above, except that in step 1 $\mathcal{Z}$ sends $(\texttt{StorePwdfile}, sid, \mathsf{pw}_i)$ to S (i.e., $\mathcal{Z}$ uses pw_i as the password where i is the index generated by **Ch**, rather than a random password among $\mathsf{pw}_1, \cdots, \mathsf{pw}_q$).
2. When S outputs K_S, $\mathcal{Z}$ queries $K'_S := H_4(T)$. If $K'_S = K_S$, $\mathcal{Z}$ outputs 1; otherwise, it outputs 0.

In the real world, S computes

$$T_S = (Av^u)^b = (Ag^{ux_i})^b = \mathbf{cdh}(B - kg^{x_i}, Ag^{ux_i}) = T,$$

so

$$M_1^S = H_2(A, B, T_S) = H_2(A, B, T) = M_1^{C*}$$

and S's check passes. Then S outputs

$$K_S = H_4(T_S) = H_4(T) = K'_S,$$

causing $\mathcal{Z}$ to output 1. We have

$$\Pr[\mathcal{Z} \text{ outputs 1 in the real world}] = 1.$$

In the ideal world, first observe that the combined behavior of $\mathcal{Z}$ and $\mathcal{F}_{\text{aPAKE}}$ is identical to the behavior of SIM', so event $\mathsf{CorrectIndex}$ is still well-defined (and has the same probability) in $\mathcal{Z}$. If $\mathsf{CorrectIndex}$ does not happen, then the S sub-session will be `fresh` or `interrupted` in $\mathcal{F}_{\text{aPAKE}}$, and $\mathcal{F}_{\text{aPAKE}}$ will let S output a uniformly random K_C (independent of the rest of the game) or $\perp$, so the probability that $K_S = K'_S$ is $1/2^\lambda$. We have

$$\Pr[\mathcal{Z} \text{ outputs 1 in the ideal world}] \leq \Pr[\mathsf{CorrectIndex}] + \frac{1}{2^\lambda}.$$

Combined, we get

$$\mathbf{Dist}_{\mathcal{Z}} \geq 1 - \Pr[\mathsf{CorrectIndex}] - \frac{1}{2^\lambda},$$

and since $\mathbf{Dist}_{\mathcal{Z}}$ is negligible, $\Pr[\mathsf{CorrectIndex}]$ must be overwhelming. □

Extension to multi-session. Recall that in the ℓ_{SS}-session CDC-RO game, there are ℓ_{SS} sessions in total, each of which has its own B and (A, T), but k and $x_1, \cdots, x_q$ are shared across all sessions; the solver needs to "win" in all sessions. The proof ideas in the single-session setting still hold: first define an equivalent non-adaptive version of ℓ_{SS}-session CDC-RO, where in each session the "testing index" i needs to be chosen at the beginning of the game; then show that any simulator for the UC-security of SRP must contain a "core" that solves ℓ_{SS}-session NA-CDC-RO, since otherwise there are two environments (analogous to $\mathcal{Z}_0$ and $\mathcal{Z}$ in the proof of Lemma 8) that have non-negligible distinguishing advantage by checking S's outputs in all ℓ_{SS} sub-sessions.

Can the security of SRP be salvaged with a weaker functionality or a stronger idealized model? One might ask whether we can get rid of the very easiness of Multi-message CDC and prove the (standard, without angels) UC-security of SRP, either w.r.t. a relaxed UC aPAKE functionality — similar to the relaxed UC PAKE functionality in [ABB+20], where the `TestPwd` command can be sent after a session completes — or in a stronger idealized model such as the generic field model. This has been explored in [DL24, Section 6.3 and Appendix D]; in short, the answer to both questions is "No, you cannot prove it". Since the arguments in [DL24] are clear and convincing enough, we do not repeat them and refer the readers to [DL24] for more details.

6 Game-Based Security for SRP

6.1 Game-Based Security Definition for aPAKE

Our game-based security definition for aPAKE is mostly identical to the one in [BP13], which in turn closely follows the standard game-based security definition for (symmetric) PAKE [AP05] — except that the possibility of server compromise and offline dictionary attack must be taken into account. We explain the minor differences between our definition and [BP13] at the end of this section.

Setup. Let $\mathcal{D}$ be the set of all candidate passwords; we generally assume $\mathcal{D}$ has polynomial size.[20] For each (client, server) pair (C, S), sample $\mathsf{pw} \leftarrow \mathcal{D}$. C holds a password pw, while S holds a password file file derived from the same password pw.

Protocol execution. The adversary $\mathcal{A}$ can create several concurrent instances U^i (where U is either a client or a server), and can interact with them via the following commands:

[20] This is a difference between the game-based and UC definitions for (a)PAKE: in UC there is no notion of the password dictionary, whereas in the game-based setting such a set must be made explicit (in order to define the adversary's baseline advantage, which contains a non-negligible term $1/|\mathcal{D}|$).

- $\texttt{Execute}(C^i, S^j)$ (passive attack command) outputs all protocol messages that are sent during an honest execution of the protocol between C^i and S^j.
- $\texttt{Send}(U^i, V^j, m^*)$ (active attack command) outputs the protocol message that V^j would generate after receiving m^* from U^i. As a special case, $\texttt{Send}(U^i, V^j, \perp)$ ("prompt" command) outputs the first protocol message that U^i would generate after being initialized. For any pair of instances (U^i, V^j), $\mathcal{A}$ cannot query both $\texttt{Execute}(U^i, V^j)$ and $\texttt{Send}(U^i, V^j, \star)$ (or $\texttt{Send}(V^j, U^i, \star)$).
- $\texttt{Compromise}(S)$ (server compromise command) outputs file.

Security. To actually define the security of aPAKE, we additionally let $\mathcal{A}$ have access to a challenge oracle $\texttt{Test}(U^i)$ in the real-or-random flavor: the game challenger samples $b \leftarrow \{0,1\}$ at the beginning of the game, and $\texttt{Test}(U^i)$ consistently outputs U^i's real key if $b = 1$ and a random key if $b = 0$. Concretely, the output of $\texttt{Test}(U^i)$ is defined as follows:

- If no session key has been computed by U^i, the output is $\perp$.
- If U^i is a client instance whose corresponding server S has been compromised, the output is $\perp$. (In this case $\mathcal{A}$ can trivially impersonate S and compute U^i's key.)
- If $\texttt{Test}(U^i)$ has already been queried and returned a non-$\perp$ result, the output remains the same.
- If U^i has a partner V^j with matching transcripts, a $\texttt{Test}(V^j)$ command has been sent and returned a non-$\perp$ result, the output is the same as $\texttt{Test}(V^j)$.
- Else, the output is the session key of U^i if $b = 1$, and a random string in $\{0,1\}^\lambda$ if $b = 0$.

At the end of the game, $\mathcal{A}$ outputs a bit b^*. Its advantage is defined as $\mathbf{Adv}_{\mathcal{A}} = 2\Pr[b^* = b] - 1$.

Definition 13. *An aPAKE protocol is (game-based) secure if for any PPT adversary $\mathcal{A}$,*

$$\mathbf{Adv}_{\mathcal{A}} \leq \frac{q}{|\mathcal{D}|} + b_{\mathrm{com}} \cdot \frac{\ell_{\mathsf{pw}}}{|\mathcal{D}|} + \mathsf{negl},$$

where ℓ_{pw} is the number of $\mathcal{A}$'s RO queries, $b_{\mathrm{com}} = 1$ if a $\texttt{Compromise}(S)$ command has been sent and 0 otherwise, and q is the number of $\texttt{Send}(\star, \star, \perp)$ commands.

Discussion. In our definition, all server instances (for the same server) share the same password file, which is leaked if there is a `Compromise` command. This matches the UC definition: essentially we assume there is only one session (with multiple sub-sessions), although an extension to the multi-session setting would be straightforward.

The bound in the definition should be interpreted as follows: the adversary can test one password — equivalently, gain $1/|\mathcal{D}|$ advantage — in either

- An online attack on an instance, or
- An offline attack, which involves an RO query. (If the server is not compromised throughout the game, then offline attacks cannot be performed and thus this term is 0.)

If we disallow the `Compromise` command and always set $b_{\text{com}} = 0$, then it becomes the standard game-based security definition for (symmetric) PAKE [AP05].

Note that we explicitly assume that an offline attack is done via querying the RO, which is certainly the case for SRP. In general, we can replace the $\ell_{\mathsf{pw}}/|\mathcal{D}|$ term with $(t/t_{\mathsf{file}})/|\mathcal{D}|$, where t is $\mathcal{A}$'s run time and t_{file} is the time of computing file from pw (the time of an RO query in the case of SRP). This is the approach taken in [BP13]; we choose to use the simpler $\ell_{\mathsf{pw}}/|\mathcal{D}|$ term because it is conceptually cleaner and already suffices for our purpose (in fact, all existing aPAKE protocols have an RO in the password file, so we don't lose much generality).

6.2 UC-Security with Angels Implies Game-Based Security

We now show that for aPAKE, UC-security with any angles implies game-based security. We first introduce another security notion we call *weak UC-security*, and then show UC-security with angels $\Rightarrow$ weak UC-security $\Rightarrow$ game-based security.

Weak UC-security.

Definition 14 (Weak UC-security)**.** *The definition of* ***weak UC-security*** *is the same as UC-security, except that the simulator is computationally unbounded. (The real adversary still has to be PPT.)*

Note that composition does not work anymore for weak UC-security; however, we only need weak UC-security to be a well-defined notion.

Compared with UC-security with angels, in weak UC-security the simulator is stronger (a computationally unbounded simulator can compute any oracle results by itself) and the real adversary is weaker (it does not have the additional oracles). We thus have:

Lemma 9. *If a protocol* Π *is UC-secure with angels* $\mathcal{O}$*, then* Π *is also weakly UC-secure.*

Weak UC-security implies game-based security.

Lemma 10. *If* Π *is an aPAKE protocol that weakly UC-realizes* $\mathcal{F}_{aPAKE}$*, then* Π *is also game-based secure.*

Proof. Consider any PPT adversary $\widehat{\mathcal{A}}$ in the game-based security definition for Π; we want to upper-bound $\mathbf{Adv}_{\widehat{\mathcal{A}}}$. (We use $\widehat{\mathcal{A}}$ instead of $\mathcal{A}$ to emphasize that it is the adversary in the game-based definition, not the UC adversary.) The high-level idea is to construct a UC environment $\mathcal{Z}$ that essentially instructs its adversary to do what $\widehat{\mathcal{A}}$ does (with necessary changes for "translating" a game-based adversary into a UC adversary), and since Π is UC-secure, $\mathbf{Adv}_{\widehat{\mathcal{A}}}$ is close to the probability that the UC simulator learns the password — which is the same as the upper bound of $\widehat{\mathcal{A}}$'s advantage allowed in the game-based definition for aPAKE.

Concretely, environment $\mathcal{Z}$ works as follows:

0. $\mathcal{Z}$ samples $b \leftarrow \{0,1\}$ and invokes $\widehat{\mathcal{A}}$.

1. For each pair of parties (C, S), $\mathcal{Z}$ samples a password $\mathtt{pw} \leftarrow \mathcal{D}$. It then picks a fresh sid and sends $(\mathtt{StorePwdFile}, sid, \mathtt{pw})$ to S.

2. When $\widehat{\mathcal{A}}$ sends an $\mathtt{Execute}(C^i, S^j)$ command, $\mathcal{Z}$ picks a fresh $ssid_{i,j}$ and sends $(\mathtt{CltSession}, sid, ssid_{i,j}, S, \mathtt{pw})$ to C and $(\mathtt{SvrSession}, sid, ssid_{i,j})$ to S. $\mathcal{Z}$ then instructs the aPAKE adversary $\mathcal{A}$ to be passive in this sub-session. After the sub-session completes, $\mathcal{Z}$ provides $\widehat{\mathcal{A}}$ with its transcript. (Note that C^i is a client instance in the security game simulated by $\mathcal{Z}$, whereas C is a UC protocol party that interacts with $\mathcal{Z}$.)

3. When $\widehat{\mathcal{A}}$ sends a $\mathtt{Send}(C^i, S^j, \perp)$ command, $\mathcal{Z}$ sends $(\mathtt{CltSession}, sid, ssid_{i,j}, S, \mathtt{pw})$ to C (if $ssid_{i,j}$ is undefined, $\mathcal{Z}$ picks a fresh $ssid_{i,j}$). When $\widehat{\mathcal{A}}$ sends a $\mathtt{Send}(S^j, C^i, \perp)$ command, $\mathcal{Z}$ sends $(\mathtt{SvrSession}, sid, ssid_{i,j})$ to S (if $ssid_{i,j}$ is undefined, $\mathcal{Z}$ picks a fresh $ssid_{i,j}$). (In this way $ssid_{i,j}$ is always the sub-session ID between instances C^i and S^j.)
   When $\widehat{\mathcal{A}}$ sends a $\mathtt{Send}(U^i, V^j, m^*)$ command, $\mathcal{Z}$ instructs the aPAKE adversary $\mathcal{A}$ to send a message $(ssid_{i,j}, m^*)$ (if $U = C$ and $V = S$) or $(ssid_{j,i}, m^*)$ (if $U = S$ and $V = C$) to V.
   In all cases above, $\mathcal{Z}$ forwards the message $\mathcal{A}$ receives to $\widehat{\mathcal{A}}$.

4. When $\widehat{\mathcal{A}}$ sends a $\mathtt{Compromise}(S)$ command, $\mathcal{Z}$ instructs $\mathcal{A}$ to send $(\mathtt{StealPwdFile}, sid)$ to S and forwards the message (the password file) $\mathcal{A}$ receives to $\widehat{\mathcal{A}}$.

5. When $\widehat{\mathcal{A}}$ sends a $\mathtt{Test}(U^i)$ command, $\mathcal{Z}$ fetches $ssid_{i,j}$ (where j is the index of U^i's partner V^j) and only considers the sub-session whose ID is $ssid_{i,j}$. It responds as follows:

   - If $\mathcal{Z}$ has received no session key from U, the output is $\perp$.
   - If U^i is a client instance whose corresponding server S has been compromised, the output is $\perp$.

- If $\texttt{Test}(U^i)$ has already been queried and returned a non-$\perp$ result, the output remains the same.
- If U and V have matching transcripts (i.e., the aPAKE adversary $\mathcal{A}$ is passive in this sub-session), a $\texttt{Test}(V^j)$ command has been sent and returned a non-$\perp$ result, the output is the same as $\texttt{Test}(V^j)$.
- Else, the output is U's session key if $b = 1$, and a random string in $\{0,1\}^\lambda$ if $b = 0$.

6. When $\widehat{\mathcal{A}}$ outputs a bit b^*, $\mathcal{Z}$ outputs 1 if $b^* = b$ and 0 otherwise.

If $\mathcal{Z}$ is in the real world, then it exactly simulates Π's security game (as in Section 6.1) for $\widehat{\mathcal{A}}$. We have

$$\Pr[\mathcal{Z} \text{ outputs 1 in the real world}] = \Pr[b^* = b] = \frac{1}{2} + \frac{1}{2}\mathbf{Adv}_{\widehat{\mathcal{A}}}.$$

Now suppose $\mathcal{Z}$ is in the ideal world with a (not necessarily PPT) simulator SIM; we want to upper-bound the probability of $b^* = b$ in this case. Let $\mathsf{CorrectGuess}$ be the event that SIM sends an $\texttt{OfflineTestPwd}$ or $\texttt{TestPwd}$ command that results in "correct guess".

Claim 3. $\Pr[\mathsf{CorrectGuess}] \leq \dfrac{q}{|\mathcal{D}|} + b_{\text{com}} \cdot \dfrac{\ell_{\mathsf{pw}}}{|\mathcal{D}|} + \mathsf{negl}$.

By Lemma 1, unless and until $\mathsf{CorrectGuess}$ happens, the probability that SIM sends $\texttt{TestPwd}$ for an instance initiated by $\texttt{Execute}$ is negligible. Therefore, SIM is (essentially) limited to

- At most q $\texttt{TestPwd}$ commands (corresponding to the q $\texttt{Send}$ commands of $\widehat{\mathcal{A}}$); and
- At most $\ell_{\mathsf{pw}}/|\mathcal{D}|$ $\texttt{OfflineTestPwd}$ commands, assuming S is compromised (corresponding to the ℓ_{pw} RO queries of $\widehat{\mathcal{A}}$ — note that SIM's $\texttt{OfflineTestPwd}$ commands will not yield any result if S is not compromised).

SIM can test one password for each command, so the overall probability that SIM hits the correct password is upper-bounded by $q/|\mathcal{D}| + b_{\text{com}} \cdot (\ell_{\mathsf{pw}}/|\mathcal{D}|) + \mathsf{negl}$.

Claim 4. *Assuming* $\mathsf{CorrectGuess}$ *does not happen, the probability that* $b^* = b$ *is* $1/2$.

This is because the session keys generated by $\mathcal{F}_{\text{aPAKE}}$ (and may be obtained by $\widehat{\mathcal{A}}$) are random and independent of the rest of the game — in particular, they are independent of $\widehat{\mathcal{A}}$'s view. Therefore, $\widehat{\mathcal{A}}$ learns no information about b, so it guesses b correctly with probability exactly $1/2$.

Combining the two claims above yields

$$\Pr[\mathcal{Z} \text{ outputs 1 in the ideal world}] \leq \frac{1}{2} + \frac{1}{2}\left(\frac{q}{|\mathcal{D}|} + b_{\text{com}} \cdot \frac{\ell_{\mathsf{pw}}}{|\mathcal{D}|}\right) + \mathsf{negl}.$$

Overall,

$$\mathbf{Dist}_{\mathcal{Z}} \geq \frac{1}{2}\left(\mathbf{Adv}_{\widehat{\mathcal{A}}} - \frac{q}{|\mathcal{D}|} - b_{\mathrm{com}} \cdot \frac{\ell_{\mathsf{pw}}}{|\mathcal{D}|}\right) - \mathsf{negl},$$

which is negligible. We have

$$\mathbf{Adv}_{\widehat{\mathcal{A}}} \leq \frac{q}{|\mathcal{D}|} + b_{\mathrm{com}} \cdot \frac{\ell_{\mathsf{pw}}}{|\mathcal{D}|} + \mathsf{negl},$$

so Π is game-based secure. □

By Lemma 9 (UC-security with any angles ⇒ weak UC-security) and Lemma 10 (weak UC-security ⇒ game-based security), we get

Theorem 10. *If Π is an aPAKE protocol that UC-realizes $\mathcal{F}_{aPAKE}$ with angels $\mathcal{O}$, then Π is also game-based secure.*

Combining Theorem 10 and Corollary 1 (SRP is UC-secure with the DDH angel assuming Gap aSDH is hard), we conclude that SRP is game-based secure assuming Gap aSDH is hard.

6.3 A Direct Proof of the Game-Based Security of SRP

The security argument in Section 6.2 relies on the hardness of *Gap* aSDH. In this section we give a more direct proof that SRP is in fact game-based secure under the weaker assumption that aSDH is hard.

Lemma 11. *SRP weakly UC-realizes $\mathcal{F}_{aPAKE}$ if the aSDH problem is hard in $\mathbb{F}_p$.*

Proof. Most of the proof is identical to the proof of Section 4, so we only provide a brief sketch here. The simulator is the same as the one in the proof of Section 4, except that in step 6(b) it solves the Multi-session CDC-RO problem on its own — which is certainly possible since the simulator is computationally unbounded. The hybrid argument that the simulator generates an indistinguishable view is also the same as in the proof of Section 4, except that hybrid 10 (use CDC solver to check client-to-server message) is replaced by a statistical argument that the "bad event" MultiIndicesSvr happens with negligible probability. This argument is essentially made in the proof of Theorem 5, where

$$\Pr[\mathsf{MultiIndicesSvr}] \leq \ell_{SS}\binom{l_{H_1}}{2}\frac{l_{H_6}}{p-1}.$$ □

By Lemmas 10 and 11, we get

Theorem 11. *SRP is game-based secure if the aSDH problem is hard in $\mathbb{F}_p$.*

7 Discussion: Should You (Continue to) Use SRP?

What do our results say about SRP? Unfortunately there is no definitive answer: there are both positive and negative results. Perhaps this "messiness" is exactly the nature of SRP: it is not the most elegant protocol, not only due to efficiency and implementation issues, but also because a formal security analysis is incredibly difficult. Should SRP be proposed today, it would almost certainly fail to get standardized and/or used in practice, due to the lack of a security proof under standard definitions and standard assumptions. This is, of course, a positive development in cryptography: we have passed the stage where someone could just introduce a new protocol out of thin air and get it adapted in practice, and the importance of formal security analyses (at least for public-key schemes) has been widely acknowledged by the community.

However, given the fact that SRP is currently being used by millions of people, there is still the (inconvenient) question of whether we should continue to use it, or we should sunset SRP and replace it with some "modern" aPAKE protocols. We will refrain from expressing our personal opinions here, and only reiterate what our results imply:

First, we show that your level of confidence in SRP mostly depends on which security notion you believe is the "right" one for aPAKE. For example, if you believe UC-security is the gold standard, then our work provides strong motivation to transition *away* from SRP; whereas if you think game-based security is already good enough, then getting rid of SRP is probably not the most urgent thing to do. By pinpointing the exact security level of SRP, we reduce the technical discussion on whether SRP should still be used to the philosophical question of which aPAKE security notion is the "right" one (which, again, we refrain from commenting on).

Second, our work also provides some guidance to cryptanalysts: namely, the UC-security of SRP is completely reduced to the hardness of aSDH and the very easiness of Multi-message CDC, so anyone who wants to investigate this matter further should focus on these two computational problems — which are much simpler than SRP itself. (This is ignoring the possibility of "breaking" the random oracles, which appears less likely.)

Apart from the main contributions above, we also provide the first correct security proof for SRP, including the concrete bound — which might be another factor in deciding whether to continue using SRP (especially since the concrete bound seems loose due to the forking lemma argument). Some techniques in the proof, as well as the general result that UC-with-angels security implies game-based security, may find applications in other contexts.

Should you trust SRP more or less after reading our work? Now the decision is yours...

Acknowledgement. The authors are partially supported by NSF award 2453883.

References

[ABB+20] Michel Abdalla, Manuel Barbosa, Tatiana Bradley, Stanislaw Jarecki, Jonathan Katz, and Jiayu Xu. Universally composable relaxed password authenticated key exchange. page CRYPTO 2020, 2020.

[ABR01] Michel Abdalla, Mihir Bellare, and Phillip Rogaway. The oracle Diffie-Hellman assumptions and an analysis of DHIES. pages 143–158, 2001.

[AP05] Michel Abdalla and David Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA 2005*, pages 191–208, 2005.

[AP24] Martin R. Albrecht and Kenneth G. Paterson. Analyzing cryptography in the wild: A retrospective. *IEEE Security & Privacy*, 22(6):12–18, 2024.

[App21] Apple. Escrow security for iCloud keychain, 2021. `https://support.apple.com/guide/security/escrow-security-for-icloud-keychain-sec3e341e75d/web`.

[Bar16] Bart Butler. Improved authentication for email encryption and security, 2016. `https://proton.me/blog/encrypted-email-authentication`.

[BMM21] Brandon Broadnax, Jeremias Mechler, and Jörn Müller-Quade. Environmentally friendly composable multi-party computation in the plain model from standard (timed) assumptions. pages 750–781, 2021.

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-key model and a general forking lemma. In *CCS 2006*, pages 390–399, 2006.

[BP13] Fabrice Benhamouda and David Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. Cryptology ePrint Archive, Report 2013/833, 2013.

[CHK+05] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *EUROCRYPT 2005*, pages 404–421, 2005.

[CKS08] David Cash, Eike Kiltz, and Victor Shoup. The twin Diffie-Hellman problem and applications. In *EUROCRYPT 2008*, pages 127–145, 2008.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In *CRYPTO 2003*, pages 265–281, 2003.

[Cry20] Crypto Forum Research Group. PAKE selection, 2020. https://github.com/cfrg/pake-selection.

[DL24] Dennis Dayanikli and Anja Lehmann. Provable security analysis of the secure remote password protocol. In *CSF 2024*, pages 620–635, 2024.

[Fil18] Rick Fillion. Developers: How we use SRP, and you can too: 1Password, 2018. https://blog.1password.com/developers-how-we-use-srp-and-you-can-too.

[GMR06] Craig Gentry, Philip MacKenzie, and Zulfikar Ramzan. A method for making password-based key exchange resilient to server compromise. In *CRYPTO 2006*, pages 142–159, 2006.

[Gre18] Matthew Green. Should you use SRP?, 2018. https://blog.cryptographyengineering.com/should-you-use-srp.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In *EUROCRYPT 2018*, pages 456–486, 2018.

[PS96] David Pointcheval and Jacques Stern. Security proofs for signature schemes. In *EUROCRYPT 1996*, pages 387–398, 1996.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. In *STOC 2004*, pages 242–251, 2004.

[RX23] Lawrence Roy and Jiayu Xu. A universally composable PAKE with zero communication cost (And why it shouldn't be considered UC-secure). In *PKC2023*, pages 714–743, 2023.

[Sch89] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO 1989*, pages 239–252, 1989.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT 1997*, pages 256–266, 1997.

[Wu97] Tom Wu. The SRP project, 1997. http://srp.stanford.edu/project.html.

[Wu98] Thomas D. Wu. The secure remote password protocol. In *NDSS 1998*, pages 97–111, 1998.

[Wu02] Thomas Wu. SRP-6: Improvements and refinements to the secure remote password protocol, 2002. http://srp.stanford.edu/srp6.ps.

A Proof of Lemma 1

Proof. Suppose there are ℓ_{SS} sub-sessions in total, and let $\mathsf{SpuriousGuess}_i$ be the event that $\mathsf{SpuriousGuess}$ first happens in the i-th sub-session.

To prove that $\Pr[\mathsf{SpuriousGuess}_i]$ is negligible, consider an environment $\widetilde{\mathcal{Z}}$ as follows:

1. $\widetilde{\mathcal{Z}}$ samples $\mathsf{pw} \leftarrow \{0,1\}^\lambda$ and initiates S's session by sending $(\texttt{StorePWfile}, sid, \mathsf{pw})$ to S.
2. In the first $i-1$ sub-sessions, $\widetilde{\mathcal{Z}}$ behaves exactly as $\mathcal{Z}$.
3. In the i-th sub-session, $\widetilde{\mathcal{Z}}$ initiates the two protocol parties' instances by sending $(\texttt{CltSession}, sid, ssid, S, \mathsf{pw})$ to C and $(\texttt{SvrSession}, sid, ssid)$ to S. (Note that the two parties' passwords are equal in this sub-session.) After that, $\mathcal{Z}$ instructs the adversary to be passive in this sub-session. When C outputs K_C and S outputs K_S, if $K_C = K_S$, $\widetilde{\mathcal{Z}}$ outputs 1; otherwise, it outputs 0.

By the correctness of the aPAKE protocol, we have

$$\Pr[\widetilde{\mathcal{Z}} \text{ outputs 1 in the real world}] = 1.$$

In the ideal world, SIM's views (note that SIM can be computationally unbounded) while interacting with $\widetilde{\mathcal{Z}}$ and $\mathcal{Z}$ are identical unless and until $\mathsf{SpuriousGuess}_i$ happens, so $\Pr[\mathsf{SpuriousGuess}_i]$ does not change when we consider the interaction with $\widetilde{\mathcal{Z}}$ instead of $\mathcal{Z}$. Suppose $\mathsf{SpuriousGuess}_i$ happens, and SIM sent ℓ $\texttt{OfflineTestPwd}$ and $\texttt{TestPwd}$ commands in total before that. Then when SIM sends $\texttt{TestPwd}$, the only information it knows about pw is the ℓ "wrong guess" answers, which rule out at most ℓ candidate passwords. Therefore, the probability that SIM sends a $\texttt{TestPwd}$ command that results in "correct guess" is at most $1/(2^\lambda - \ell)$. If the $\texttt{TestPwd}$ command (say it is for C) results in "wrong guess", then $\mathcal{F}_{\text{aPAKE}}$ will mark C's instance $\texttt{interrupted}$ and sample K_C uniformly at random from $\{0,1\}^\lambda$ independent of the rest of the game; thus, the probability that $K_C = K_S$ is $1/2^\lambda$. Overall, we have

$$\begin{aligned}
&\Pr[\widetilde{\mathcal{Z}} \text{ outputs 1 in the real world}] \\
\leq &\Pr[\overline{\mathsf{SpuriousGuess}_i}] + \Pr[K_C = K_S \mid \mathsf{SpuriousGuess}_i] \\
\leq &1 - \Pr[\mathsf{SpuriousGuess}_i] + \frac{1}{2^\lambda - \ell} + \frac{1}{2^\lambda}.
\end{aligned}$$

Combined, we get

$$\mathbf{Dist}_{\widetilde{\mathcal{Z}}} \geq \Pr[\mathsf{SpuriousGuess}_i] - \frac{1}{2^\lambda - \ell} - \frac{1}{2^\lambda},$$

and since $\mathbf{Dist}_{\widetilde{\mathcal{Z}}}$ is negligible, $\Pr[\mathsf{SpuriousGuess}_i]$ must be negligible. Then $\Pr[\mathsf{SpuriousGuess}] = \sum_{i=1}^{\ell_{SS}} \Pr[\mathsf{SpuriousGuess}_i]$ is also negligible. □

B Proof of Lemma 5

To begin with, note that the Two-for-One Guessing problem and the aSDH problem are equivalent, with no loss in the reductions (Theorem 3). Thus, while the proofs in Section 5.2 use a Two-for-One Guessing solver, we can freely replace it with an aSDH solver. Below we study aSDH as it is simpler than Two-for-One Guessing.

The "twin Diffie–Hellman test". The crux of our proof of Lemma 5 is the following lemma:

Lemma 12 ([CKS08, Theorem 2])**.** *Let $\mathbb{G}$ be a group of prime order q, and suppose $A \in \mathbb{G}$ follows any distribution. Sample $r, s \leftarrow \mathbb{Z}_q$, and compute $A' := g^s/A^r$. Then*

1. *A' is uniformly random in $\mathbb{G}$, and independent of A;*
2. *For any $B, T, T' \in \mathbb{G}$ that depend on A and A' only (and not r, s), $T^r T' = B^s$ if and only if $T = \mathbf{cdh}(B, A)$ and $T' = \mathbf{cdh}(B, A')$, except with negligible probability.*

Lemma 12 suggests a way to implement a "twin Diffie–Hellman test" in a prime-order group that, given A, A, B, T, T' (where A' is computed as g^s/A^r), tells if $T = \mathbf{cdh}(B, A)$ and $T' = \mathbf{cdh}(B, A')$. Such test can be straightforwardly extended to any group of order $2q$ (e.g., $\mathbb{F}_p^*$ for a strong prime p), by decomposing the group into an order-q subgroup and an order-2 subgroup:

- Test the order-q component using Lemma 12, i.e., check if
$$T^{2r}(T')^2 = B^{2s}.$$
- Test the order-2 component by computing the Jacobi symbols, i.e., check if
$$\mathcal{J}_p(T) = \begin{cases} 1, & \text{if } \mathcal{J}_p(B) = 1 \vee \mathcal{J}_p(A) = 1 \\ -1, & \text{if } \mathcal{J}_p(B) = \mathcal{J}_p(A) = -1 \end{cases}$$
and
$$\mathcal{J}_p(T') = \begin{cases} 1, & \text{if } \mathcal{J}_p(B) = 1 \vee \mathcal{J}_p(A') = 1 \\ -1. & \text{if } \mathcal{J}_p(B) = \mathcal{J}_p(A') = -1 \end{cases}$$

If all checks above pass, then we output 1 (i.e., we decide that $T = \mathbf{cdh}(B, A)$ and $T' = \mathbf{cdh}(B, A')$); otherwise we output 0.

Given an aSDH solver $\mathcal{A}$ (in $\mathbb{F}_p^*$), the above in turn allows us to construct another aSDH solver that runs $\mathcal{A}$ twice and tells whether $\mathcal{A}$ wins *both times* — which we show below.

An aSDH solver that can tell whether itself wins or not.

Lemma 13. *If the aSDH problem is easy in $\mathbb{F}_p$, then there exists a PPT algorithm whose probability of winning the aSDH game is non-negligible, and the algorithm can tell whether it wins or not with overwhelming accuracy.*

Proof. Since aSDH is easy, there exists a PPT $\mathcal{A}$ that wins the aSDH game with non-negligible probability. We construct another $\widetilde{\mathcal{A}}$ in the aSDH game:

1. On Δ from its challenger, $\widetilde{\mathcal{A}}$ invokes $\mathcal{A}$ on Δ.
2. When $\mathcal{A}$ outputs B, $\widetilde{\mathcal{A}}$ outputs the same value to its challenger.
3. On A from its challenger, $\widetilde{\mathcal{A}}$ sends the same value to $\mathcal{A}$.
4. $\mathcal{A}$ outputs (T_1, T_2).
5. $\widetilde{\mathcal{A}}$ invokes $\mathcal{A}$ again using the same random coins and on the same Δ.
6. When $\mathcal{A}$ outputs (the same) B, $\widetilde{\mathcal{A}}$ computes A' as in Lemma 12 and sends A' to $\mathcal{A}$.
7. When $\mathcal{A}$ outputs (T_1', T_2'), $\widetilde{\mathcal{A}}$ checks if

   $$T_1 = \mathbf{cdh}(B, A) \wedge T_1' = \mathbf{cdh}(B, A')$$

   and

   $$T_2 = \mathbf{cdh}(B + \Delta, A) \wedge T_2' = \mathbf{cdh}(B + \Delta, A')$$

   using the method described after Lemma 12. If so, $\widetilde{\mathcal{A}}$ outputs (T_1, T_2) together with bit 1 (indicating that $\widetilde{\mathcal{A}}$ thinks it wins). Otherwise $\widetilde{\mathcal{A}}$ aborts with bit 0.

We now claim that $\widetilde{\mathcal{A}}$ wins with non-negligible probability. If $\mathcal{A}$ wins twice, then the four equations in step 7 hold and thus $\widetilde{\mathcal{A}}$ wins — unless at least one of $\widetilde{\mathcal{A}}$'s two checks in step 7 returns an inaccurate result, which happens with negligible probability by Lemma 12.

Next, we estimate $\Pr[\mathcal{A}$ wins twice$]$. Note that $\mathcal{A}$'s two executions are *not* independent of each other, as they share the first message Δ. However, Lemma 12 ensures that the second messages A and A' are independent of each other, which suffices for our purpose.

Concretely, let $\mathbf{Adv}_{\mathcal{A}|\Delta}$ be a shorthand for $\Pr[\mathcal{A}$ wins $\mid \mathcal{A}$ receives Δ as the first message$]$. Then

$$\mathbf{Adv}_{\mathcal{A}} = \frac{1}{p-1} \sum_{\Delta \in \mathbb{F}_p^*} \mathbf{Adv}_{\mathcal{A}|\Delta}.$$

We have

$$\begin{aligned}\Pr[\mathcal{A}\text{ wins twice}] &= \frac{1}{p-1}\sum_{\Delta\in\mathbb{F}_p^*}\Pr[\mathcal{A}\text{ wins twice} \mid \mathcal{A}\text{ receives }\Delta\text{ as the first message}]\\ &= \frac{1}{p-1}\sum_{\Delta\in\mathbb{F}_p^*}(\mathbf{Adv}_{\mathcal{A}|\Delta})^2\\ &\geq \frac{1}{p-1}\cdot\frac{\left(\sum_{\Delta\in\mathbb{F}_p^*}\mathbf{Adv}_{\mathcal{A}|\Delta}\right)^2}{p-1}\\ &= \frac{1}{p-1}\cdot\frac{[(p-1)\cdot\mathbf{Adv}_{\mathcal{A}}]^2}{p-1} = (\mathbf{Adv}_{\mathcal{A}})^2\end{aligned}$$

(where the second equation is because $\mathcal{A}$'s two runs are independent after Δ is determined), so

$$\mathbf{Adv}_{\widetilde{\mathcal{A}}} \geq (\mathbf{Adv}_{\mathcal{A}})^2 - \mathsf{negl},$$

which is non-negligible.

Finally, if $\widetilde{\mathcal{A}}$ outputs bit 0 then of course it loses, and if $\widetilde{\mathcal{A}}$ outputs bit 1 then it wins unless at least one of its two checks returns an inaccurate result — which happens with negligible probability. Therefore, $\widetilde{\mathcal{A}}$ outputs 1 if and only if it wins, except with negligible probability. □

Lemmas 4 and 13 (also see comments after the proof of Lemma 4) immediately yield Lemma 5.

C Proof of Lemma 7

The idea of this proof is relatively simple, so we only give a sketch.

Proof. Suppose SIM wins the Non-adaptive CDC-RO game with overwhelming probability; we construct a SIM' in the CDC-RO game. SIM' fixes SIM's random coins and runs SIM q times; in the j-th execution SIM' picks index j, and forwards (A, T) from SIM''s own challenger to SIM. (Note that $(B, k, x_1, \cdots, x_q)$ are the same across all q executions of SIM, so SIM' only needs to run once with its own challenger.) If SIM outputs the same index $i \in [0, q]$ all q times, then SIM' outputs i; otherwise SIM' aborts.

To see why SIM' wins with overwhelming probability, consider the following two possibilities:

- If there is a single index i such that $\mathbf{ddh}(B - kg^{x_i}, Ag^{ux_i}, T) = 1$, then among the q executions of SIM by SIM', in the i-th execution SIM' plays the role of a "legal" challenger for SIM, so in this execution SIM outputs i. (Here and below we ignore the negligible probability that SIM loses.)
  In the j-th ($j \neq i$) execution, SIM' is not a "legal" challenger, since the message (A, T) that SIM receives does not satisfy the constraint. However,

the index j chosen at the beginning of the game is unknown to SIM*, and* SIM*'s views in its i-th run (with a "legal" challenger) and in its j-th run (with an "illegal" challenger) are identical.* Thus, in the j-th execution SIM also outputs i.
In other words, SIM outputs the same index i all q times. This causes SIM' to output i, which is the correct output.

- If there is no index i such that $\mathbf{ddh}(B - kg^{x_i}, Ag^{ux_i}, T) = 1$, then SIM outputs 0 all q times. This causes SIM' to output 0, which is the correct output. □