Highly Efficient and Round-Optimal Asymmetric PAKE

Zachary Barbanell and Jiayu Xu

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

Abstract. An asymmetric Password-Authenticated Key Exchange (aPAKE) protocol allows a client, who holds a raw password, and a server, who holds a one-way mapping of the password, to jointly establish a cryptographically strong session key, without an authenticated channel. The standard security definition for aPAKE is in the Universal Composability (UC) framework. Despite its great potential of being used in practice, existing aPAKE protocols are either not round-optimal, computationally inefficient, or not proven secure w.r.t. the standard UC functionality for aPAKE.

In this work, we present two aPAKE protocols that have a single simultaneous round, are UC-secure, have reasonable computational costs, and rely on standard cryptographic assumptions. Our first protocol is in the Random Oracle Model and the Algebraic Group Model (ROM+AGM), secure under the CDH assumption, and has computational cost comparable to the most efficient aPAKE protocols in the literature (single simultaneous round or not). Our second protocol is in the ROM only and secure under the DDH assumption, and also has significantly lower computational cost than the only other UC-secure single simultaneous round aPAKE protocol. Interestingly, even our second protocol does not rely on the gap Diffie–Hellman assumption, which requires creative application of the forking lemma.

Keywords: Password-authenticated key exchange · Universal composability

1 Introduction

A *Password-Authenticated Key Exchange (PAKE)* protocol allows two parties to jointly establish a cryptographically strong session key, in the setting where the only information shared in advance is a low-entropy password. Crucially, a PAKE protocol must be secure against man-in-the-middle adversaries that can arbitrarily modify protocol messages, and the only feasible attack should be the inevitable online guessing attack, where the adversary impersonates one party using a password guess and communicates with the other party. Starting from the seminal work of Encrypted Key Exchange [7], PAKE has been extensively studied in the literature.

This work is about *asymmetric PAKE (aPAKE)*, an extension of PAKE in the setting where one party (the client) holds the raw password pw and the other

(the server) holds a one-way mapping of the password $f(\mathsf{pw})$ called the *password file*. This requires an adversary to perform an offline dictionary attack to recover the password (or impersonate the client) even if it compromises the server, which is a crucial security property in client-server authentication in practice. The standard security definition — which is adopted in almost all works on aPAKE — is in the Universal Composability (UC) framework, which guarantees security even under arbitrary composition and naturally models password reuse; we defer a detailed description and explanation to Section 2.4.

Single simultaneous round aPAKE: why is it so difficult? Despite its great potential for being used in real life, aPAKE remains a somewhat understudied topic. The most computationally efficient aPAKE protocol to date, OKAPE [29], has 2 message flows (where the server sends the first message). Achieving round optimality — a single round where the two parties' messages can be sent simultaneously — is of both theoretical and practical interest; unfortunately, it seems extremely difficult to build a single simultaneous round aPAKE, and all existing constructions have severe deficiencies in either security or efficiency. Below we summarize existing aPAKE construction ideas in the literature, and explain where the challenges come from:

1. A straightforward idea is to let the server's password file be $H(\mathsf{pw})$, and run a PAKE using $H(\mathsf{pw})$ as the password. This is not a secure aPAKE on its own, as an adversary that compromises the server can impersonate the *client* without an offline dictionary attack. In order to prevent this, it is necessary for the client to additionally prove that it knows pw, not just $H(\mathsf{pw})$.
   This is the idea behind [14,20], in which the client performs an interactive zero-knowledge proof of knowledge to show that it knows the password used to generate the server file. The interactivity of the proofs require at least a second message flow, and it is impossible to use a non-interactive proof with this approach since the statement being proved is about the server file, which must be hidden from an eavesdropper.[1]
2. Alternatively, we could let the two parties agree upon a low-entropy secret that can be derived by the client only with knowledge of pw, and then add a PAKE on top of it to derive a high-entropy session key. For example, in [27], the server stores $V = g^{H(\mathsf{pw})}$ and sends to the client $R \coloneqq g^r$ for a random integer r; then the two parties compute the low-entropy secret $R^{H(\mathsf{pw})} = V^r$ and enter it into a PAKE as the password. This adds a server-to-client message flow before the PAKE and thus cannot lead to a single simultaneous round aPAKE.
3. Another idea is to use an Authenticated Key Exchange (AKE), rather than PAKE, to construct an aPAKE. This is the approach taken by [24,15,29], where the server sends an encryption of the client's AKE credentials, and

[1] Our protocol uses a somewhat similar approach, but rather than proving knowledge of the password used to generate the server file, we prove knowledge of the password used to generate the *client's protocol message*, which allows using a NIZK since the statement of the proof can be public.

then (after client decryption) the two parties run an AKE and derive the session key. Again this requires a client-to-server message flow before the AKE and cannot be done in a single simultaneous round.

One might argue that a 2-flow aPAKE protocol where the client sends the first message is almost as good as a single simultaneous round protocol in practice (although the latter is of more theoretical significance). However, none of the above ideas yield such a protocol either: in (1) the interactive proof of knowledge requires the client to receive the server's challenge before responding, and in (2) (resp. (3)) the client must wait for the server's message before sending its PAKE (resp. AKE) message. At best one can construct a 2-flow protocol where the server initiates.

4. Finally, one could use the "encrypt-and-Smooth Projective Hash Function (SPHF)" approach for aPAKE. This is a major paradigm for (a)PAKE construction, and indeed can be used to build a single simultaneous round aPAKE protocol [8,25][2] or a 2-flow aPAKE protocol where the client initiates [9]. However, the former protocol is only proven game-based secure (and is likely not UC-secure), and both protocols are computationally inefficient compared to other aPAKE protocols (this is especially the case for [25], which uses a pairing group). Indeed, a significant drawback of the "encrypt-and-SPHF" approach for both PAKE and aPAKE is its inefficiency; this is especially true if we want to achieve UC-security, as the protocol messages need to additionally include a NIZK to allow for extraction of the adversary's password guess.

The remaining two single simultaneous aPAKE protocols are SPAKE2+ and aEKE, which we explain in more detail below.

SPAKE2+ and aEKE. SPAKE2+ and aEKE are two instances of single simultaneous round aPAKE protocols that are computationally efficient; however, *neither of them has been proven secure (with respect to the standard UC aPAKE functionality).* The original SPAKE2+ paper [11, Section 9] does not include a security proof, and [31] later gave a UC-security proof for a 3-message flow variant called KC-SPAKE2+ (with the security of the original single simultaneous round protocol still unclear). The original aEKE paper [29] claims the UC-security of aEKE, but in the EUROCRYPT conference talk for this paper [30], it was revealed that aEKE is actually insecure (at least with respect to the standard UC aPAKE functionality).

In fact, SPAKE2+ is also UC-insecure for the same reason: The client-to-server message can be generated using the server file only, and the password is only used when deriving the session key. This means that after compromising the server, an adversary can impersonate the client and send its message to the server; later, whenever the adversary succeeds in an offline dictionary attack, it

[2] Another single simultaneous round aPAKE protocol using SPHF [26] was later found to be insecure (see the end of [8, Section 1]).

can compute the server's session key — *even after the session completes*. This violates the forward secrecy of the protocol.[3]

In sum, there are only three single simultaneous round aPAKE protocols in the literature that have a security proof: [8] that is unlikely to be UC-secure, [11] that only realizes a relaxed UC functionality, and [25] whose computational cost is huge. Given this state of the art, a natural question arises:

> *Is there an aPAKE protocol that has a single simultaneous round, is computationally efficient, and is UC-secure w.r.t. the standard UC aPAKE functionality?*

1.1 Our Contributions

In this work, we present two aPAKE protocols that answer the preceding question in the affirmative:

- Both protocols have a single simultaneous round;
- Both protocols realize the standard UC aPAKE functionality, without any relaxations;
- Regarding computational costs, our first protocol is comparable to the most efficient aPAKE protocols in the literature, while our second protocol is also more efficient than the existing single simultaneous round UC aPAKE protocol in the literature;
- Both protocols rely on standard cryptographic assumptions (CDH and DDH, respectively).

Regarding idealized models, it has been shown that any UC aPAKE protocol must rely on the (programmable) Random Oracle Model (ROM) [19]. Our first protocol additionally relies on the Algebraic Group Model (AGM) [13], which has been previously used in the UC-security proofs of (a)PAKE protocols [2,27]. Interestingly, even our non-AGM-based protocol does not need the Gap Diffie–Hellman (GDH) assumption commonly seen in (a)PAKE security analyses (see, e.g., [1]), which is achieved via creative application of the forking lemma — a contribution on proof techniques that we believe might find further applications elsewhere.

Extension to strong aPAKE. A *strong* aPAKE (saPAKE) protocol additionally requires that all work in an offline dictionary attack must be performed *prior to* server compromise. Neither of our aPAKE protocols is an saPAKE; however, both can be converted to an saPAKE using the general compiler in [24, Section 4] (which adds one message flow).

For a detailed comparison between our protocols and existing (s)aPAKE protocols in the literature, see Table 1. Below we provide a high-level overview of the design ideas of our protocols.

[3] SPAKE2+ has been proven to realize a relaxed UC aPAKE functionality that does not imply perfect forward secrecy [17]; no one has explored the potential relaxed UC-security of aEKE. Here we stress that this is a "real" attack that breaks some desired security properties of aPAKE, rather than merely breaking some UC formalism.

scheme	rounds[1]	client[2]	server[2]	security	assumptions	idealized models
aEKE with HMQV	1	1E+1M	1E+1M	(insecure?)[3]	GDH	RO/IC[6]
SPAKE2+	1	1E+2M	1E+2M	relaxed UC[3]	CDH	RO
BP13	1	4E+6M	3E+5M	game-based	DDH	none
Our Protocol 1	1	1E+2M	2E+1M[4]	UC	CDH	RO, AGM
Our Protocol 2	1	5E+2M	2E+3M[4]	UC	DDH	RO/IC[6]
JR18	1	large[5]	large[5]	UC	SXDH, MDDH	RO
OKAPE with HMQV	2(S)	1E+1M	1E+1M	UC	GDH	RO/IC[6]
HJK+18 with EKE	2(S)	3E	2E+1M	UC	CDH	RO
AuCPace [16,4]	2(S)	3E	4E	relaxed UC	sCDH, sSDH	RO
KHAPE with HMQV	3(S)	1E+1M	1E+1M	UC	GDH	RO/IC[6]
KC-SPAKE2+	3(C)	3M	3M	UC	CDH	RO
GMR06 with EKE+Schnorr signature	3(C)	3E	2E+1M	UC	CDH	RO
SRP[7] [32]	3(C)	3E	2E+1M	angel-based UC	aSDH	RO
Owl [18]	3(C)	5E+5M	3E+3M	game-based	DDH	RO
MX23[8]	2(S)	3E	4E	UC	CDH	RO, GGM
OPAQUE[8]	3(C)	3E+1M	2E+1M	relaxed UC	OMGDH	RO
BJX19[8]	2(C)	2E+5M	3M	UC	2-SDH, DDH	RO, GGM
Our Protocol 1 with JKX18 compiler[8]	2(C)	3E+2M	3E+1M	relaxed UC	OMGDH	RO, AGM

(1) #(C) and #(S) denote number of message flows when the client or server, respectively, sends the first message. "1" denotes a single simultaneous round. (2) E and M denote single and multi-exponentiation, respectively. All multi-exponentiations in this table are two- or three-base multi-exponentiations. The computation costs of two- or three-base multi-exponentiations are roughly 1.17 or 1.25 single exponentiations, respectively.[4](3) Both protocols are unlikely to be UC-secure in the standard sense due to an attack on forward secrecy, although SPAKE2+ has been shown to realize a relaxed UC functionality. See text for more discussion. (4) These can be reduced further by using multi-exponentiations with more than three bases, see the full version for more details (5) JR18 has significantly greater computation costs than any other protocol listed here, partially due to its use of a pairing group. A concrete comparison is difficult since the costs depend on the parameters of the security assumption and other specifics. (6) These protocols use an IC (over a group), which can be constructed from an RO [12]. (7) SRP operates in a field and thus cannot be implemented over elliptic curves, so it is difficult to compare its computational cost with other protocols. (8) We also include three *strong* aPAKE protocols: OPAQUE [24] that is the most well-known saPAKE protocol; MX23 [27] that is the most efficient saPAKE to date; and BJX19 [9] that is the only two-flow saPAKE (or aPAKE) protocol where the client sends the first message. We compare them with the saPAKE protocol derived by plugging our first aPAKE protocol into the compiler in [24] (see text for more details). No single simultaneous round saPAKE protocol has been constructed.

Table 1. Comparison of our protocols and existing aPAKE protocols

[4] These values are based off of the use of Shamir's trick, which is what is used for similar performance analysis in prior works [24,29].

1.2 Technical Overview

Our AGM-based protocol. The starting point for our first protocol is SPEKE, which is a PAKE protocol introduced in [21]: Using a hash function f, the client C and the server S compute and send $A := f(\mathsf{pw})^a$ and $B := f(\mathsf{pw})^b$ respectively, and both parties output $SK = f(\mathsf{pw})^{ab}$ (as B^a and A^b respectively) as the session key. This protocol can be naïvely converted to an aPAKE by having the server store $f(\mathsf{pw})$, but the resulting protocol is insecure: while the server file successfully hides pw until an offline dictionary attack is performed, an adversary can impersonate the client using a stolen server file without knowing the password.

To fix this, we require the C-to-S message to include a NIZK demonstrating that it knows the derivation of $f(\mathsf{pw})$. This is impossible if f is implemented as a random oracle, so we need a more structured function. The most straightforward option is to use $f(x) = g^{H(x)}$ for an RO H, which would allow the C-to-S message to include a Schnorr proof to demonstrate that C knows the discrete logarithm of A. Unfortunately, as mentioned in [22], this choice of f breaks the security of the underlying PAKE: an adversary can transform the protocol messages into messages for another password pw' because

$$\log_{f(\mathsf{pw})} f(\mathsf{pw}') = \frac{H(\mathsf{pw}')}{H(\mathsf{pw})},$$

so the session key of either party has low entropy to a man-in-the-middle adversary.

However, by introducing a second independently chosen group generator to the CRS, we can remove this unwanted homomorphism, at the cost of making the proof slightly more complicated: Using $f(\mathsf{pw}) = g_1^{H(\mathsf{pw})} g_2$ avoids this issue, since $\log_{f(\mathsf{pw})} f(\mathsf{pw}')$ cannot be efficiently computed by an adversary which does not know $\log_{g_1} g_2$. The NIZK used on the C-to-S message is a straightforward extension of the standard Schnorr sigma protocol in which the C shows knowledge of a representation of A with respect to g_1 and g_2, first published in [28], which is only marginally more expensive to compute.

Necessity of the AGM. It is worth explaining exactly why our first protocol requires the AGM to achieve UC-security. For a UC-secure (a)PAKE protocol, the simulator needs to be able to extract a password guess from the messages sent by each party. In SPEKE, the messages are independent of the password, so it is impossible for even an unbounded simulator to extract a password guess from a protocol message alone. Despite this, a password guess can be extracted when given the *algebraic representation* of a message, so requiring adversaries to be algebraic allows the simulation to go through. Several similar PAKE schemes, SPAKE2 and CPace, are proven UC-secure this way [2], and it appears that SPEKE could similarly be shown UC-secure in the ROM+AGM.

Our change to the SPEKE hash function f preserves independence of messages from password and still allows the extraction of password guesses from

the representation (although in a different way), and the NIZK added does not affect either property, so we can similarly achieve UC-security only in the AGM.

We note that the AGM has been used in the security proofs for other aPAKE protocols such as [27], and although not UC-secure, SPAKE2+ also has protocol messages which are independent of the password and thus would require the AGM to show UC-security in a relaxed functionality (see Footnote 3).

Our non-AGM-based protocol. As discussed above, in our AGM-based protocol both messages are independent of the password, so to make a version which is secure outside of the AGM we need to make each party's message distribution depend on the password (while computationally hiding it). In the S-to-C direction, this is relatively straightforward to fix by encrypting the message using an ideal cipher, similarly to EKE, but this does not work in the C-to-S direction since an eavesdropper could determine the password by decrypting the ciphertext under various candidate passwords and checking if the NIZK is valid.

To resolve this, we use a more complicated NIZK in the C-to-S message, based on the method described in [10], which makes the password extractable from the message by using DL trapdoors held by the simulator. In our AGM-based protocol, the NIZK proves knowledge of some α, β such that the group element in the C-to-S message $A = g_1^\alpha g_2^\beta$, and although it is impossible for a party to know two different such representations without knowing $\log_{g_1} g_2$, there is no way to determine which one is known from the NIZK. To fix this, we require the client to send additional group elements A', A''[5] and prove that it knows a representation of A which is related to the representations of A', A''. In particular, the client sends group elements A, A', A'' along with a proof of knowledge of some α, β, γ satisfying

$$A = g_1^\alpha g_2^\beta, A' = g_1^{\beta+\gamma}, A'' = g_3^\gamma,$$

and since there exists exactly one such tuple of α, β, γ for any A, A', A'', this commits to a specific representation of A. Note that while this proof is still zero-knowledge, the attested relationship between the client's known representation of A and the logarithms of A' and A'' can potentially be used by an adversary to break security, so care needs to be taken to prevent this — any (linear) relationship between the representation of A and a single other group element would break the security of the protocol.[6] While a simulator without rewinding still cannot extract the value of α or β from this proof, it can use DL trapdoors $\log_{g_1} g_2$ and $\log_{g_1} g_3$ along with A' and A'' to test guesses for the value of $\frac{\alpha}{\beta}$ (and thus $H(\mathsf{pw})$), and thereby extract a password guess from $\mathcal{A}$'s C-to-S message.

[5] In the actual protocol, A, A', A'' are called A_1, A_3, A_5, respectively.

[6] For instance, if the client sends A, A' and a proof that $A = g_1^\alpha g_2^\beta$ and $A' = g_1^\beta$, an adversary can compute $A^* = A(A')^{H(\mathsf{pw}')-H(\mathsf{pw})}$ to transform the message into one for another password pw'. If an independent base g_3 is chosen for A', we do not know of a "real" attack but the protocol is not UC-secure, because an adversarial server can send an encryption of $(g_1^{H(\mathsf{pw})} g_2)^b g_3^{b'}$ as its message, which the simulator cannot correctly respond to.

2 Preliminaries

Let λ be the security parameter. Let $\mathsf{GenGroup}$ be a PPT algorithm that, on input 1^λ, outputs $(\mathbb{G}, p)$ where $\mathbb{G}$ is a group of prime order p.[7] Let $\bar{\mathbb{G}} = \mathbb{G} \setminus \{e\}$ be the set of all generators of $\mathbb{G}$.

For any $X \in \bar{\mathbb{G}}, Y \in \mathbb{G}$, there is a unique $\alpha \in \mathbb{Z}_p$ such that $X^\alpha = Y$, which we write as $\log_X Y$.

2.1 The Algebraic Group Model

The Algebraic Group Model (AGM) [13] is an idealized model where the adversary $\mathcal{A}$ is required to be *algebraic*, i.e., when $\mathcal{A}$ outputs a group element X, it must also provide X's *algebraic representation*

$$X = X_1^{\alpha_1} \cdots X_n^{\alpha_n},$$

where $X_1, \ldots, X_n$ are group elements in $\mathcal{A}$'s current view. When using the AGM in the UC framework, we require both the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$ to be algebraic; this in particular allows the UC simulator to see the algebraic representations when $\mathcal{Z}$ instructs $\mathcal{A}$ to send a protocol message (see [2] for more details).

2.2 Cryptographic Assumptions

The Discrete Logarithm (DL) assumption. Let $\mathbf{Adv}^{\mathrm{DL}}_{\mathcal{A}}$ be the probability that algorithm $\mathcal{A}$, given $X, Y \leftarrow \bar{\mathbb{G}}$, outputs $\log_X Y$. The DL assumption states that for any PPT algorithm $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{DL}}_{\mathcal{A}}$ is negligible.

Note that we use the "variable-base" version of DL, where the discrete logarithm has uniformly random base X (rather than a fixed base g that is part of the group description). Also, the challenge Y is sampled from $\bar{\mathbb{G}}$ rather than $\mathbb{G}$. This is equivalent to the standard version of DL.

The Computational Diffie–Hellman (CDH) assumption. Let $\mathbf{Adv}^{\mathrm{CDH}}_{\mathcal{A}}$ be the probability that algorithm $\mathcal{A}$, given $X, Y, Z \leftarrow \bar{\mathbb{G}}$, outputs $W_1, \ldots, W_q \in \bar{\mathbb{G}}$ such that $\log_X Y = \log_Z W_i$ for some $i \in [1, q]$. The CDH assumption states that for any PPT algorithm $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{CDH}}_{\mathcal{A}}$ is negligible.

For convenience, our definition of CDH allows the adversary to output multiple guesses of the correct value, and the adversary succeeds if any guess is correct. This formulation is similar to [6, Section 3], and it can be reduced to the more standard, single-output version of CDH with a loss of q by simply choosing a random W_i to output.

The CDH assumption has been shown to be equivalent to the DL assumption in the AGM [13, Section 3.1].

[7] We take the convention that $\mathsf{GenGroup}$ does not output a "standard generator". This distinction is relevant in the AGM, but our proofs could easily be adapted to the opposite convention by letting g_1 in both protocols be the standard generator, rather than randomly sampled.

The One Oracle Diffie–Hellman (1ODH) assumption. Let $\mathbf{Adv}^{\mathrm{1ODH}}_{\mathcal{A}}$ be the probability that algorithm $\mathcal{A}$, given $X, Y, Z \leftarrow \bar{\mathbb{G}}$, access to an oracle for a random permutation $H : \bar{\mathbb{G}} \to \bar{\mathbb{G}}$, and access to an oracle $\mathcal{O}(N) \coloneqq H(N^{\log_X Y})$ to which it may only make at most one query, outputs $W \in \bar{\mathbb{G}}$ such that $\log_X Y = \log_Z W$, outputs $W_1, \ldots, W_q \in \bar{\mathbb{G}}$ such that $\log_X Y = \log_Z W_i$ for some $i \in [1, q]$. The 1ODH assumption states that for any PPT algorithm $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{1ODH}}_{\mathcal{A}}$ is negligible.

Note that, unlike the definition of 1ODH in [23, Appendix A.1] and the definition of ODH in [3], this is still a computational problem where the adversary's goal is to output W_i rather than distinguishing $H(W_i)$ from random. This version can be reduced to CDH with a loss of $q_H + 1$, where q_H is the number of H queries by $\mathcal{A}$ ($\mathcal{A}$ may sample $r \leftarrow \mathbb{Z}_p^*$ and check if $\mathcal{O}(X^r) = H(Y^r)$, and the reduction has to make a guess on which of $\mathcal{A}$'s H queries has input Y^r).

The Decisional Diffie–Hellman (DDH) problem. Let $\mathbf{Adv}^{\mathrm{DDH}}_{\mathcal{A}}$ be the distinguishing advantage of algorithm $\mathcal{A}$ between $(X, Y, Z, Z^{\log_X Y})$ and (X, Y, Z, R), for $X, Y \leftarrow \bar{\mathbb{G}}^2$ and $Z, R \leftarrow \mathbb{G}^2$. The DDH assumption states that for any PPT algorithm $\mathcal{A}$, $\mathbf{Adv}^{\mathrm{DDH}}_{\mathcal{A}}$ is negligible. Clearly, DDH is a stronger assumption than CDH.

2.3 Forking Lemma

We present the forking lemma from [5].

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 $h_1, \ldots, h_q \in \mathbf{H}$, outputs an index $I \in [0, q]$ and a side output σ, and let IG be an input-generator algorithm for $\mathcal{W}$. Define $\mathcal{W}$'s* accepting probability *as*

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

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

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

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

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

Then

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

When $\mathcal{W}$'s output index is 0, we slightly abuse notations and say $\mathcal{W}$ outputs 0, rather than $(0, \perp)$.

2.4 UC Functionality for aPAKE

See Figure 1 for the standard UC functionality for aPAKE. Our functionality closely follows the original version in [14, Section 4]; one minor difference is that since our protocols allow parties to abort (by outputting the rejection symbol $\perp$) upon receiving certain malformed messages, in the functionality we add an Abort command that will trigger the same event in the ideal world.[8]

This functionality specifies that the only feasible attacks on the aPAKE protocol are the following.

1. *Online guessing attack*, modeled by TestPwd: The adversary runs one party's algorithm on a password guess and communicates with the other party; if the password guess is correct, the adversary learns the honest party's session key.
2. *Offline dictionary attack*, modeled by OfflineTestPwd: The adversary steals the server's password file and locally checks a password guess against the file.
3. *Impersonation attack*, modeled by Impersonate: The adversary steals the server's password file and uses it to communicate with the client (without knowing the password itself); if the two parties' passwords match, the adversary learns the client's session key.

Conventions. In both the functionality and the protocols, we assume without loss of generality that the two protocol parties' names are included in the session ID *sid*. Also, note that the client C's password is generally denoted as pw', the server S's password is denoted as pw, and the adversary $\mathcal{A}$'s password guess is denoted as pw^*.

[8] In a Diffie–Hellman-type protocol, it is generally implicitly assumed that the protocol parties check if their received messages are in the group, and abort if not. Therefore, the UC aPAKE functionality should include the Abort command anyway. Our version of the functionality only allows Abort to be sent on a fresh protocol instance, so the validity of a message must be determinable entirely from public information, but it could be reasonable to allow Abort to be sent to other instances as well.

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

Registration
Upon receiving $(\mathsf{StorePwdfile}, sid, \mathsf{C}, \mathsf{pw})$ from S, if $\mathsf{password}[sid]$ does not exist, set $\mathsf{password}[sid] := \mathsf{pw}$ and $\mathsf{fileStatus}[sid] :=$ uncompromised.

Stealing Password Files
Upon receiving $(\mathsf{StealPwdfile}, sid)$ from $\mathcal{A}^*$, if $\mathsf{password}[sid]$ does not exist, return "no password file". Otherwise, set $\mathsf{fileStatus}[sid] :=$ stolen, and if there exists $\mathsf{offline}[(sid, \mathsf{pw})]$ satisfying $\mathsf{password}[sid] = \mathsf{pw}$, return pw.

Offline Attacks
Upon receiving $(\mathsf{OfflineTestPwd}, sid, \mathsf{pw}^*)$ from $\mathcal{A}^*$, store $\mathsf{offline}[(sid, \mathsf{pw}^*)]$. If $\mathsf{fileStatus}[sid] =$ stolen and $\mathsf{password}[sid] = \mathsf{pw}^*$, return "correct guess"; if $\mathsf{fileStatus}[sid] =$ stolen and $\mathsf{password}[sid] \neq \mathsf{pw}^*$, return "wrong guess". (If $\mathsf{fileStatus}[sid] =$ uncompromised then do not return anything.)

Session Creation
Upon receiving $(\mathsf{ClientSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$ from C, if it is the first $\mathsf{ClientSession}$ message for this $ssid$, send $(\mathsf{ClientSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}^*$ and store $\mathsf{session}[(ssid, \mathsf{C}, \mathsf{S}, \mathsf{pw}')] :=$ fresh.
Upon receiving $(\mathsf{ServerSession}, sid, ssid, \mathsf{C})$ from S, if it is the first $\mathsf{ServerSession}$ message for this $ssid$ and $\mathsf{password}[sid] = \mathsf{pw}$ exists, send $(\mathsf{ServerSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ to $\mathcal{A}^*$ and store $\mathsf{session}[(ssid, \mathsf{S}, \mathsf{C}, \mathsf{pw})] :=$ fresh.

Active Attacks
Upon receiving $(\mathsf{TestPwd}, sid, ssid, \mathsf{P}, \mathsf{pw}^*)$ from $\mathcal{A}$, if $\mathsf{session}[(ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw})] =$ fresh, then:
If $\mathsf{pw} = \mathsf{pw}^*$, set $\mathsf{session}[(ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw})] :=$ compromised and return "correct guess". Otherwise, set $\mathsf{session}[(ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw})] :=$ interrupted and return "wrong guess".
Upon receiving $(\mathsf{Impersonate}, sid, ssid)$ from $\mathcal{A}$, if $\mathsf{fileStatus}[sid] =$ stolen and $\mathsf{session}[(ssid, \mathsf{C}, \mathsf{S}, \mathsf{pw}')] =$ fresh, then:
If $\mathsf{password}[sid] = \mathsf{pw}'$, set $\mathsf{session}[(ssid, \mathsf{C}, \mathsf{S}, \mathsf{pw}')] :=$ compromised and return "correct guess". Otherwise, set $\mathsf{session}[(ssid, \mathsf{C}, \mathsf{S}, \mathsf{pw}')] :=$ interrupted and return "wrong guess".

Key Generation
Upon receiving $(\mathsf{Abort}, sid, ssid, \mathsf{P})$ from $\mathcal{A}$, if $\mathsf{session}[(ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw})] =$ fresh, set $\mathsf{session}[(ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw})] :=$ completed and send $(sid, ssid, \perp)$ to P.
Upon receiving $(\mathsf{NewKey}, sid, ssid, \mathsf{P}, SK \in \{0,1\}^\lambda)$ from $\mathcal{A}$, let $\mathsf{status} := \mathsf{session}[(ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw})]$ if it exists. Then:

- If $\mathsf{status} =$ compromised, send $(sid, ssid, SK)$ to P.
- If $\mathsf{status} =$ fresh, and P' received $(sid, ssid, SK')$ at a time when $\mathsf{session}[(ssid, \mathsf{P}', \mathsf{P}, \mathsf{pw})] =$ fresh held, send $(sid, ssid, SK')$ to P.
- If $\mathsf{status} =$ interrupted or if $\mathsf{status} =$ fresh but the condition in the prior bullet was not met, then sample $SK' \leftarrow \{0,1\}^\lambda$ and send $(sid, ssid, SK')$ to P.

If any of the above were satisfied, set $\mathsf{session}[(ssid, \mathsf{P}, \mathsf{P}', \mathsf{pw})] :=$ completed.

Fig. 1. UC aPAKE functionality $\mathcal{F}_{\text{aPAKE}}$

3 Security Analysis of Our AGM-Based Protocol

3.1 UC Description of Our Protocol

Public Parameters =
$(\mathbb{G}, p, g_1, g_2, H_1 : \{0,1\}^* \to \mathbb{Z}_p, H_2 : \{0,1\}^* \to \mathbb{Z}_p, H_3 : \{0,1\}^* \to \{0,1\}^\lambda)$

Client(pw')		Server($\mathsf{C}, V \coloneqq g_1^{H_1(\mathsf{pw})} g_2$)
$\pi' \coloneqq H_1(\mathsf{pw}') \in \mathbb{Z}_p$		
$a \leftarrow \mathbb{Z}_p^*; r_1, r_2 \leftarrow \mathbb{Z}_p$		
$A_1 = (g_1^{\pi'} g_2)^a, A_2 = g_1^{r_1} g_2^{r_2}$		$b \leftarrow \mathbb{Z}_p^*$
$d = H_2(A_1, A_2)$		$B = V^b$
$s_1 = \pi' a + d r_1 \pmod p$		
$s_2 = a + d r_2 \pmod p$		
	$\xleftarrow{B}$	
	$\xrightarrow{A_1, A_2, s_1, s_2}$	
		$d_c \coloneqq H_2(A_1, A_2)$
$K' \coloneqq B^a \quad (= (g_1^\pi g_2)^{ab})$		If $g_1^{s_1} g_2^{s_2} \neq A_1 A_2^{d_c}$ abort
$SK' \coloneqq H_3(A_1, B, K')$		$K \coloneqq A_1^b \quad (= (g_1^{\pi'} g_2)^{ab})$
		$SK \coloneqq H_3(A_1, B, K)$

Fig. 2. Our AGM-based aPAKE protocol

A graphic illustration of our protocol is shown in Figure 2, with a formal UC description in Figure 3.

UC conventions. We omit *sid* in H_1's inputs and $(sid, ssid)$ in H_2 and H_3's inputs when they are clear from the context; we stress that this is for brevity only and session IDs must be included as part of RO inputs to achieve domain separation.

Public parameters. Security of the protocol relies on g_1 and g_2 being independently uniformly sampled from $\bar{\mathbb{G}}$. This requires a common random string, which is a weaker assumption than RO: we could instead equivalently add another random oracle $H_4 \colon \{0,1\}^* \to \bar{\mathbb{G}}$ and define $g_1 \coloneqq H_4(1)$ and $g_2 \coloneqq H_4(2)$, although this would require additional AGM bookkeeping in the simulator.

Correctness. Suppose $\mathsf{pw}' = \mathsf{pw}$ (i.e., the two parties' passwords match) and there is no adversary. Then

– S's check passes: This is because

$$g_1^{s_1} g_2^{s_2} = g_1^{\pi' a + d r_1} g_2^{a + d r_2} = (g_1^{\pi'} g_2)^a \cdot (g_1^{r_1} g_2^{r_2})^d = A_1 A_2^d,$$

Public parameters

Cyclic group $\mathbb{G}$ with order p, generated by algorithm $\mathsf{GenGroup}(1^\lambda)$; random group generators $g_1, g_2 \leftarrow \bar{\mathbb{G}}$; hash functions (modeled as ROs) H_1, H_2, H_3 whose ranges are $\mathbb{Z}_p, \mathbb{Z}_p, \{0,1\}^\lambda$, respectively.

Password Registration

On input $(\mathsf{StorePwdfile}, sid, \mathsf{C}, \mathsf{pw})$, S computes $V \coloneqq g_1^{H_1(\mathsf{pw})} g_2$ and stores $\mathsf{file}[sid] \coloneqq (\mathsf{C}, V)$.

Login

1. On input $(\mathsf{ClientSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$, C samples $a \leftarrow \mathbb{Z}_p^*$ and $r_1, r_2 \leftarrow \mathbb{Z}_p$, computes $\pi' \coloneqq H_1(\mathsf{pw}')$, $A_1 \coloneqq (g_1^{\pi'} g_2)^a$, $A_2 \coloneqq g_1^{r_1} g_2^{r_2}$, $d \coloneqq H_2(A_1, A_2)$, $s_1 \coloneqq \pi' a + d r_1$, $s_2 \coloneqq a + d r_2$, and sends $(sid, ssid, A_1, A_2, s_1, s_2)$ to S.
   On receiving $(sid, ssid, B)$ from S, if $B \notin \bar{\mathbb{G}}$, C outputs $(sid, ssid, \perp)$. Otherwise, C computes $K' \coloneqq B^a$ and $SK' \coloneqq H_3(A_1, B, K')$, and outputs $(sid, ssid, SK')$.
2. On input $(\mathsf{ServerSession}, sid, ssid)$, if there is a record $\mathsf{file}[sid] = (\mathsf{C}, V)$, S samples $b \leftarrow \mathbb{Z}_p^*$, computes $B \coloneqq V^b$, and sends $(sid, ssid, B)$ to C.
   On receiving $(sid, ssid, A_1, A_2, s_1, s_2)$ from C, if $A_1 \notin \bar{\mathbb{G}}$ or $g_1^{s_1} g_2^{s_2} \neq A_1 A_2^{H_2(A_1, A_2)}$, S outputs $(sid, ssid, \perp)$. Otherwise, S computes $K \coloneqq A_1^b$ and $SK \coloneqq H_3(A_1, B, K)$, and outputs $(sid, ssid, SK)$.

Fig. 3. UC description of our AGM-based aPAKE protocol

CRS generation, RO queries, and algebraic representations

0. SIM runs $(\mathbb{G}, p) \leftarrow \mathsf{GenGroup}(1^\lambda)$, samples $g_1, g_2 \leftarrow \bar{\mathbb{G}}$, and outputs $\mathbb{G}, p, g_1, g_2$ as the public parameters. Furthermore, SIM stores $\langle \mathsf{Factorization}, g_1, 1, 0\rangle$ and $\langle \mathsf{Factorization}, g_2, 0, 1\rangle$.
   SIM answers H_2 and H_3 queries via lazy sampling, H_1 queries are described below. If two H_1 queries with the same sid give the same result, if the result of an H_1 query matches the value of π sampled for the stolen password file for the same sid, or if the result of an H_1 query or sampled π value satisfies $g_1^\pi g_2 = e$, SIM outputs $\mathsf{Collision}$ and halts.
   Whenever SIM defines a group element H which is one of V, A_1, A_2 or B, it stores $\langle \mathsf{Factorization}, H, \alpha, \beta\rangle$ for some specific α, β satisfying $H = g_1^\alpha g_2^\beta$. In particular, for V, $(\alpha, \beta) \coloneqq (\pi, 1)$, and for A_1, A_2, and B, they are $(x, 0)$, (r_1, r_2), and $(y, 0)$, respectively. // SIM stores the same representation of V that $\mathcal{A}$ can compute on its own (after querying $H_1(\mathsf{pw})$). For A_1, A_2, and B, the choice of representation is arbitrary as $\mathcal{A}$ cannot compute any representation of these.
   Finally, when $\mathcal{A}$ sends a group element G with algebraic representation $\prod H_i^{k_i}$, and there exist records $\langle \mathsf{Factorization}, H_i, \alpha_i, \beta_i\rangle$, SIM replaces its representation with the simplified "canonical" form $g_1^\alpha g_2^\beta$, where $\alpha = \sum k_i \alpha_i$ and $\beta = \sum k_i \beta_i$.

Offline attacks

1. On receiving $(\mathsf{StealPwdfile}, sid)$ from $\mathcal{A}$ to S, SIM sends $(\mathsf{StealPwdfile}, sid)$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
   (a) If the response is "no password file", SIM returns "no password file".
   (b) If the response is pw, then there must have been an $(\mathsf{OfflineTestPwd}, sid, \mathsf{pw})$ command, which means that there has been an $H_1(\mathsf{pw})$ query. Let $\pi \coloneqq H_1(\mathsf{pw})$. SIM computes $V \coloneqq g_1^\pi g_2$, stores $\langle \mathsf{PWRecord}, sid, \mathsf{pw}, \pi, V\rangle$, and outputs V. // pre-computation attack
   (c) Otherwise, SIM samples $\pi \leftarrow \mathbb{Z}_p$, computes $V \coloneqq g_1^\pi g_2$, stores $\langle \mathsf{PWRecord}, sid, \bot, \pi, V\rangle$, and outputs V.
2. On RO query $H_1(\mathsf{pw})$, SIM sends $(\mathsf{OfflineTestPwd}, sid, \mathsf{pw})$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
   (a) If the response is "correct guess", then $\mathcal{Z}$ must have sent $(\mathsf{StealPwdfile}, sid)$ previously, so there must be a record $\langle \mathsf{PWRecord}, sid, \cdot, \pi, V\rangle$. SIM modifies the record to $\langle \mathsf{PWRecord}, sid, \mathsf{pw}, \pi, V\rangle$, defines $H_1(\mathsf{pw}) \coloneqq \pi$, and outputs π. // $\mathcal{A}$ compromises S and then performs a successful offline attack; SIM programs H_1 to match π used while generating password file
   (b) Otherwise, SIM samples $\pi \leftarrow \mathbb{Z}_p^*$, defines $H_1(\mathsf{pw}) \coloneqq \pi$, and outputs π.

Simulation of protocol messages

3. On receiving $(\mathsf{ClientSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\mathrm{aPAKE}}$, SIM samples $x \leftarrow \mathbb{Z}_p^*$ and $r_1, r_2 \leftarrow \mathbb{Z}_p$, computes $A_1 \coloneqq g_1^x$, $A_2 \coloneqq g_1^{r_1} g_2^{r_2}$, $d \coloneqq H_2(A_1, A_2)$, $s_1 \coloneqq x + dr_1$, $s_2 \coloneqq dr_2$, stores $\langle \mathsf{ClientSession}, ssid, x, A_1\rangle$, and sends (A_1, A_2, s_1, s_2) from C to S.
4. On receiving $(\mathsf{ServerSession}, sid, ssid, \mathsf{S}, \mathsf{C})$ from $\mathcal{F}_{\mathrm{aPAKE}}$, SIM samples $y \leftarrow \mathbb{Z}_p^*$, computes $B \coloneqq g_1^y$, stores $\langle \mathsf{ServerSession}, ssid, y, B\rangle$, and sends B from S to C.

Fig. 4. Simulator SIM for our AGM-based protocol (part I)

Active attacks

5. On receiving B^* from S to C (together with its algebraic representation), once a record $\langle \mathsf{ClientSession}, ssid, x, A_1 \rangle$ exists,
   (a) If $B^* \notin \bar{\mathbb{G}}$, SIM sends $(\mathsf{Abort}, sid, ssid, \mathsf{C})$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
   (b) If there is a record $\langle \mathsf{ServerSession}, ssid, y, B \rangle$ with $B^* = B$, SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, 0^\lambda)$ to $\mathcal{F}_{\mathrm{aPAKE}}$. // $\mathcal{A}$ does not modify S-to-C message
   (c) Otherwise let the "canonical" algebraic representation of B^* be $g_1^\alpha g_2^\beta$.
      i. If $\alpha = \pi^* \beta$ for some $\pi^* = H_1(\mathsf{pw}^*)$, SIM sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, \mathsf{pw}^*)$ to $\mathcal{F}_{\mathrm{aPAKE}}$. (There must be at most one such pw^* since SIM excludes H_1 collisions in step 0.) // pw^* is $\mathcal{A}$'s password guess
         – If the response is "wrong guess", SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, 0^\lambda)$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
         – If the response is "correct guess", SIM computes $K' := A_1^\beta$ and $SK' := H_3(A_1, B^*, K')$, and sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, SK')$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
      ii. If there is a record $\langle \mathsf{PWRecord}, sid, \bot, \pi, V \rangle$ and $\alpha = \pi\beta$, SIM sends $(\mathsf{Impersonate}, sid, ssid)$ to $\mathcal{F}_{\mathrm{aPAKE}}$. // $\mathcal{A}$ compromises S and executes S's algorithm on its file V
         – If the response is "wrong guess", SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, 0^\lambda)$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
         – If the response is "correct guess", SIM computes $K' := A_1^\beta$ and $SK' := H_3(A_1, B^*, K')$, and sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, SK')$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
      iii. Otherwise, SIM sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, \bot)$ followed by $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, 0^\lambda)$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
6. On receiving $(A_1^*, A_2^*, s_1^*, s_2^*)$ from C to S (together with the algebraic representation of A_1^* and A_2^*), once a record $\langle \mathsf{ServerSession}, ssid, y, B \rangle$ exists,
   (a) If $A_1^* \notin \bar{\mathbb{G}}$ or $A_1^*(A_2^*)^{H_2(A_1^*, A_2^*)} \neq g_1^{s_1^*} g_2^{s_2^*}$, SIM sends $(\mathsf{Abort}, sid, ssid, \mathsf{S})$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
   (b) If there is a record $\langle \mathsf{ClientSession}, ssid, x, A_1 \rangle$ with $A_1^* = A_1$, SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, 0^\lambda)$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
   (c) Otherwise let the "canonical" algebraic representation of A_1 be $g_1^\alpha g_2^\beta$.
      i. If $\alpha = \pi^* \beta$ for some $\pi^* = H_1(\mathsf{pw}^*)$, SIM sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, \mathsf{pw}^*)$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
         – If the response is "wrong guess", SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, 0^\lambda)$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
         – If the response is "correct guess", SIM computes $K := B^\beta$ and $SK := H_3(A_1^*, B, K)$, and sends $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, SK)$ to $\mathcal{F}_{\mathrm{aPAKE}}$.
      ii. If there is a record $\langle \mathsf{PWRecord}, sid, \bot, \pi, V \rangle$ and $\alpha = \pi\beta$, SIM outputs $\mathsf{ClientImpersonation}$ and halts. // $\mathcal{A}$ impersonates client after compromising server, without successful offline attack
      iii. Otherwise, SIM sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, \bot)$ followed by $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, 0^\lambda)$ to $\mathcal{F}_{\mathrm{aPAKE}}$.

Fig. 5. Simulator SIM for our AGM-based protocol (part II)

where $d = H_2(A_1, A_2)$.

– The two parties' session keys are equal: Since $V = g_1^{H_1(\mathsf{pw})} g_2$, $A_1 = (g_1^{H_1(\mathsf{pw}')} g_2)^a$, and $\mathsf{pw}' = \mathsf{pw}$, we have $A_1 = V^a$. Then

$$K' = B^a = (V^b)^a = (V^a)^b = A_1^b = K,$$

so $SK' = SK$.

3.2 Security Proof

Theorem 2. *The protocol in Figure 3 UC-realizes functionality $\mathcal{F}_{aPAKE}$ in the ROM+AGM, assuming that the DL assumption holds in $(\mathbb{G}, p)$.*

Proof. We construct the simulator SIM in Figures 4 and 5. As standard in UC, we assume that the adversary $\mathcal{A}$ is "dummy" and merely passes all messages to and from the environment $\mathcal{Z}$. We use the following conventions: if SIM sends a message m to $\mathcal{Z}$ that pretends to be from party P to P' and intercepted by $\mathcal{A}$, we abbreviate it as "send m from P to P'" — although $\mathcal{A}$ does not exist in the ideal world. Similarly, if SIM receives a message m from $\mathcal{Z}$ that instructs $\mathcal{A}$ to send m to P, we abbreviate it as "on m from $\mathcal{A}$ to P". We assume that $\mathcal{A}$ never repeats an RO query.

Suppose there are q_S sessions (i.e., q_S unique *sid*'s) and q_{SS} sub-sessions (i.e., q_{SS} unique *ssid*'s) in total, and $\mathcal{A}$ makes q_{H_1} and q_{H_2} queries to H_1 and H_2, respectively.

Hybrids. We now show a sequence of hybrids that begins in the real world and ends in the ideal world. For two adjacent hybrids i and $i+1$, we will argue that the distinguishing advantage $\mathbf{Dist}_{\mathcal{Z}}^{i,i+1}$ is negligible for all PPT environments $\mathcal{Z}$.

Hybrid 0 (*real world*). This is the real execution of the protocol. In each sub-session, the flow of events is:

– C sends (A_1, A_2, s_1, s_2) to S, and S sends B to C; both messages are intercepted by $\mathcal{A}$.
– $\mathcal{A}$ sends $(A_1^*, A_2^*, s_1^*, s_2^*)$ to S, causing S to compute K and output SK.
– $\mathcal{A}$ sends B^* to C, causing C to compute K' and output SK'.

(The order of the last two events is decided by $\mathcal{Z}$.)

Below we call a message to S $(A_1^*, A_2^*, s_1^*, s_2^*)$ "valid" if $A_1^* \in \bar{\mathbb{G}}$ and $g_1^{s_1^*} g_2^{s_2^*} = A_1^* (A_2^*)^{H_2(A_1^*, A_2^*)}$, and a message to C "valid" if $B^* \in \bar{\mathbb{G}}$ (i.e., the receiving party does not abort). In all subsequent hybrids, we assume that the message is valid, i.e., we do not make any changes if the message is invalid.

Hybrid 1 (*both parties output the same key if passwords match and $\mathcal{A}$ is passive except in the NIZK*). When either party P receives a message with *ssid*, if its counterparty has already output $(sid, ssid, SK)$ with $SK \neq \perp$, $\mathsf{pw} = \mathsf{pw}'$, $A_1^* = A_1$, and $B^* = B$, then P outputs $(sid, ssid, SK)$ instead of its normal response. (In all subsequent hybrids, we never change this rule and do not repeat it in the hybrid description.)

By correctness of the protocol and the fact that SK does not depend on A_2^*, s_1^*, or s_2^* for valid messages (note that as stated above we do not change behavior for invalid messages), we have

$$\mathbf{Dist}_{\mathcal{Z}}^{0,1} = 0.$$

Hybrid 2 (*halt on H_1 collision or H_1 result is DL trapdoor***).** If two H_1 queries with the same *sid* give the same result, or if the result π^* of an H_1 query satisfies $g_1^{-\pi^*} = g_2$, output $\mathsf{Collision}$ and halt. (In all subsequent hybrids, we assume that $\mathsf{Collision}$ does not happen.)

We have

$$\mathbf{Dist}_{\mathcal{Z}}^{1,2} \leq \Pr[\mathsf{Collision}] \leq \frac{(q_{H_1}+2)(q_{H_1}+1)}{2p} + \frac{q_{H_1}+2q_S}{p} = \frac{q_{H_1}^2 + 5q_{H_1} + 4q_S + 2}{2p},$$

where the "+2" and "$+2q_S$" are due to honest parties' H_1 queries.

Hybrid 3 (*store DL trapdoor***).** When generating public parameters, instead of sampling $g_2 \leftarrow \bar{\mathbb{G}}$, sample (and store) $z \leftarrow \mathbb{Z}_p^*$ and compute $g_2 \coloneqq g_1^z$.

In both Hybrids 2 and 3, (g_1, g_2) is uniformly distributed in $\bar{\mathbb{G}}^2$, so

$$\mathbf{Dist}_{\mathcal{Z}}^{2,3} = 0.$$

Ultimately, the simulator does not need to store the DL trapdoor, so it will be removed in Hybrid 10, but it simplifies the description of the intervening hybrids.

Hybrid 4 (*compute A_1 and B without π or π'***).** Instead of sampling $a \leftarrow \mathbb{Z}_p^*$ and computing $A_1 \coloneqq (g_1^{\pi'} g_2)^a$, C samples $x \leftarrow \mathbb{Z}_p^*$ and computes $A_1 \coloneqq g_1^x$ and $a \coloneqq \frac{x}{\pi'+z}$. Similarly, instead of sampling $b \leftarrow \mathbb{Z}_p^*$ and computing $B \coloneqq V^b$, S samples $y \leftarrow \mathbb{Z}_p^*$ and computes $B \coloneqq g_1^y$ and $b \coloneqq \frac{y}{\pi+z}$.

Both Hybrids 3 and 4 satisfy

$$x = a(\pi' + z)$$

(where $x = \log_{g_1} A_1$), and $\pi' \neq -z$ (the $\pi' = -z$ case was excluded in Hybrid 2). Subject to these conditions, if we sample $a \leftarrow \mathbb{Z}_p^*$ then we get Hybrid 3, and if we sample $x \leftarrow \mathbb{Z}_p^*$ then we get Hybrid 4. The argument for b and y is similar. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{3,4} = 0.$$

Hybrid 5 (*compute integers in protocol messages without π'***).** Instead of computing $s_1 \coloneqq \pi' a + dr_1$ and $s_2 \coloneqq a + dr_2$, C computes $s_1 \coloneqq x + dr_1$ and $s_2 \coloneqq dr_2$. (Note that after this hybrid, a, b are only used while computing the session keys of resp. C, S, so the protocol messages can be computed without π or π'.)

Hybrid 5 satisfies

$$s_1 + zs_2 = x + d(r_1 + zr_2)$$

and thus (consider the exponent of base g_1 on both sides)

$$g_1^{s_1} g_2^{s_2} = A_1 A_2^d,$$

so the C-to-S message is still valid. If A_1, A_2, and s_2 are fixed, there is exactly one value of s_1 which makes the message valid, and the distribution of A_1 and A_2 is unchanged between the hybrids, so it suffices to show that s_2 is identically distributed between the two hybrids.

Fix any values for A_1 and A_2, which also fixes values for $a = \frac{\log_{g_1} A_1}{\pi' + z}$ and $d = H_2(A_1, A_2)$, and consider the distribution of (r_1, r_2). Conditioned on the value of A_2, the possible values for the pair are the solutions to $g_1^{r_1} g_2^{r_2} = A_2$, which take the form $r_1 = \log_{g_1} A_2 - kz, r_2 = k$ for $k \in \mathbb{Z}_p$, all of which are equally likely, so r_2 is uniformly distributed in $\mathbb{Z}_p$.

In both Hybrids 4 and 5, once a and d are fixed, the possible values of s_2 are in bijection with the possible values of r_2, so we in both hybrids, s_2 is uniformly distributed in $\mathbb{Z}_p$. We conclude that the message (A_1, A_2, s_1, s_2) is distributed identically between the hybrids, so

$$\mathbf{Dist}_{\mathcal{Z}}^{4,5} = 0.$$

Hybrid 6 (*express group elements under bases g_1, g_2***).** When public parameters are generated, store $\langle \mathsf{Factorization}, g_1, 1, 0 \rangle$ and $\langle \mathsf{Factorization}, g_2, 0, 1 \rangle$. When S receives StorePwdfile, store $\langle \mathsf{Factorization}, V, \pi, 1 \rangle$. When C sends a message (A_1, A_2, s_1, s_2), store $\langle \mathsf{Factorization}, A_1, x, 0 \rangle$ and $\langle \mathsf{Factorization}, A_2, r_1, r_2 \rangle$. When S sends a message B, store $\langle \mathsf{Factorization}, B, y, 0 \rangle$.

Furthermore, when $\mathcal{A}$ sends a group element G with algebraic representation $\prod H_i^{k_i}$, and there exist records $\langle \mathsf{Factorization}, H_i, \alpha_i, \beta_i \rangle$, replace its representation with the simplified "canonical" form $g_1^{\alpha} g_2^{\beta}$, where $\alpha = \sum k_i \alpha_i$ and $\beta = \sum k_i \beta_i$. In further hybrids, we use "party P receives a message $G = g_1^{\alpha} g_2^{\beta}$" to mean that the "canonical" form of G is $g_1^{\alpha} g_2^{\beta}$.

Hybrid 6 only adds some internal bookkeeping, so

$$\mathbf{Dist}_{\mathcal{Z}}^{5,6} = 0.$$

Hybrid 7 (*compute session keys without secret exponents a, b on correct password guess***).** When C receives a message $B^* = g_1^{\alpha} g_2^{\beta}$, if $\alpha = \pi' \beta$, C computes $K' \coloneqq A_1^{\beta}$ instead of $K' \coloneqq (B^*)^a$. Similarly, when S receives a message with $A_1^* = g_1^{\alpha} g_2^{\beta}$, if $\alpha = \pi \beta$, S computes $K \coloneqq B^{\beta}$ instead of $K \coloneqq (A_1^*)^b$.

If $\alpha = \pi' \beta$ we have

$$B^* = g_1^{\alpha} g_2^{\beta} = g_1^{\alpha + \beta z} = g_1^{\beta(\pi' + z)},$$

so (recall that in Hybrid 4 a is set to $\frac{x}{\pi'+z}$ and A_1 is set to g_1^x)

$$(B^*)^a = g_1^{\beta x} = A_1^\beta.$$

Similarly, $(A_1^*)^b = B^\beta$. Therefore, the values of K and K' are identical between Hybrids 6 and 7, and

$$\mathbf{Dist}_{\mathcal{Z}}^{6,7} = 0.$$

Hybrid 8 (*random session key on forwarded protocol message or incorrect password guess***).** When C receives a message $B^* = g_1^\alpha g_2^\beta$, if $\alpha \neq \pi'\beta$, C outputs $SK' \leftarrow \{0,1\}^\lambda$ instead of computing SK' normally. Similarly, when S receives a message with $A_1^* = g_1^\alpha g_2^\beta$, if $\alpha \neq \pi\beta$, S outputs $SK \leftarrow \{0,1\}^\lambda$ instead of computing SK normally.

Let Query be the event that $\mathcal{A}$ sends $B^* = g_1^\alpha g_2^\beta$ to C with $\alpha \neq \pi'\beta$, and queries $H_3(A_1, B^*, K')$ (call this sub-event $\mathsf{Query}_\mathsf{C}$); or $\mathcal{A}$ sends $(A_1^* = g_1^\alpha g_2^\beta, \dots)$ to S with $\alpha \neq \pi\beta$, and queries $H_3(A_1^*, B, K)$ (call this sub-event $\mathsf{Query}_\mathsf{S}$). Hybrids 7 and 8 are identical unless Query happens. We now construct a reduction $\mathcal{R}_{\text{DL-1}}$ that reduces Query to the DL problem.

$\mathcal{R}_{\text{DL-1}}$, on DL challenge $(X, Y) \in \bar{\mathbb{G}}^2$, sets CRS parameters $g_1 := X, g_2 := Y$, leaving $z = \log_{g_1} g_2$ unknown (which prevents $\mathcal{R}_{\text{DL-1}}$ from computing a or b).

We first rule out the possibility that $\mathcal{A}$ submits two different (canonical) algebraic representations of the same group element $G = g_1^\alpha g_2^\beta = g_1^{\alpha'} g_2^{\beta'}$. If so, then $\log_X Y = z = \frac{\alpha'-\alpha}{\beta-\beta'}$, so $\mathcal{R}_{\text{DL-1}}$ outputs that and halts.

On an $H_3(ssid, \dots)$ query, first suppose that this query triggers $\mathsf{Query}_\mathsf{C}$, i.e., $\mathcal{A}$ queries $H_3(A_1, B^*, K^*)$ where the (canonical) algebraic representations of B^* and K^* are $B^* = g_1^\alpha g_2^\beta$ and $K^* = g_1^{\alpha'} g_2^{\beta'}$, with $\alpha \neq \pi\beta$. (If either C or S have not yet started their algorithms for $ssid$, run their algorithms up to the point that A_1 and B are computed.) This query triggers $\mathsf{Query}_\mathsf{C}$ only if $K^* = (B^*)^a = (B^*)^{\frac{x}{\pi'+z}}$, i.e.,

$$g_1^{\alpha'} g_2^{\beta'} = (g_1^\alpha g_2^\beta)^{\frac{x}{\pi'+z}}$$

(where z is unknown to $\mathcal{R}_{\text{DL-1}}$). Using $g_2 = g_1^z$, we derive

$$\alpha' + z\beta' = \frac{x}{\pi'+z}(\alpha + z\beta),$$

which simplifies to the following equation in z:

$$\beta' z^2 + (\alpha' + \pi'\beta' - x\beta)z + (\pi'\alpha' - x\alpha) = 0. \tag{1}$$

We first argue that if $\mathsf{Query}_\mathsf{C}$ happens (for this $ssid$), then the degree of equation (1) is non-zero. Otherwise

$$\beta' = \alpha' + \pi'\beta' - x\beta = \pi'\alpha' - x\alpha = 0;$$

substituting $\beta' = 0$ into the second equation yields $\alpha' = x\beta$, and substituting $\alpha' = x\beta$ into the third equation yields $\alpha = \pi\beta$ (note that here we need $x \neq 0$),

so $\mathsf{Query}_{\mathsf{C}}$ cannot happen. In this case, $\mathcal{R}_{\text{DL-1}}$ answers the H_3 query via lazy sampling and continues simulating other sub-sessions.

Next, if equation (1) has positive degree, it has up to two solutions. Furthermore, if $\mathsf{Query}_{\mathsf{C}}$ happens, then equation (1) must have at least one solution, since $\log_{g_1} g_2$ is a solution. Therefore, $\mathcal{R}_{\text{DL-1}}$ solves equation (1) and checks if any solution satisfies $g_1^z = g_2$. If so, $\mathcal{R}_{\text{DL-1}}$ outputs that solution and halts. If not, then this H_3 query does not trigger $\mathsf{Query}_{\mathsf{C}}$, so $\mathcal{R}_{\text{DL-1}}$ answers via lazy sampling and continues simulating other sub-sessions.

For $\mathsf{Query}_{\mathsf{S}}$, suppose $\mathcal{A}$ queries $H_3(A_1^*, B, K^*)$ with $A_1^* = g_1^{\alpha} g_2^{\beta}$ and $K^* = g_1^{\alpha'} g_2^{\beta'}$. By a similar argument, $\mathsf{Query}_{\mathsf{S}}$ happens only if

$$\beta' z^2 + (\alpha' + \pi\beta' - y\beta)z + (\pi\alpha' - y\alpha) = 0$$

(where π corresponds to S's password), and if $\mathsf{Query}_{\mathsf{S}}$ happens, $\mathcal{R}_{\text{DL-1}}$ can solve for z.

In sum, if Query happens, then $\mathcal{R}_{\text{DL-1}}$ solves the DL instance (X, Y) correctly. Therefore,

$$\mathbf{Dist}_{\mathcal{Z}}^{7,8} \leq \Pr[\mathsf{Query}] \leq \mathbf{Adv}^{\text{DL}}_{\mathcal{R}_{\text{DL-1}}}.$$

Hybrid 9 (*rule out client impersonation after server compromise*)**.** When S receives a valid message $(A_1^*, A_2^*, s_1^*, s_2^*)$ with $A_1^* = g_1^{\alpha} g_2^{\beta}$, if $\alpha = \pi\beta$ and $\mathcal{A}$ has not queried $H_1(\mathsf{pw})$, output $\mathsf{ClientImpersonation}$ and halt.

Hybrids 8 and 9 are identical unless $\mathsf{ClientImpersonation}$ happens. We now construct a reduction $\mathcal{R}_{\text{DL-2}}$ that reduces $\mathsf{ClientImpersonation}$ to the DL problem.

$\mathcal{R}_{\text{DL-2}}$, on DL challenge $(X, Y) \in \bar{\mathbb{G}}^2$, samples $k \leftarrow [1, q_S]$ (as a guess that $\mathsf{ClientImpersonation}$ first happens in a certain sub-session within the k-th session). When generating CRS parameters, $\mathcal{R}_{\text{DL-2}}$ sets $g_1 := X$. On the k-th $\mathsf{StorePwdfile}$ message, $\mathcal{R}_{\text{DL-2}}$ sets $V := Y$ (below we only describe $\mathcal{R}_{\text{DL-2}}$'s behavior within this session). If $\mathcal{A}$ ever queries $H_1(\mathsf{pw})$ for this session, this implies that $\mathsf{ClientImpersonation}$ does not happen in this session, so $\mathcal{R}_{\text{DL-2}}$ aborts.

Since $\mathcal{R}_{\text{DL-2}}$ does not know the representation of V w.r.t. g_1, g_2, it cannot compute canonical algebraic representations of all group elements as $g_1^{\alpha} g_2^{\beta}$ (as in Hybrid 6); however, $\mathcal{R}_{\text{DL-2}}$ can instead compute and store "sub-canonical" algebraic representations in the form of $g_1^{\alpha} g_2^{\beta} V^{\gamma}$. We first rule out the possibility that $\mathcal{A}$ submits two different such algebraic representations of the same group element $G = g_1^{\alpha} g_2^{\beta} V^{\gamma} = g_1^{\alpha'} g_2^{\beta'} V^{\gamma'}$ with $\gamma \neq \gamma'$. If so, then $\log_X Y = \log_{g_1} V = \frac{\alpha' + z\beta' - \alpha - z\beta}{\gamma - \gamma'}$, so $\mathcal{R}_{\text{DL-2}}$ outputs that and halts.

When $\mathcal{A}$ sends a valid message $(A_1^* = g_1^{\alpha} g_2^{\beta} V^{\gamma}, A_2^* = g_1^{\alpha'} g_2^{\beta'} V^{\gamma'}, s_1^*, s_2^*)$, there are three possibilities:

- $\gamma = 0$: In this case, $\alpha = \pi\beta$ if and only if $V = g_1^{\frac{\alpha}{\beta}} g_2$. If this is false, then this message does not trigger $\mathsf{ClientImpersonation}$, so $\mathcal{R}_{\text{DL-2}}$ sets S's session key $SK \leftarrow \{0,1\}^{\lambda}$, matching the behavior of Hybrid 8 (and continues simulating other sub-sessions). If it is true, $\log_X Y = \log_{g_1} V = \frac{\alpha}{\beta} + z$, so $\mathcal{R}_{\text{DL-2}}$ outputs that and halts.

- $\gamma \neq 0$ but $\gamma + H_2(A_1^*, A_2^*)\gamma' = 0$: $\mathcal{R}_{\text{DL-2}}$ outputs Coll and aborts. For each H_2 query (by either $\mathcal{A}$ or $\mathcal{R}_{\text{DL-2}}$ simulating C or S), the probability that this query triggers Coll is $\frac{1}{p}$; since $\mathcal{A}$ makes q_{H_2} queries to H_2 (so there are up to $q_{H_2} + 2$ H_2 queries for this sub-session in total), we have that $\Pr[\mathsf{Coll}] \leq \frac{q_{H_2}+2}{p}$.
- $\gamma \neq 0$ and $\gamma + H_2(A_1^*, A_2^*)\gamma' \neq 0$: In this case, if the message $(A_1^* = g_1^{\alpha} g_2^{\beta} V^{\gamma}, A_2^* = g_1^{\alpha'} g_2^{\beta'} V^{\gamma'}, s_1^*, s_2^*)$ triggers $\mathsf{ClientImpersonation}$, then (let $d = H_2(A_1^*, A_2^*)$)

$$\begin{aligned} g_1^{s_1^*} g_2^{s_2^*} &= A_1^* (A_2^*)^d \\ &= g_1^{\alpha + d\alpha'} g_2^{\beta + d\beta'} V^{\gamma + d\gamma'}, \end{aligned}$$

  plugging in $g_2 = g_1^z$ and $V = g_1^{\log_X Y}$, we get

$$\begin{aligned} g_1^{s_1^* + z s_2^*} &= g_1^{\alpha + d\alpha' + z\beta + dz\beta'} V^{\gamma + d\gamma'} \\ &= g_1^{\alpha + d\alpha' + z\beta + dz\beta'} (g_1^{\log_X Y})^{\gamma + d\gamma'}. \end{aligned}$$

  Taking $\log_{g_1}$ on both sides yields a linear equation in $\log_X Y$:

$$(\gamma + d\gamma') \log_X Y + (\alpha + d\alpha' + z\beta + dz\beta' - s_1^* - z s_2^*) = 0$$

  (which is nondegenerate since we assumed $\gamma + d\gamma' \neq 0$), which $\mathcal{R}_{\text{DL-2}}$ can solve and check if the solution is indeed $\log_X Y$. If so, $\mathcal{R}_{\text{DL-2}}$ outputs the solution and halts. If this not, then this message does not trigger $\mathsf{ClientImpersonation}$, so $\mathcal{R}_{\text{DL-2}}$ sets S's session key $SK \leftarrow \{0,1\}^{\lambda}$, matching the behavior of Hybrid 8 (and continues simulating other sub-sessions).

In sum, if $\mathsf{ClientImpersonation}$ happens and $\mathcal{R}_{\text{DL-2}}$'s guess is correct (i.e., $\mathsf{ClientImpersonation}$ happens in a sub-session within the k-th session), then $\mathcal{R}_{\text{DL-2}}$ solves the DL instance (X, Y) correctly unless Coll happens. Therefore,

$$\mathbf{Dist}_{\mathcal{Z}}^{8,9} \leq \Pr[\mathsf{ClientImpersonation}] \leq q_S(\mathbf{Adv}^{\text{DL}}_{\mathcal{R}_{\text{DL-2}}} + \Pr[\mathsf{Coll}]) \leq q_S \left(\mathbf{Adv}^{\text{DL}}_{\mathcal{R}_{\text{DL-2}}} + \frac{q_{H_2} + 2}{p} \right).$$

Hybrid 10 (*forget DL trapdoor*). When generating public parameters, instead of storing z and computing $g_2 = g_1^z$, sample $g_2 \leftarrow \bar{\mathbb{G}}$.

Nothing in the behavior of Hybrid 9 depends on z apart from g_2, so since the distribution of g_2 is identical between Hybrids 9 and 10,

$$\mathbf{Dist}_{\mathcal{Z}}^{9,10} = 0.$$

Hybrid 11 (*rule out $\mathcal{A}$ predicting hash value without querying or compromising* S). When S receives a valid message $A_1^* = g_1^{\alpha} g_2^{\beta}$ with $\alpha = \pi\beta$, if $\mathcal{A}$ has neither queried $H_1(\mathsf{pw})$ (for this *sid*) nor stolen a server file generated with pw and *sid*, S outputs $SK \leftarrow \{0,1\}^{\lambda}$. Similarly, when C receives a message $B^* = g_1^{\alpha} g_2^{\beta}$ with $\alpha = \pi'\beta$, if $\mathcal{A}$ has neither queried $H_1(\mathsf{pw}')$ (for this *sid*) nor stolen a server file generated with pw' and *sid*, C outputs $SK' \leftarrow \{0,1\}^{\lambda}$.

Suppose the event in the description of Hybrid 11 happens. For any message from $\mathcal{A}$ to S with $A_1^* = g_1^\alpha g_2^\beta \neq e$, there is only one value of $\pi = H_1(\mathsf{pw})$ satisfying $\alpha = \pi\beta$. However, note that the behavior of S is independent of $H_1(\mathsf{pw})$ unless and until $\mathcal{A}$ queries $H_1(\mathsf{pw})$ or steals the server file generated by pw, so the probability that this message causes a difference between Hybrids 10 and 11 is at most $\frac{1}{p}$. $\mathcal{A}$'s message to C can be argued about similarly. In total, there are up to $2q_{SS}$ protocol messages sent by $\mathcal{A}$, so

$$\mathbf{Dist}_{\mathcal{Z}}^{10,11} \leq \frac{2q_{SS}}{p}.$$

Comparison between Hybrid 11 and the ideal world. We now claim that Hybrid 11 and the ideal world are identical in $\mathcal{Z}$'s view, where the challenger in Hybrid 11 corresponds to the combination of the simulator SIM and the functionality $\mathcal{F}_{\text{aPAKE}}$ in the ideal world. We compare $\mathcal{Z}$'s views in Hybrid 11 and in the ideal world message by message:

C-*to*-S *message:* In Hybrid 4, C samples $x \leftarrow \mathbb{Z}_p^*$ and computes $A_1 \coloneqq g_1^x$. The computation of A_2 never changes in the hybrids, i.e., C samples $r_1, r_2 \leftarrow \mathbb{Z}_p$ and computes $A_2 \coloneqq g_1^{r_1} g_2^{r_2}$. Then in Hybrid 5, C computes $s_1 \coloneqq x + dr_1$ and $s_2 \coloneqq dr_2$ (where $d = H_2(A_1, A_2)$). The definitions of A_1, A_2, s_1, s_2 match step 3 of SIM.

S-*to*-C *message:* In Hybrid 4, S samples $y \leftarrow \mathbb{Z}_p^*$ and computes $B \coloneqq g_1^y$. The definition of B matches step 4 of SIM.

C*'s output:* On receiving protocol message B^*,

- If $B^* \notin \bar{\mathbb{G}}$, i.e., the message C receives is invalid: C outputs $\perp$ and halts in the real world, which never changes in the hybrids. This behavior matches step 5(a) of SIM.
- If $B^* = B$, i.e., $\mathcal{A}$ forwards the S-to-C message without modification: Since B is computed as g_1^y, the representation of B^* under base g_1, g_2 (in the sense of Hybrid 6) is $g_1^\alpha g_2^\beta$ where $\alpha = y, \beta = 0$. Since $\alpha \neq H_1(\mathsf{pw}')\beta$ (note that y is sampled from $\mathbb{Z}_p^*$), in Hybrid 8 C outputs $SK' \leftarrow \{0,1\}^\lambda$. This matches step 5(b) of SIM.
- If neither of the previous cases happens, let $g_1^\alpha g_2^\beta$ be the "canonical" representation of B^*. Note that since $B^* \in \bar{\mathbb{G}}$, α and β are not both equal to 0, so $\alpha = \pi^*\beta$ has at most one solution; call this value π^* if it exists.
  - If π^* does not exist, or if it exists but $\pi^* \neq \pi'$, then the check in Hybrid 8 does not pass, so C outputs $SK' \leftarrow \{0,1\}^\lambda$. This matches step 5(c)(i), "wrong guess" case, step 5(c)(ii), "wrong guess" case, and step 5(c)(iii) of SIM.
  - If $\pi^* = \pi'$ and $\mathcal{A}$ has queried $H_1(\mathsf{pw}')$, then in Hybrid 7 C computes $K' \coloneqq A_1^\beta$ and $SK' \coloneqq H_3(A_1, B^*, K')$. This matches step 5(c)(i), "correct guess" case, of SIM.

- If $\pi^* = \pi'$ and $\mathcal{A}$ has not queried $H_1(\mathsf{pw})$ but has stolen a server file for *sid* instantiated with $\mathsf{pw} = \mathsf{pw}'$, then in Hybrid 7 C computes $K' \coloneqq A_1^{\beta}$ and $SK' \coloneqq H_3(A_1, B^*, K')$. This matches step 5(c)(ii), "correct guess" case, of SIM.
- Otherwise, i.e., if $\pi^* = \pi'$ and $\mathcal{A}$ has not queried $H_1(\mathsf{pw})$ or stolen a server file for *sid* with a matching pw, then in Hybrid 11 C outputs $SK' \leftarrow \{0,1\}^{\lambda}$. This matches step 5(c)(iii) of SIM.

We conclude that in all cases, C's behavior on receiving B^* is identical to the ideal world.

S*'s output:* On receiving protocol message $(A_1^*, A_2^*, s_1^*, s_2^*)$,

– If the message S receives is invalid: S outputs $\perp$ and halts in the real world, which never changes in the hybrids. This behavior matches step 6(a) of SIM.
– If $A_1^* = A_1$, i.e., $\mathcal{A}$ forwards the C-to-S message A_1 without modification: Since A_1 is computed as g_1^x, the representation of A_1^* under base g_1, g_2 is $g_1^{\alpha} g_2^{\beta}$ where $\alpha = x, \beta = 0$. Since $\alpha \neq H_1(\mathsf{pw})\beta$ (note that x is sampled from $\mathbb{Z}_p^*$), in Hybrid 8 S outputs $SK \leftarrow \{0,1\}^{\lambda}$. This matches step 6(b) of SIM.
– If neither of the previous cases happens, let $g_1^{\alpha} g_2^{\beta}$ be the "canonical" representation of A_1^*. Since the message is valid, $A_1^* \in \bar{\mathbb{G}}$, so α and β are not both equal to 0, so $\alpha = \pi^* \beta$ has at most one solution; call this value π^* if it exists.
  - If π^* does not exist, or if it exists but $\pi^* \neq \pi$, then the check in Hybrid 8 does not pass, so S outputs $SK \leftarrow \{0,1\}^{\lambda}$. This matches step 6(c)(i), "wrong guess" case and step 6(c)(iii) of SIM.
  - If $\pi^* = \pi$ and $\mathcal{A}$ has queried $H_1(\mathsf{pw})$, then in Hybrid 7 S computes $K \coloneqq B^{\beta}$ and $SK \coloneqq H_3(A_1^*, B, K)$. This matches step 6(c)(i), "correct guess" case of SIM.
  - If $\pi^* = \pi$, $\mathcal{A}$ has stolen the server file but has never queried $H_1(\mathsf{pw})$, then the game challenger in Hybrid 9 outputs $\mathsf{ClientImpersonation}$ and halts. This matches step 6(c)(ii) of SIM.
  - If $\pi^* = \pi$ but $\mathcal{A}$ has neither queried $H_1(\mathsf{pw})$ nor stolen the server file, then in Hybrid 11 S outputs $SK \leftarrow \{0,1\}^{\lambda}$. This matches step 6(c)(iii) of SIM.

We conclude that in all cases, S's behavior on receiving $(A_1^*, A_2^*, s_1^*, s_2^*)$ is identical to the ideal world.

S*'s password file:* In both Hybrid 11 and the ideal world, when $\mathcal{A}$ sends ($\mathsf{StealPwdfile}$, *sid*) to S (assuming a ($\mathsf{StorePwdfile}$, *sid*, pw) command has been sent), $\mathcal{A}$ receives $V = g_1^{\pi} g_2$ where $\pi = H_1(\mathsf{pw})$. The only difference lies in when π is defined: in Hybrid 11 π is defined when $\mathcal{A}$ queries $H_1(\mathsf{pw})$, whereas in the ideal world, if the ($\mathsf{StealPwdfile}$, *sid*) message comes before the $H_1(\mathsf{pw})$ query, then SIM first samples π uniformly at random and later programs $H_1(\mathsf{pw})$ to be π. This makes no external difference in $\mathcal{Z}$'s view.

Overall, we conclude that Hybrid 11 is identical to the ideal world, and thus the real world (Hybrid 0) and the ideal world are indistinguishable. This completes the proof.

Remark 1. Our proof has the feature that no reductions lose a factor of the number of sub-sessions (the number of login attempts by the client); this is because both reductions in Hybrids 8 and 9 can check if the (respective) "bad event" happens within a certain sub-session or not. We only lose a factor of the number of sessions (the number of client/server relationships) in Hybrid 9.

4 Security Analysis of Our Non-AGM-Based Protocol

4.1 UC Description of Our Protocol

A graphic illustration of our protocol is shown in Figure 6, with a formal UC description in Figure 7.

UC conventions and public parameters. Similar to the protocol in Section 3, we omit sid in H_1 and H_3's inputs and $(sid, ssid)$ in H_2 and H_4's inputs when they are clear from the context. Also similar to the protocol in Section 3, we require a common random string for generation of g_1, g_2, and g_3.

Correctness. Suppose $\mathsf{pw}' = \mathsf{pw}$ (i.e., the two parties' passwords match) and there is no adversary. Then

- S's check passes: This is because

$$g_1^{s_1} g_2^{s_2} = g_1^{\pi' a + dr_1} g_2^{a+dr_2} = (g_1^{\pi'} g_2)^a \cdot (g_1^{r_1} g_2^{r_2})^d = A_1 A_2^d,$$
$$g_1^{s_2+s_3} = g_1^{(a+dr_2)+(k+dr_3)} = g_1^{a+k} \cdot (g_1^{r_2+r_3})^d = A_3 A_4^d,$$
$$g_3^{s_3} = g_3^{k+dr_3} = g_3^k \cdot (g_3^{r_3})^d = A_5 A_6^d,$$

  where $d = H_2(A_1, A_2, A_3, A_4, A_5, A_6)$.
- The two parties' session keys are equal: Since $V = g_1^{H_1(\mathsf{pw})} g_2$, $A_1 = (g_1^{H_1(\mathsf{pw}')} g_2)^a$, and $\mathsf{pw}' = \mathsf{pw}$, we have $A_1 = V^a$. Then

$$K' = \mathcal{D}(\tau, B)^a = (V^b)^a = (V^a)^b = A_1^b = K,$$

  so $SK' = SK$.

Instantiation of the ideal cipher. Our protocol requires an Ideal Cipher (IC) over a group. As mentioned in Table 1, an IC can be instantiated from ROs (using an 8-round Feistel network). Specific, more efficient constructions exist for elliptic curve groups; we refer to [29, Section 6] for more details.

Public Parameters $= (\mathbb{G}, p, g_1, g_2, g_3, H_1 : \{0,1\}^* \to \mathbb{Z}_p, H_2 : \{0,1\}^* \to \mathbb{Z}_p,$
$H_3 : \{0,1\}^* \to \{0,1\}^\lambda, H_4 : \{0,1\}^* \to \{0,1\}^\lambda, \mathcal{E} : \{0,1\}^\lambda \times \bar{\mathbb{G}} \to \{0,1\}^\lambda,$
$\mathcal{D} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \bar{\mathbb{G}})$

Client(pw')		Server($\mathcal{C}, V \coloneqq g_1^{H_1(\mathsf{pw})} g_2, \tau \coloneqq H_3(\mathsf{pw})$)
$\pi' \coloneqq H_1(\mathsf{pw}'), \tau \coloneqq H_3(\mathsf{pw}')$		
$a \leftarrow \mathbb{Z}_p^*, k, r_1, r_2, r_3 \leftarrow \mathbb{Z}_p$		
$A_1 \coloneqq g_1^{\pi' a} g_2^a, A_2 \coloneqq g_1^{r_1} g_2^{r_2}$		
$A_3 \coloneqq g_1^{a+k}, A_4 \coloneqq g_1^{r_2+r_3}$		
		$b \leftarrow \mathbb{Z}_p^*$
$A_5 \coloneqq g_3^k, A_6 \coloneqq g_3^{r_3}$		$B \coloneqq \mathcal{E}(\tau, V^b)$
$d \coloneqq H_2(A_1, A_2, A_3, A_4, A_5, A_6)$		
$s_1 \coloneqq \pi a + d r_1$		
$s_2 \coloneqq a + d r_2$		
$s_3 \coloneqq k + d r_3$		
	$\xleftarrow{B}$	
	$\xrightarrow{A_1, A_2, A_3, A_4, A_5, A_6, s_1, s_2, s_3}$	
		$d \coloneqq H_2(A_1, A_2, A_3, A_4, A_5, A_6)$
		If any of
		$g_1^{s_1} g_2^{s_2} \neq A_1 A_2^d$
$K' \coloneqq \mathcal{D}(\tau, B)^a \quad (= (g_1^\pi g_2)^{ab})$		$g_1^{s_2+s_3} \neq A_3 A_4^d$
$SK' \coloneqq H_4(A_1, B, K')$		$g_3^{s_3} \neq A_5 A_6^d$
		then abort
		$K \coloneqq A_1^b \quad (= (g_1^{\pi'} g_2)^{ab})$
		$SK \coloneqq H_4(A_1, B, K)$

Fig. 6. Our non-AGM-based aPAKE protocol

Public parameters

Cyclic group $\mathbb{G}$ with order p, generated by algorithm $\mathsf{GenGroup}(1^\lambda)$; random group generators $g_1, g_2, g_3 \leftarrow \bar{\mathbb{G}}$; hash functions (modeled as ROs) H_1, H_2, H_3, H_4 whose ranges are $\mathbb{Z}_p, \mathbb{Z}_p, \{0,1\}^\lambda, \{0,1\}^\lambda$, respectively; ideal cipher $(\mathcal{E}, \mathcal{D})$ with message space $\bar{\mathbb{G}}$ and key and ciphertext space $\{0,1\}^\lambda$.

Password Registration

On input $(\mathsf{StorePwdfile}, sid, \mathsf{C}, \mathsf{pw})$, S computes $V := g_1^{H_1(\mathsf{pw})} g_2$ and $\tau := H_3(\mathsf{pw})$ and stores $\mathsf{file}[sid] := (\mathsf{C}, V, \tau)$.

Login

1. On input $(\mathsf{ClientSession}, sid, ssid, \mathsf{S}, \mathsf{pw}')$, C samples $a \leftarrow \mathbb{Z}_p^*$ and $k, r_1, r_2, r_3 \leftarrow \mathbb{Z}_p$, computes $\pi' := H_1(\mathsf{pw}')$, $A_1 := (g_1^{\pi'} g_2)^a$, $A_2 := g_1^{r_1} g_2^{r_2}$, $A_3 := g_1^{a+k}$, $A_4 := g_1^{r_2+r_3}$, $A_5 := g_3^k$, $A_6 := g_3^{r_3}$, $d := H_2(A_1, A_2, A_3, A_4, A_5, A_6)$, $s_1 := \pi' a + d r_1$, $s_2 := a + d r_2$, $s_3 := k + d r_3$, and sends $(sid, ssid, A_1, A_2, A_3, A_4, A_5, A_6, s_1, s_2, s_3)$ to S.
   On receiving $(sid, ssid, B)$ from S, C computes $K' := \mathcal{D}(H_3(\mathsf{pw}'), B)^a$ and $SK' := H_4(A_1, B, K')$, and outputs $(sid, ssid, SK')$.
2. On input $(\mathsf{ServerSession}, sid, ssid)$, if there is a record $\mathsf{file}[sid] = (\mathsf{C}, V, \tau)$, S samples $b \leftarrow \mathbb{Z}_p^*$, computes $B := \mathcal{E}(\tau, V^b)$, and sends $(sid, ssid, B)$ to C.
   On receiving $(sid, ssid, A_1, A_2, A_3, A_4, A_5, A_6, s_1, s_2, s_3)$ from C, S computes $d := H_2(A_1, A_2, A_3, A_4, A_5, A_6)$, then, if $A_1 \notin \bar{\mathbb{G}}$, $g_1^{s_1} g_2^{s_2} \neq A_1 A_2^d$, $g_1^{s_2+s_3} \neq A_3 A_4^d$, or $g_3^{s_3} \neq A_5 A_6^d$, S outputs $(sid, ssid, \perp)$. Otherwise, S computes $K := A_1^b$ and $SK := H_4(A_1, B, K)$, and outputs $(sid, ssid, SK)$.

Fig. 7. UC description of our non-AGM-based protocol

4.2 Security Proof

Theorem 3. *The protocol in Figure 7 UC-realizes functionality $\mathcal{F}_{aPAKE}$ in the ROM, assuming that the DDH assumption holds in $(\mathbb{G}, p)$.*

Two of the hybrids in the proof of Theorem 3 (Hybrids 6 and 10 below) rely on a technical lemma, which we present below.

Lemma 1. *Suppose $g_1, g_2, g_3, A_1 \in \bar{\mathbb{G}}$, $A_2, A_3, A_4, A_5, A_6 \in \mathbb{G}$, and $d, d', s_1, s_1', s_2, s_2', s_3, s_3' \in \mathbb{Z}_p$ with $d \neq d'$ satisfy*

$$\begin{aligned} g_1^{s_1} g_2^{s_2} &= A_1 A_2^d, \qquad (2)\\ g_1^{s_2+s_3} &= A_3 A_4^d,\\ g_3^{s_3} &= A_5 A_6^d,\\ g_1^{s_1'} g_2^{s_2'} &= A_1 A_2^{d'}, \qquad (3)\\ g_1^{s_2'+s_3'} &= A_3 A_4^{d'},\\ g_3^{s_3'} &= A_5 A_6^{d'}. \end{aligned}$$

Then for any $\pi \in \mathbb{Z}_p^*$,

$$A_1 = \left(A_3 A_5^{-\log_{g_3} g_1}\right)^{\pi + \log_{g_1} g_2} \qquad (4)$$

if and only if

$$s_1' d - s_1 d' = \pi(s_2' d - s_2 d'). \qquad (5)$$

The proof follows from straightforward algebraic manipulation and can be found in Appendix A.

Proof (of Theorem 3). Since the DDH assumption implies the CDH assumption, which in turn implies the 1ODH assumption (see Section 2.2), in the proof below we may reduce to CDH or 1ODH (although the theorem statement only mentions DDH).

We construct the simulator SIM in Figures 8 and 9. We reuse the conventions in the first paragraph of the proof of Theorem 2.

Suppose there are q_{SS} sub-sessions (i.e., q_{SS} unique *ssid*'s) in total, and $\mathcal{A}$ makes q_{H_1}, q_{H_2} and q_E queries to H_1, H_2 and $\mathcal{E}$, respectively.

Hybrids.

Hybrid 0 (*real world*). This is the real execution of the protocol. In each sub-session, the flow of events is:

- C sends $(A_1, A_2, A_3, A_4, A_5, A_6, s_1, s_2, s_3)$ to S, and S sends B to C; both messages are intercepted by $\mathcal{A}$.
- $\mathcal{A}$ sends $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ to S, causing S to compute K and output SK.

CRS generation, RO queries, and IC queries

0. SIM runs $(\mathbb{G}, p) \leftarrow \mathsf{GenGroup}(1^\lambda)$; samples $g_1 \leftarrow \bar{\mathbb{G}}$; samples (and stores) $z_2, z_3 \leftarrow \mathbb{Z}_p^*$ and computes $g_2 := g_1^{z_2}, g_3 := g_1^{z_3}$; and outputs $\mathbb{G}, p, g_1, g_2, g_3$ as the public parameters.
   SIM answers H_2 and H_4 queries via lazy sampling. H_1 and H_3 queries are described in the "Offline attacks" section. $(\mathcal{E}, \mathcal{D})$ queries are answered via lazy sampling, except that when a fresh decryption query is received, rather than sampling the result from $\bar{\mathbb{G}}$ directly, SIM samples $t \leftarrow \mathbb{Z}_p^*$, stores $\langle \mathsf{Trapdoor}, g_1^t, t \rangle$, and programs g_1^t as the result.
   If any of the following happens: (1) two H_1 queries with the same sid give the same result, (2) the result of an H_1 query matches the value of π sampled for the stolen password file for the same sid, (3) the result of an H_1 query or sampled π value is $-z_2$, (4) two H_3 queries for the same $ssid$ give the same result, (5) the result of an H_3 query matches the value of τ sampled for the stolen password file for the same sid, (6) two $\mathcal{E}$ queries with the same $ssid$ give the same result, or (7) the result of an $\mathcal{E}$ query equals a message B^* sent to C with the same $ssid$, SIM outputs $\mathsf{Collision}$ and halts.

Offline attacks

1. On receiving $(\mathsf{StealPwdfile}, sid)$ from $\mathcal{A}$ to S, SIM sends $(\mathsf{StealPwdfile}, sid)$ to $\mathcal{F}_{\text{aPAKE}}$.
   (a) If the response is "no password file", SIM returns "no password file".
   (b) If the response is pw, then there must have been an $(\mathsf{OfflineTestPwd}, sid, \mathsf{pw})$ command, which means that there has been either an $H_1(\mathsf{pw})$ query or an $H_3(\mathsf{pw})$ query.
   If there has been an $H_1(\mathsf{pw})$ query, let $\pi := H_1(\mathsf{pw})$ and SIM computes $V := g_1^\pi g_2$ and $\tau := H_3(\mathsf{pw})$; if there has been an $H_3(\mathsf{pw})$ query, let $\tau := H_3(\mathsf{pw})$ and SIM computes $\pi := H_1(\mathsf{pw})$ and $V := g_1^\pi g_2$. // pre-computation attack on either V or τ
   Either way, SIM stores $\langle \mathsf{PWRecord}, sid, \mathsf{pw}, \pi, V, \tau \rangle$, and outputs (V, τ).
   (c) Otherwise, SIM samples $\pi \leftarrow \mathbb{Z}_p$ and $\tau \leftarrow \{0,1\}^\lambda$, computes $V := g_1^\pi g_2$, stores $\langle \mathsf{PWRecord}, sid, \perp, \pi, V, \tau \rangle$, and outputs (V, τ).
2. On RO query $H_1(\mathsf{pw})$ or $H_3(\mathsf{pw})$, SIM sends $(\mathsf{OfflineTestPwd}, sid, \mathsf{pw})$ to $\mathcal{F}_{\text{aPAKE}}$.
   (a) If the response is "correct guess", then $\mathcal{Z}$ must have sent $(\mathsf{StealPwdfile}, sid)$ previously, so there must be a record $\langle \mathsf{PWRecord}, sid, \cdot, \pi, V, \tau \rangle$. SIM modifies the record to $\langle \mathsf{PWRecord}, sid, \mathsf{pw}, \pi, V, \tau \rangle$, defines $H_1(\mathsf{pw}) := \pi$ and $H_3(\mathsf{pw}) := \tau$, and outputs π (if the query is $H_1(\mathsf{pw})$) or τ (if the query is $H_3(\mathsf{pw})$).
   (b) Otherwise, SIM samples $\pi \leftarrow \mathbb{Z}_p$ and $\tau \leftarrow \{0,1\}^\lambda$, defines $H_1(\mathsf{pw}) := \pi$ and $H_3(\mathsf{pw}) := \tau$, and outputs π (if the query is $H_1(\mathsf{pw})$) or τ (if the query is $H_3(\mathsf{pw})$).

Simulation of protocol messages

3. On receiving $(\mathsf{ClientSession}, sid, ssid, \mathsf{C}, \mathsf{S})$ from $\mathcal{F}_{\text{aPAKE}}$, SIM samples $x \leftarrow \mathbb{Z}_p^*$ and $a', k, r_1, r_2, r_3 \leftarrow \mathbb{Z}_p$, computes $A_1 := g_1^x$, $A_2 := g_1^{r_1} g_2^{r_2}$,$A_3 := g_1^{a'+k}$, $A_4 := g_1^{r_2+r_3}$, $A_5 := g_3^k$, $A_6 := g_3^{r_3}$, $d := H_2(A_1, A_2, A_3, A_4, A_5, A_6)$, $s_1 := x - z_2 a' + d r_1$, $s_2 := a' + d r_2$, and $s_3 := k + d r_3$, stores $\langle \mathsf{ClientSession}, ssid, x, A_1 \rangle$, and sends $(A_1, A_2, A_3, A_4, A_5, A_6, s_1, s_2, s_3)$ from C to S.
4. On receiving $(\mathsf{ServerSession}, sid, ssid, \mathsf{S}, \mathsf{C})$ from $\mathcal{F}_{\text{aPAKE}}$, SIM samples $y \leftarrow \mathbb{Z}_p^*$, computes $B := \mathcal{E}(\perp, g_1^y)$, stores $\langle \mathsf{ServerSession}, ssid, B \rangle$, and sends B from S to C. // B is an encryption under $\perp$ so it counts for $\mathcal{E}$ collisions.

Fig. 8. Simulator SIM for our non-AGM-based protocol (part I)

Active attacks

5. On receiving B^* from S to C, once a record $\langle \mathsf{ClientSession}, ssid, x, A_1 \rangle$ exists,
   (a) If there is a record $\langle \mathsf{ServerSession}, ssid, y, B \rangle$ with $B^* = B$, SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, 0^\lambda)$ to $\mathcal{F}_{\text{aPAKE}}$. // $\mathcal{A}$ does not modify S-to-C message
   (b) If B^* is an IC encryption under some $\tau^* = H_3(\mathsf{pw}^*)$, SIM sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, \mathsf{pw}^*)$ to $\mathcal{F}_{\text{aPAKE}}$. (There must be at most one such τ^* and at most one such pw^* since SIM excluded $\mathcal{E}$ and H_3 collisions in step 0.) // pw^* is $\mathcal{A}$'s password guess
      – If the response is "wrong guess", SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, 0^\lambda)$ to $\mathcal{F}_{\text{aPAKE}}$.
      – If the response is "correct guess", SIM computes $a := \frac{x}{H_1(\mathsf{pw}^*)+z_2}$, $K' := \mathcal{D}(\tau^*, B^*)^a$ and $SK' := H_4(A_1, B^*, K')$, and sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, SK')$ to $\mathcal{F}_{\text{aPAKE}}$.
   (c) If there is a record $\langle \mathsf{PWRecord}, sid, \perp, \pi, V, \tau \rangle$, and B^* is an IC encryption (of some group element) under τ, SIM sends $(\mathsf{Impersonate}, sid, ssid)$ to $\mathcal{F}_{\text{aPAKE}}$. // $\mathcal{A}$ compromises S and executes S's algorithm on its file (V, τ)
      – If the response is "wrong guess", SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, 0^\lambda)$ to $\mathcal{F}_{\text{aPAKE}}$.
      – If the response is "correct guess", SIM computes $a := \frac{x}{\pi+z_2}$, $K' := \mathcal{D}(\tau, B^*)^a$ and $SK' := H_4(A_1, B^*, K')$, and sends $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, SK')$ to $\mathcal{F}_{\text{aPAKE}}$.
   (d) Otherwise, SIM sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{C}, \perp)$ followed by $(\mathsf{NewKey}, sid, ssid, \mathsf{C}, 0^\lambda)$ to $\mathcal{F}_{\text{aPAKE}}$.
6. On receiving $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ from C to S, once a record $\langle \mathsf{ServerSession}, ssid, B \rangle$ exists,
   (a) If $A_1^* \notin \bar{\mathbb{G}}$, or $g_1^{s_1^*} g_2^{s_2^*} \neq A_1^*(A_2^*)^d$, or $g_1^{s_2^*+s_3^*} \neq A_3^*(A_4^*)^d$, or $g_3^{s_3^*} \neq A_5^*(A_6^*)^d$ (where $d := H_2(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*)$), SIM sends $(\mathsf{Abort}, sid, ssid, \mathsf{S})$ to $\mathcal{F}_{\text{aPAKE}}$.
   (b) If there is a record $\langle \mathsf{ClientSession}, ssid, x, A_1 \rangle$ with $A_1^* = A_1$, SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, 0^\lambda)$ to $\mathcal{F}_{\text{aPAKE}}$.
   (c) Otherwise:
      i. If $A_1^* = \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi^*+z_2}$ for some $\pi^* = H_1(\mathsf{pw}^*)$, SIM sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, \mathsf{pw}^*)$ to $\mathcal{F}_{\text{aPAKE}}$.
         – If the response is "wrong guess", SIM sends $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, 0^\lambda)$ to $\mathcal{F}_{\text{aPAKE}}$.
         – If the response is "correct guess", SIM computes $T := \mathcal{D}(H_3(\mathsf{pw}^*), B)$, looks up t in the record $\langle \mathsf{Trapdoor}, T, t \rangle$, computes $b := \frac{t}{\pi^*+z_2}$, $K := (A_1^*)^b$, and $SK := H_4(A_1^*, B, K)$, and sends $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, SK)$ to $\mathcal{F}_{\text{aPAKE}}$.
      ii. If there is a record $\langle \mathsf{PWRecord}, sid, \perp, \pi, V, \tau \rangle$ and $A_1^* = \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi+z_2}$, SIM outputs $\mathsf{ClientImpersonation}$ and halts. // $\mathcal{A}$ impersonates client after compromising server, without successful offline attack
      iii. Otherwise, SIM sends $(\mathsf{TestPwd}, sid, ssid, \mathsf{S}, \perp)$ followed by $(\mathsf{NewKey}, sid, ssid, \mathsf{S}, 0^\lambda)$ to $\mathcal{F}_{\text{aPAKE}}$.

Fig. 9. Simulator SIM for our non-AGM-based protocol (part II)

– $\mathcal{A}$ sends B^* to C, causing C to compute K' and output SK'.

(The order of the last two events is decided by $\mathcal{Z}$.)

Below we call a message to S $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ "valid" if $A_1^* \in \bar{\mathbb{G}}$, $g_1^{s_1^*} g_2^{s_2^*} = A_1^*(A_2^*)^d$, $g_1^{s_2^*+s_3^*} = A_3^*(A_4^*)^d$ and $g_3^{s_3^*} = A_5^*(A_6^*)^d$, where $d = H_2(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*)$. In all subsequent hybrids, we assume that the message is valid, i.e., we do not make any changes if the message is invalid. Note that for any $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*) \in \bar{\mathbb{G}} \times \mathbb{G}^5$, there is a unique tuple $(s_1^*, s_2^*, s_3^*) \in \mathbb{Z}_p^3$ that makes the message valid; we will repeatedly use this fact in our argument.

Hybrid 1 (*both parties output same key if passwords match and $\mathcal{A}$ passive except in the validation part***).** When either party P receives a message with *ssid*, if its counterparty has already output $(sid, ssid, SK)$ with $SK \neq \perp$, $\mathsf{pw} = \mathsf{pw}'$, $A_1^* = A_1$, and $B^* = B$, then P outputs $(sid, ssid, SK)$ instead of its normal response. (In all subsequent hybrids, we never change this rule and do not repeat it in the hybrid description.)

By correctness of the protocol and the fact that the group elements A_3^* and A_5^* along with the NIZK $(A_2^*, A_4^*, A_6^*, s_1^*, s_2^*, s_3^*)$ have no affect on the derivation of SK for valid messages (note that as stated above we do not change behavior for invalid messages), we have

$$\mathbf{Dist}_{\mathcal{Z}}^{0,1} = 0.$$

Hybrid 2 (*halt on H_1 collision, or H_1 result is DL trapdoor, or H_3 or $\mathcal{E}$ collision***).** If two H_1 queries with the same *sid* give the same result, or if the result π^* of an H_1 query satisfies $g_1^{-\pi^*} = g_2$, output $\mathsf{Collision}$ and halt. If two H_3 queries with the same *sid* give the same result, output $\mathsf{Collision}$ and halt. If two $\mathcal{E}$ queries with the same *ssid* give the same result, or if the result of an $\mathcal{E}$ query is equal to B^* in this sub-session, also output $\mathsf{Collision}$ and halt. (In all subsequent hybrids, we assume that $\mathsf{Collision}$ does not happen.)

We have

$$\begin{aligned}\mathbf{Dist}_{\mathcal{Z}}^{1,2} &\leq \Pr[\mathsf{Collision}] \\ &\leq \frac{(q_{H_1}+2)(q_{H_1}+1)}{2p} + \frac{q_{H_1}+2q_S}{p} + \frac{(q_{H_3}+2)(q_{H_3}+1)}{2^\lambda} + \frac{(q_E+1)q_E}{2p} + \frac{q_E+q_{SS}}{p} \\ &= \frac{(q_{H_1}^2+5q_{H_1}+4q_S+3)+(q_E^2+3q_E+2q_{SS})}{2p} + \frac{q_{H_3}^2+3q_{H_3}+2}{2^\lambda},\end{aligned}$$

where the "+2" etc. are due to honest parties' H_1 and $\mathcal{E}$ queries.

Hybrid 3 (*store DL trapdoors***).** When generating public parameters, instead of sampling $g_2 \leftarrow \bar{\mathbb{G}}$ and $g_3 \leftarrow \bar{\mathbb{G}}$, sample $z_2, z_3 \leftarrow \mathbb{Z}_p^*$, store z_2 and z_3, and compute $g_2 \coloneqq g_1^{z_2}$ and $g_3 \coloneqq g_1^{z_3}$.

In both Hybrids 2 and 3, (g_1, g_2, g_3) is uniformly random in $\bar{\mathbb{G}}^3$, so

$$\mathbf{Dist}_{\mathcal{Z}}^{2,3} = 0.$$

Hybrid 4 (*compute A_1 and B without using π or π'*). Instead of sampling $a \leftarrow \mathbb{Z}_p^*$ and computing $A_1 \coloneqq (g_1^{\pi'} g_2)^a$, C samples $x \leftarrow \mathbb{Z}_p^*$ and computes $A_1 \coloneqq g_1^x$ and $a \coloneqq \frac{x}{\pi' + z_2}$. Similarly, instead of sampling $b \leftarrow \mathbb{Z}_p^*$ and computing $B \coloneqq \mathcal{E}(\tau, V^b)$, S samples $y \leftarrow \mathbb{Z}_p^*$ and computes $B \coloneqq \mathcal{E}(\perp, g_1^y)$ and $b \coloneqq \frac{1}{\pi + z_2} \log_{g_1} \mathcal{D}(\tau, B)$.

Both Hybrids 3 and 4 satisfy

$$x = a(\pi' + z_2)$$

(where $x = \log_{g_1} A_1$), and $\pi' \neq -z_2$ (the $\pi' = -z_2$ case was excluded in Hybrid 2). Subject to these conditions, if we sample $a \leftarrow \mathbb{Z}_p^*$ then we get Hybrid 3, and if we sample $x \leftarrow \mathbb{Z}_p^*$ then we get Hybrid 4.

Similarly, in both Hybrids 3 and 4 B is a uniformly random element of $\{0,1\}^\lambda$ (that is, a result of an $\mathcal{E}$ query),

$$\log_{g_1} \mathcal{D}(\tau, B) = b(\pi + z_2),$$

and $\pi \neq -z_2$. Subject to these conditions, if we sample $b \leftarrow \mathbb{Z}_p^*$ then we get Hybrid 3, and if we sample $B \leftarrow \{0,1\}^\lambda$ (as the result of an $\mathcal{E}$ query) then we get Hybrid 4. We have

$$\mathbf{Dist}_{\mathcal{Z}}^{3,4} = 0.$$

Hybrid 5 (*compute s_1, s_2, and A_3 without using π'*). Instead of computing $A_3 \coloneqq g_1^{a+k}$, $s_1 \coloneqq \pi' a + dr_1$, and $s_2 \coloneqq a + dr_2$, C samples $a' \leftarrow \mathbb{Z}_p$ and computes $A_3 \coloneqq g_1^{a'+k}$, $s_1 \coloneqq x - z_2 a' + dr_1$, and $s_2 \coloneqq a' + dr_2$. That is, in place of $a = \frac{x}{\pi' + z_2}$, we use a random integer a' independent of x to compute A_3, s_1, and s_2.

The difference between Hybrid 4 and Hybrid 5 is that in Hybrid 4 $\left(g_1, g_3, \frac{A_3}{g_1^a}, A_5\right) = (g_1, g_3, g_1^k, g_3^k)$ forms a DH tuple, whereas in Hybrid 5 $\left(g_1, g_3, \frac{A_3}{g_1^a}, A_5\right)$ is a uniformly random tuple in $\bar{\mathbb{G}}^2 \times \mathbb{G}^2$. This suggests a reduction $\mathcal{R}_{\mathrm{DDH}}$ to the DDH problem.

Description of $\mathcal{R}_{DDH}$. $\mathcal{R}_{\mathrm{DDH}}$, on challenge $(X, Y, Z, W) \in \bar{\mathbb{G}}^2 \times \mathbb{G}^2$, sets CRS parameters $g_1 \coloneqq X$ and $g_3 \coloneqq Y$, leaving $z_3 = \log_{g_1} g_3$ unknown. ($g_2 \coloneqq g_1^{z_2}$ is computed as normal.)

$\mathcal{R}_{\mathrm{DDH}}$ then runs the code of Hybrid 4, except that in each sub-session, while generating the C-to-S message, it samples $\alpha, \beta \leftarrow \mathbb{Z}_p^*$ and computes $N \coloneqq X^\alpha Z^\beta, M \coloneqq Y^\alpha W^\beta$. (This is necessary since $\mathcal{R}_{\mathrm{DDH}}$ needs to re-randomize its C-to-S message in each sub-session.) Then $\mathcal{R}_{\mathrm{DDH}}$ samples x, r_1, and r_2 as in Hybrid 4 (but not k or r_3), computes a, A_1, and A_2 as in Hybrid 4 and

$$A_3 \coloneqq g_1^a N, A_5 \coloneqq M,$$

then samples $d \leftarrow \mathbb{Z}_p^*$ and $s_3 \leftarrow \mathbb{Z}_p$, computes s_1 and s_2 as in Hybrid 4, and

$$A_4 \coloneqq g_1^{r_2 + \frac{s_3}{d}} N^{-\frac{1}{d}}, A_6 \coloneqq g_3^{\frac{s_3}{d}} M^{-\frac{1}{d}},$$

and finally programs $H_2(A_1, A_2, A_3, A_4, A_5, A_6) \coloneqq d$. When $\mathcal{Z}$ outputs, $\mathcal{R}_{\mathrm{DDH}}$ copies its output bit.

Analysis of $\mathcal{R}_{DDH}$. We argue that if (X, Y, Z, W) forms a DH tuple then $\mathcal{R}_{\mathrm{DDH}}$ simulates Hybrid 4, whereas if (X, Y, Z, W) is uniformly random then $\mathcal{R}_{\mathrm{DDH}}$ simulates Hybrid 5. This can be seen as follows:

1. In both Hybrids 4 and 5 and in $\mathcal{R}_{\mathrm{DDH}}$, $A_1 = g_1^x$.
2. In both Hybrids 4 and 5, (A_2, A_4, A_5, A_6) is uniformly random in $\mathbb{G}^4$, since (k, r_1, r_2, r_3) is uniformly random in $\mathbb{Z}_p^4$. In $\mathcal{R}_{\mathrm{DDH}}$, (A_2, A_4, A_5, A_6) is also uniformly random in $\mathbb{G}^4$, since (M, r_1, r_2, s_3) is uniformly random in $\mathbb{G} \times \mathbb{Z}_p^3$.
3. In both Hybrids 4 and 5, due to correctness of the protocol, (s_1, s_2, s_3) is the unique tuple that makes $(A_1, A_2, A_3, A_4, A_5, A_6, s_1, s_2, s_3)$ a valid C-to-S message. This is also the case for $\mathcal{R}_{\mathrm{DDH}}$:
   - $A_1 A_2^d = g_1^{s_1} g_2^{s_2}$, since all of the values are computed the same way as in Hybrid 4.
   - $A_3 A_4^d = (g_1^a N) \left(g_1^{r_2 + \frac{s_3}{d}} N^{-\frac{1}{d}}\right)^d = g_1^{a + d r_2 + s_3} = g_1^{s_2 + s_3}$.
   - $A_5 A_6^d = M \left(g_3^{\frac{s_3}{d}} M^{-\frac{1}{d}}\right)^d = g_3^{s_3}$.
4. This leaves the definition of A_3, which is where the difference between Hybrids 4 and 5 lies. In Hybrid 4, A_3 is the unique value that satisfies $A_1 = \left(A_3 A_5^{-\frac{1}{\log_{g_1} g_3}}\right)^{\pi + z_2}$. This matches the behavior of $\mathcal{R}_{\mathrm{DDH}}$ when (X, Y, Z, W) is a DH tuple: since (X, Y, Z, W) is a DH tuple, $(g_1, g_3, N, M) = (X, Y, X^\alpha Z^\beta, Y^\alpha W^\beta)$ is also a DH tuple, i.e., $\log_{g_1} g_3 = \log_N M$, so

   $$\left(A_3 A_5^{-\frac{1}{\log_{g_1} g_3}}\right)^{\pi + z_2} = \left[(g_1^a N) M^{-\frac{1}{\log_{g_1} g_3}}\right]^{\pi + z_2} = (g_1^a N N^{-1})^{\pi + z_2} = (g_1^{\pi + z_2})^a = (g_1^\pi g_2)^a = A_1.$$

   On the other hand, in Hybrid 5, A_3 is uniformly random in $\mathbb{G}$, independent of A_1 and A_5. This matches the behavior of $\mathcal{R}_{\mathrm{DDH}}$ when (X, Y, Z, W) is uniformly random: since N is independent of other group elements, $A_3 = g_1^a N$ is independent of A_1 and A_5 whose expressions do not involve N.

   Therefore,

$$\mathbf{Dist}_{\mathcal{Z}}^{4,5} = \mathbf{Adv}_{\mathcal{R}_{\mathrm{DDH}}}^{\mathrm{DDH}}.$$

Hybrid 6 (S *outputs random session key on incorrect password guess*). When S receives a valid message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ with $A_1^* \neq \left(A_3^* (A_5^*)^{-\frac{1}{z_3}}\right)^{\pi + z_2}$ from $\mathcal{A}$, if it is not equal to any message sent by a C instance, S outputs $SK \leftarrow \{0,1\}^\lambda$ instead of computing SK normally.

Let WrongGuessS be the event that $\mathcal{A}$ sends a valid message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ with $A_1^* \neq \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi+z_2}$ which is not forwarded from a C instance to S and queries $H_4(A_1^*, B, K)$, where B and K are the same values that S would use in Hybrid 5. Hybrids 5 and 6 are identical unless WrongGuessS happens. We now show that Pr[WrongGuess] is negligible.

Overview of indistinguishability argument. The indistinguishability argument in Hybrid 6 is the most involved one, so we provide a high-level overview before getting into the details.

We first explain the hybrid itself. Intuitively, when $\mathcal{A}$ sends a message to S with A_1^*, the only way it can compute S's value of K (and therefore SK) is if it knows $\log_{g_1^\pi g_2} A_1^*$, which is equivalent to knowing the unique pair $(\alpha, \beta) \in \mathbb{Z}_p^2$ satisfying $\alpha = \pi\beta$ and $A_1^* = g_1^\alpha g_2^\beta$. The NIZK, which consists of A_2^*, A_4^*, A_6^*, s_1^*, s_2^*, and s_3^*, is for the relation

$$\{((A_1^*, A_3^*, A_5^*), (\alpha, \beta, \gamma)) : A_1^* = g_1^\alpha g_2^\beta \wedge A_3^* = g_1^{\beta+\gamma} \wedge A_5^* = g_3^\gamma\}$$

which has exactly one possible witness for any value of (A_1^*, A_3^*, A_5^*). $\mathcal{A}$ likely only knows only one representation of A_1^* in terms of g_1 and g_2, since otherwise $\mathcal{A}$ can compute $\log_{g_1} g_2$, and since the NIZK requires knowledge of such a representation, the representation $\mathcal{A}$ uses for the NIZK must satisfy $\alpha = \pi\beta$ in order for it to compute K. While the NIZK itself reveals nothing about this representation (since it is zero-knowledge), A_3^* and A_5^* do — in particular, if $\alpha = \pi\beta$ then

$$A_1^* = \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi+z_2},$$

so Hybrid 6 uses this condition to check if $\mathcal{A}$'s password guess is correct (and lets S output a random session key if not).

Next, we discuss the reduction strategy. Suppose the "bad event" WrongGuessS happens. While the intuition above suggests a reduction to DL, by combining the $\alpha = \pi\beta$ representation needed to compute K with the $\alpha \neq \pi\beta$ representation used in the NIZK to compute $\log_{g_1} g_2$, the former is inaccessible and in fact need not actually exist for non-algebraic $\mathcal{A}$. Instead, the value of K is involved in a DH tuple with the group element $U \coloneqq \mathcal{D}(\tau, B)$ which is the decryption of the message sent by S:

Let $U = V^b = (g_1^\pi g_2)^b = g_1^{(\pi+z_2)b}$. On message A_1^*, S computes

$$K = A_1^b = (g_1^\alpha g_2^\beta)^b = g_1^{(\alpha+\beta z_2)b},$$

so $(KU^{-\beta})^{\frac{1}{\alpha-\pi\beta}} = g_1^b$ and thus $\left(V, g_1, U, (KU^{-\beta})^{\frac{1}{\alpha-\pi\beta}}\right)$ forms a DH tuple. This suggests a reduction to CDH (although ultimately 1ODH is used instead for simplicity).

In order to compute the last element of the DH tuple given the value of K, the reduction must learn α and β. By the special soundness property of the

NIZK, this can be computed from any two NIZKs for the same $A_1^*, \ldots, A_6^*$ with different values of d, which can be obtained by rewinding $\mathcal{A}$.

One further complication is that $\mathcal{A}$ does not output a guess for K, but makes a number of guesses for K implicitly through querying H_4. This is the reason for reducing to 1ODH rather than CDH, since 1ODH oracle queries can be used to identify the correct value of K.

Concretely, we must define a series of algorithms that fit the syntax of the forking lemma and allow the reduction to go through:

- We first define a *wrapper algorithm* $\mathcal{W}$, which runs the hybrid for $\mathcal{A}$ and identifies (1) whether $\mathcal{A}$ causes $\mathsf{WrongGuessS}$ to happen, and (2) if so, the corresponding values (s_1^*, s_2^*, d) (together with which of $\mathcal{A}$'s H_2 queries results in d).
- By the forking lemma, the wrapper algorithm immediately yields the *forking algorithm* $\mathcal{F}^{\mathcal{W}}$, which runs $\mathcal{A}$ twice (with the same random coins) and outputs two tuples of (s_1^*, s_2^*, d).
- Finally, the *reduction* $\mathcal{R}_{\text{1ODH-1}}$ runs $\mathcal{F}^{\mathcal{W}}$ and computes α, β from $\mathcal{F}^{\mathcal{W}}$'s outputs.

Additional care must be taken while constructing these algorithms. Since $\mathcal{R}_{\text{1ODH-1}}$ needs to embed its 1ODH challenge as $(V = g_1^{\pi} g_2, g_1, U)$, and π is accessible to $\mathcal{A}$, $\mathcal{R}_{\text{1ODH-1}}$ cannot know $z_2 = \log_{g_1} g_2$, and therefore neither can $\mathcal{W}$. In order to detect whether $\mathsf{WrongGuessS}$ happens, however, $\mathcal{W}$ needs to check $A_1^* \neq \left(A_3^* (A_5^*)^{-\frac{1}{z_3}}\right)^{\pi + z_2}$, which it cannot do.

Normally, this could be solved simply by having a reduction guess the session and sub-session in which $\mathsf{WrongGuessS}$ happens. This works because the reduction merely needs to have non-negligible probability of success so long as $\mathcal{A}$ does, but it is allowed to have "false positives". However, this poses an issue for analysis when combined with the forking lemma: if $\mathcal{W}$ has a non-negligible "false positive" rate, then it is difficult to guarantee that $\mathcal{F}^{\mathcal{W}}$'s positive outputs are not false positives with non-negligible probability.

To circumvent this issue in our analysis, we first describe $\mathcal{W}$ with an extra oracle $\mathcal{O}'$ which it can use to verify that its session and sub-session guess is correct, and later replace this oracle with a constant function that always returns 0. This requires the introduction of an intermediate reduction $\mathcal{R}$, which can be analyzed assuming that all positive outputs from $\mathcal{F}^{\mathcal{W}}$ are true positives, simplifying the analysis. Since $\mathcal{W}$'s guesses must be correct for $\mathcal{R}$ to succeed, replacing $\mathcal{O}'$ with the constant function 0 does not affect any successful execution.

Description of the wrapper algorithm $\mathcal{W}$. Let $q = q_{H_2} + 1$ (where the "+1" is due to honest parties' H_2 query). For a fixed adversary $\mathcal{A}$, we construct the wrapper algorithm $\mathcal{W}$ as follows. $\mathcal{W}$ takes as input $(X, Y, Z) \in \bar{\mathbb{G}}^3$ and $(h_1, \ldots, h_q) \in \mathbb{Z}_p^q$, and is equipped with the following three oracles:

- $H : \mathbb{G} \to \{0,1\}^{\lambda}$, a random oracle.
- $\mathcal{O} : \mathbb{G} \to \{0,1\}^{\lambda}$, which on input N returns $H(N^{\log_X Z})$.

– $\mathcal{O}' : \mathbb{G}^2 \to \{0,1\}$, which on input (N, M) returns 1 if $N^{\log_X Y} = M$ and 0 otherwise.

$\mathcal{W}$, on input (X, Y, Z) and $(h_1, \ldots, h_q)$, samples $k \leftarrow [1, q_{SS}]$ (as a guess that $\mathsf{WrongGuessS}$ happens first in the k-th sub-session) and $\pi \leftarrow \mathbb{Z}_p$. When generating CRS parameters, $\mathcal{W}$ sets $g_1 \coloneqq Y$ and $g_2 \coloneqq X g_1^{-\pi}$ (with g_3 computed as normal). When S's instance begins, $\mathcal{W}$ programs $H_1(\mathsf{pw}) \coloneqq \pi$. After that, $\mathcal{W}$ runs the code of Hybrid 5, except that in the k-th sub-session, $\mathcal{W}$ does the following:

On the i-th unique query to H_2 (by $\mathcal{A}$ or by $\mathcal{W}$ itself), $\mathcal{W}$ returns h_i. Note that $\mathcal{W}$ only uses h_i to answer $H_2(ssid, \ldots)$ queries where $ssid$ is the ID for the k-th sub-session; for H_2 queries for other sub-sessions, $\mathcal{W}$ answers with lazy sampling.

While generating the C-to-S message, $\mathcal{W}$ samples $a \leftarrow \mathbb{Z}_p^*$, computes $A_1 \coloneqq (g_1^{\pi'} g_2)^a$, samples $A_3, A_5 \leftarrow \mathbb{G}$, $s_1, s_2, s_3 \leftarrow \mathbb{Z}_p$, and $d \leftarrow \mathbb{Z}_p^*$, computes

$$A_2 \coloneqq g_1^{\frac{s_1}{d}} g_2^{\frac{s_2}{d}} A_1^{-\frac{1}{d}}, A_4 \coloneqq g_1^{\frac{s_2+s_3}{d}} A_3^{-\frac{1}{d}}, A_6 \coloneqq g_3^{\frac{s_3}{d}} A_5^{-\frac{1}{d}},$$

and programs $H_2(A_1, A_2, A_3, A_4, A_5, A_6) \coloneqq d$.[9]

In the S instance, $\mathcal{W}$ programs $\mathcal{D}(\tau, B) \coloneqq Z$.

When S receives a message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$, if the message is invalid or is equal to a message sent by a C instance (in the current sub-session or another sub-session), $\mathcal{W}$ outputs 0 and halts. If $\mathcal{O}'\left(A_1^*, A_3^*(A_5^*)^{-\frac{1}{z_3}}\right) = 1$, $\mathcal{W}$ also outputs 0 and halts. Otherwise, let I be the (unique) index for which the input to the I-th H_2 query is $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*)$ (showing that such a query exists is left for the analysis). Then $\mathcal{W}$ queries $h \coloneqq \mathcal{O}(A_1^*)$ and lets S output $SK \leftarrow \{0,1\}^\lambda$.

On an $H_4(A_1^*, B, K^*)$ query, $\mathcal{W}$ answers with $H(K^*)$ (using an H query). If $H(K^*) = h$ (including retroactively for H_4 queries made before h is determined), $\mathcal{W}$ knows that $\mathsf{WrongGuessS}$ happens, so $\mathcal{W}$ collects and outputs the relevant values $(I, (s_1^*, s_2^*, d, \pi, K^*))$. (If $\mathcal{A}$ halts before this occurs, $\mathcal{W}$ outputs 0 and halts.)

Description of the intermediate reduction $\mathcal{R}$. Next, we define the "intermediate reduction" $\mathcal{R}$, which also takes $(X, Y, Z) \in \bar{\mathbb{G}}^3$ as input and uses the same collection of oracles $H, \mathcal{O}, \mathcal{O}'$ as $\mathcal{W}$.

$\mathcal{R}$, on input (X, Y, Z), runs the forking algorithm $\mathcal{F}^{\mathcal{W}}$ on (X, Y, Z). For the oracle queries made by $\mathcal{F}^{\mathcal{W}}$, $\mathcal{R}$ answers the H and $\mathcal{O}'$ queries by querying its own oracle. For $\mathcal{O}$, $\mathcal{R}$ answers the first query by querying its own oracle, but if

[9] $\mathcal{W}$, as described, needs to abort if $H_2(A_1, A_2, A_3, A_4, A_5, A_6)$ has already been queried, which incurs a negligible failure probability. To avoid this, $\mathcal{W}$ can precompute the C-to-S message at the beginning of its execution (after sampling k and before invoking $\mathcal{Z}$). The same goes for the reductions in Hybrids 7, 8, 9, and 10, which simulate the C-to-S message in the same manner.

a subsequent query is made on a different input, $\mathcal{R}$ outputs TwoOQueries and halts.

When $\mathcal{F}^{\mathcal{W}}$ outputs $(b, \star, \star)$, if $b = 0$, $\mathcal{R}$ outputs NotAccept and halts. Otherwise $\mathcal{R}$ parses $\mathcal{F}^{\mathcal{W}}$'s output as $(1, (s_1, s_2, d, \pi, K)), (s_1', s_2', d', \pi', K'))$. In this case the definition of $\mathcal{F}^{\mathcal{W}}$ (see step 7 of Theorem 1) guarantees that $h_I \neq h'_{I'}$ in the two runs of $\mathcal{W}$; but $\mathcal{W}$ defines $d = h_I$ and $d' = h'_{I'}$, so $d \neq d'$. This allows $\mathcal{R}$ to compute $\alpha \coloneqq \frac{s_1' d - s_1 d'}{d - d'}$ and $\beta \coloneqq \frac{s_2' d - s_2 d'}{d - d'}$. If $\alpha = \pi\beta$, $\mathcal{R}$ outputs NoWrongGuessS and halts. Otherwise, $\mathcal{R}$ computes $W \coloneqq (KZ^{-\beta})^{\frac{1}{\alpha - \pi\beta}}$ and outputs W.

Description of the reduction $\mathcal{R}_{1ODH\text{-}1}$. Finally, we construct the reduction $\mathcal{R}_{1\text{ODH-1}}$. On input (X, Y, Z), and access to oracles H and $\mathcal{O}$, $\mathcal{R}_{1\text{ODH-1}}$ runs $\mathcal{R}$ on (X, Y, Z). $\mathcal{R}_{1\text{ODH-1}}$ answers $\mathcal{R}$'s H and $\mathcal{O}$ queries by querying its own oracles. $\mathcal{O}'$ queries are answered with 0 regardless of content. When $\mathcal{R}$ outputs, $\mathcal{R}_{1\text{ODH-1}}$ copies its output.

Analysis of $\mathcal{W}$. First, we analyze the behavior of $\mathcal{W}$ when its input $(X, Y, Z, h_1, \ldots h_q)$ is uniformly random in $\bar{\mathbb{G}}^3 \times \mathbb{Z}_p^q$, and all oracles are implemented as specified. Suppose $\mathcal{W}$'s guess is correct, i.e., WrongGuessS indeed happens in the k-th sub-session. To begin with, one can check that in the C-to-S message simulated by $\mathcal{W}$, $(A_1, A_2, A_3, A_4, A_5, A_6)$ is uniformly random in $\mathbb{G} \times \bar{\mathbb{G}}^5$ and (s_1, s_2, s_3) is the unique value that makes the message valid, so $\mathcal{W}$ simulates Hybrid 5 correctly until WrongGuessS happens.

Next, we show that an $H_2(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*)$ query must be made by $\mathcal{A}$ (rather than programmed by $\mathcal{W}$ itself). Since WrongGuessS happens in the k-th sub-session, $\mathcal{A}$'s message to S $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ must be valid and not forwarded from an honest C instance. If $H_2(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*)$ is programmed by $\mathcal{W}$, i.e., $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*)$ is part of an honest message from C, then the remaining parts of this message must be equal to (s_1^*, s_2^*, s_3^*), since the honest message is valid. But then the entire message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ is forwarded, which forms a contradiction.

In sum, if WrongGuessS happens in the k-th sub-session, then $\mathcal{W}$ simulates the hybrids correctly until that point and outputs $I > 0$ such that the I-th query input to H_2 is $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*)$. So

$$\Pr[\mathsf{WrongGuess}] \leq q_{SS} \cdot \mathbf{Acc}_{\mathcal{W}}.$$

Applying the forking lemma. By the forking lemma, $\mathcal{F}^{\mathcal{W}}$'s accepting probability, i.e., the probability that $\mathcal{F}^{\mathcal{W}}$ outputs $(1, \star, \star)$, satisfies

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

so

$$\Pr[\mathsf{WrongGuess}] \leq q_{SS} \left(\sqrt{q \cdot \mathbf{Frk}_{\mathcal{F}^{\mathcal{W}}}} + \frac{q}{p} \right).$$

Analysis of $\mathcal{R}$. Next, we analyze the "intermediate reduction" $\mathcal{R}$, again assuming its input is uniformly random and all oracles are implemented as described. Assume that $\mathcal{F}^{\mathcal{W}}$ accepts, i.e., it outputs $(1, \star, \star)$. We argue that none of $\mathcal{R}$'s failure events can happen:

- Note that $\mathcal{W}$'s random coins are fixed, and its two runs have the same inputs up to the I-th query of H_2 (whose input is $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*)$); thus, $\mathcal{W}$'s two runs are identical up to that point, and in particular, its two queries to $\mathcal{O}$ have the same input A_1^* (unless $\mathcal{W}$ outputs 0 before making such a query, which will cause $\mathcal{F}^{\mathcal{W}}$ to output 0). Therefore, $\mathsf{TwoOQueries}$ cannot happen.
- $\mathsf{NotAccept}$ cannot happen by assumption that $\mathcal{F}^{\mathcal{W}}$ accepts.
- By Lemma 1, $\alpha = \pi\beta$ only if $A_1 = \left(A_3 A_5^{-\frac{1}{\log_{g_1} g_3}}\right)^{\pi + \log_{g_1} g_2}$, in which case $\mathcal{W}$'s query to $\mathcal{O}'$ would result in 1, causing $\mathcal{W}$ (and thus $\mathcal{F}^{\mathcal{W}}$) to output 0, so $\mathsf{NoWrongGuessS}$ cannot happen.

We conclude that as long as $\mathcal{F}^{\mathcal{W}}$ accepts, $\mathcal{R}$ does not abort and thus outputs $W = (KZ^{-\beta})^{\frac{1}{\alpha - \pi\beta}}$. Recall that (V, g_1, U) are embedded as (X, Y, Z), and by the analysis in the overview, $\left(V, g_1, U, (KU^{-\beta})^{\frac{1}{\alpha - \pi\beta}}\right)$ forms a DH tuple in the hybrid; therefore, $\mathcal{R}$'s output is correct. We have

$$\mathbf{Frk}_{\mathcal{F}^{\mathcal{W}}} \leq \mathbf{Adv}_{\mathcal{R}},$$

so

$$\Pr[\mathsf{WrongGuess}] \leq q_{SS}\left(\sqrt{q \cdot \mathbf{Adv}_{\mathcal{R}}} + \frac{q}{p}\right).$$

Analysis of $\mathcal{R}_{1ODH\text{-}1}$. Finally, since $\mathcal{R}$ fails if any $\mathcal{O}'$ query returns 1, responding to all queries with 0 cannot reduce the probability of success, so

$$\mathbf{Adv}_{\mathcal{R}} \leq \mathbf{Adv}^{\mathrm{1ODH}}_{\mathcal{R}_{\mathrm{1ODH\text{-}1}}}.$$

We conclude that (plugging in $q = q_{H_2} + 1$)

$$\mathbf{Dist}^{5,6}_{\mathcal{Z}} \leq \Pr[\mathsf{WrongGuess}] \leq q_{SS}\left[\sqrt{(q_{H_2}+1)\mathbf{Adv}^{\mathrm{1ODH}}_{\mathcal{R}_{\mathrm{1ODH\text{-}1}}}} + \frac{q_{H_2}+1}{p}\right].$$

Remark 2. If we reduce to the GDH assumption instead of 1ODH, then a single DH oracle (with multiple queries) suffices, and we can apply the forking lemma in a more "standard" manner. However, GDH is a stronger assumption that yields exactly the same security bound, so there is no advantage using it.

Hybrid 7 (S *outputs random session key on forwarded message*). When S receives a message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ which is equal to a message sent by C, S outputs $SK \leftarrow \{0,1\}^\lambda$ instead of computing SK normally. (Note that this includes $\mathcal{A}$ forwarding the protocol message from C to S in the same sub-session, or sending the honest C-to-S message to another sub-session.)

Let $\mathsf{Forward}$ be the event that $\mathcal{A}$ forwards a message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ to S and queries $H_4(A_1^*, B, K)$. Hybrids 6 and 7 are identical unless $\mathsf{Forward}$ happens. We now construct a reduction $\mathcal{R}_{\mathrm{1ODH\text{-}2}}$ that reduces $\mathsf{Forward}$ to the 1ODH problem.

Description of $\mathcal{R}_{1ODH\text{-}2}$. $\mathcal{R}_{\text{1ODH-2}}$, on 1ODH challenge $(X, Y, Z) \in \bar{\mathbb{G}}^3$ and given oracles H and $\mathcal{O}$, samples $k_{\mathsf{C}}, k_{\mathsf{S}} \leftarrow [1, q_{SS}]$ (as a guess that k_{C} and k_{S} are the sub-sessions involved in the first occurrence of Forward). When generating CRS parameters, $\mathcal{R}_{\text{1ODH-2}}$ sets $g_1 \coloneqq X$.

In the C instance for the k_{C}-th sub-session, $\mathcal{R}_{\text{1ODH-2}}$ sets $A_1 \coloneqq Y$ and generates the remaining part of the C-to-S message as what the wrapper algorithm does in Hybrid 6.

In the S instance for the k_{S}-th sub-session, $\mathcal{R}_{\text{1ODH-2}}$ programs $\mathcal{D}(\tau, B) \coloneqq Z$.

When the C instance for the k_{C}-th sub-session receives a message B^*, $\mathcal{R}_{\text{1ODH-2}}$ queries $h \coloneqq \mathcal{O}\left(\mathcal{D}(\tau', B^*)^{\frac{1}{\pi'+z_2}}\right)$. If there was a previous query $H_2(A_1, B^*, K')$ with $H(K') = h$, $\mathcal{R}_{\text{1ODH-2}}$ lets C output $SK' \coloneqq H_4(A_1, B^*, K')$. Otherwise, $\mathcal{R}_{\text{1ODH-2}}$ lets C output $SK' \leftarrow \{0,1\}^\lambda$, and on a future $H_4(A_1, B^*, K^*)$ query for any K^*, $\mathcal{R}_{\text{1ODH-2}}$ checks if $H(K^*) = h$, programming the result to SK' if so and answering the query normally otherwise.

When the S instance for the k_{S} sub-session receives a message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$, if this is not the message sent by the C instance for the k_{C}-th sub-session, $\mathcal{R}_{\text{1ODH-2}}$ aborts. Otherwise, $\mathcal{R}_{\text{1ODH-2}}$ lets S output $SK \leftarrow \{0,1\}^\lambda$.

Finally, on an $H_4(A_1^*, B, K^*)$ query for any K^* (including retroactively those made before $\mathcal{A}$ sends the protocol message containing A_1^*), $\mathcal{R}_{\text{1ODH-2}}$ computes $W \coloneqq (K^*)^{\pi+z_2}$ and answers the query as normal. When $\mathcal{A}$ halts, $\mathcal{R}_{\text{1ODH-2}}$ outputs all values of W computed this way.

Analysis of $\mathcal{R}_{1ODH\text{-}2}$. Assume that Forward happens, and $\mathcal{R}_{\text{ODH-2}}$'s guesses are correct. We first argue that $\mathcal{R}_{\text{1ODH-2}}$ simulates Hybrid 6 correctly unless and until Forward happens. Note that only the behavior of the C instance for the k_{C}-th sub-session and the behavior of the S sub-session for the k_{S}-th instance differ from Hybrid 6.

For the C instance, the C-to-S message is distributed identically as in Hybrid 6, by the same argument that the wrapper algorithm in Hybrid 6 simulates this message correctly. Regarding C's output SK', it is defined as $H_4(A_1, B^*, K')$ where K' satisfies $H(K') = \mathcal{O}\left(\mathcal{D}(\tau', B^*)^{\frac{1}{\pi'+z_2}}\right)$. Note that $x = \log_{g_1} A_1 = \log_X Y$, so

$$\mathcal{O}\left(\mathcal{D}(\tau', B^*)^{\frac{1}{\pi'+z_2}}\right) = H\left(\left(\mathcal{D}(\tau', B^*)^{\frac{1}{\pi'+z_2}}\right)^x\right) = H(\mathcal{D}(\tau', B^*)^a),$$

i.e., $K' = \mathcal{D}(\tau', B^*)^a$. This again matches Hybrid 6.

For the S instance, S's message $\mathcal{E}(\tau, Z)$ is distributed identically to the message in Hybrid 6. When S receives a message, if it does not trigger Forward, then in Hybrid 6 S's output SK is independent of $\mathcal{A}$'s view, and it remains so until Forward happens, so $\mathcal{R}_{\text{1ODH-2}}$ simulates Hybrid 6 correctly until then.

We now show that $\mathcal{R}_{\text{1ODH-2}}$ correctly solves the 1ODH instance (X, Y, Z). When Forward happens, there is a query $H_4(A_1^*, B, K)$ with

$$K = (A_1^*)^b = A_1^b = Y^b$$

where

$$b = \log_V \mathcal{D}(\tau, B) = \frac{\log_{g_1} \mathcal{D}(\tau, B)}{\log_{g_1} V} = \frac{\log_X Z}{\pi + z_2},$$

so $\mathcal{R}_{\text{1ODH-2}}$'s corresponding output

$$W = K^{\pi+z_2} = \left(Y^{\frac{\log_X Z}{\pi+z_2}}\right)^{\pi+z_2} = Y^{\log_X Z}$$

and thus $\mathcal{R}_{\text{1ODH-2}}$ succeeds. Therefore,

$$\mathbf{Dist}_{\mathcal{Z}}^{6,7} \leq \Pr[\mathsf{Forward}] \leq q_{SS}^2 \cdot \mathbf{Adv}^{\text{1ODH}}_{\mathcal{R}_{\text{1ODH-1}}}.$$

Hybrid 8 (C *outputs random session key on incorrect password guess or forwarded message with non-matching password*)**.** When C receives a message B^*, if B^* is not an IC encryption under τ' and also not equal to a message B sent by S with $\tau = \tau'$, C outputs $SK' \leftarrow \{0,1\}^\lambda$ instead of computing SK' normally.

Let $\mathsf{WrongGuessC}$ be the event that $\mathcal{A}$ sends a message B^* to C which is neither an IC encryption under τ' nor a forwarded message with a matching password, and queries $H_4(A_1, B^*, K')$ where A_1 and K' are the same values that C would use in Hybrid 7. Hybrids 7 and 8 are identical unless $\mathsf{WrongGuessC}$ happens. We now construct a reduction $\mathcal{R}_{\text{CDH-2}}$ that reduces $\mathsf{WrongGuessC}$ to the CDH problem.

Description of $\mathcal{R}_{\text{CDH-2}}$. $\mathcal{R}_{\text{CDH-2}}$, on challenge $(X, Y, Z) \in \bar{\mathbb{G}}^3$, samples $k \leftarrow [1, q_{SS}]$ (as a guess that $\mathsf{WrongGuessC}$ first happens in the k-th sub-session). When generating CRS parameters, $\mathcal{R}_{\text{CDH-2}}$ sets $g_1 \coloneqq X$.

In the C instance for the k-th sub-session, $\mathcal{R}_{\text{CDH-2}}$ sets $A_1 \coloneqq Y$ and generates the remaining part of the C-to-S message as what the wrapper algorithm does in Hybrid 6.

On IC decryption query $\mathcal{D}(\tau', B^*)$ (either by $\mathcal{A}$ or internally), $\mathcal{R}_{\text{CDH-2}}$ samples $\beta \leftarrow \mathbb{Z}_p^*$ and sets $\mathcal{D}(\tau', B^*) \coloneqq Z^\beta$.

When C receives B^*, if B^* is an IC encryption under τ', or if $B^* = B$ and $\mathsf{pw}' = \mathsf{pw}$, then $\mathsf{WrongGuessC}$ does not happen in this sub-session, so $\mathcal{R}_{\text{CDH-2}}$ aborts. Otherwise C outputs $SK' \leftarrow \{0,1\}^*$.

Finally, on an $H_4(A_1, B^*, K^*)$ query for any K^*, $\mathcal{R}_{\text{CDH-2}}$ computes $W \coloneqq (K^*)^{\frac{\pi'+z_2}{\beta}}$ and answers the query via lazy sampling. When $\mathcal{A}$ is done with all H_4 queries, $\mathcal{R}_{\text{CDH-2}}$ outputs all values of W computed this way.

Analysis of $\mathcal{R}_{\text{CDH-2}}$. Assume that $\mathsf{WrongGuessC}$ happens, and $\mathcal{R}_{\text{CDH-2}}$'s guess is correct. It is straightforward to see that $\mathcal{R}_{\text{CDH-2}}$ simulates Hybrid 8 correctly. We now show that $\mathcal{R}_{\text{CDH-2}}$ successfully solves the CDH instance (X, Y, Z). Let $H_4(A_1, B^*, K')$ be the query that triggers $\mathsf{WrongGuessC}$; then

$$K' = \mathcal{D}(\tau', B^*)^a = (Z^\beta)^a$$

where

$$a = \frac{x}{\pi' + z_2} = \frac{\log_{g_1} A_1}{\pi' + z_2} = \frac{\log_X Y}{\pi' + z_2},$$

so $\mathcal{R}_{\text{CDH-2}}$'s corresponding output

$$W = (K')^{\frac{\pi'+z_2}{\beta}} = \left(Z^{\frac{\beta \log_X Y}{\pi'+z_2}}\right)^{\frac{\pi'+z_2}{\beta}} = Z^{\log_X Y}$$

and thus $\mathcal{R}_{\text{CDH-2}}$ succeeds. Therefore,

$$\mathbf{Dist}_{\mathcal{Z}}^{7,8} \leq \Pr[\mathsf{WrongGuessC}] \leq q_{SS} \cdot \mathbf{Adv}^{\text{CDH}}_{\mathcal{R}_{\text{CDH-2}}}.$$

Hybrid 9 (C *outputs random session key on forwarded message with matching password*)**.** When C receives a message B^* which is equal to a message sent by a S instance in the same session with a matching password, C outputs $SK' \leftarrow \{0,1\}^\lambda$ instead of computing SK' normally.

This reduction $\mathcal{R}_{\text{1ODH-3}}$ is largely similar to the previous two hybrids, so we provide only a summary: $\mathcal{R}_{\text{1ODH-3}}$, on challenge (X, Y, Z), sets $g_1 \coloneqq X$, guesses a random C and S sub-session in which the bad event first happens, uses Y as $\mathcal{D}(\tau, B)$ and Z as A_1, generate the rest of the C-to-S message as in Hybrid 6, uses its oracles $\mathcal{O}$ and H to simulate S's computation of SK, and generates outputs on $H_4(A_1, B, K^*)$ queries as $W \coloneqq (K^*)^{\pi'+z_2}$. Ultimately,

$$\mathbf{Dist}_{\mathcal{Z}}^{8,9} \leq q_{SS}^2 \cdot \mathbf{Adv}^{\text{1ODH}}_{\mathcal{R}_{\text{1ODH-3}}}.$$

Hybrid 10 (*rule out client impersonation after server compromise*)**.** When S receives a valid message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ which is not equal to a message sent by C, if $A_1^* = \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi+z_2}$ and $\mathcal{A}$ has not queried $H_1(\mathsf{pw})$, output $\mathsf{ClientImpersonation}$ and halt.

The indistinguishability argument uses the forking lemma and is somewhat similar to the one in Hybrid 6; however, here we reduce to DL rather than 1ODH.

Description of the wrapper algorithm $\mathcal{W}$. Let $q = q_{H_2} + 1$. For a fixed adversary $\mathcal{A}$, we construct the wrapper algorithm $\mathcal{W}$ as follows. $\mathcal{W}$ takes as input $(X, Y) \in \bar{\mathbb{G}}^2$ and $(h_1, \ldots, h_q) \in \mathbb{Z}_p^q$, and is equipped with an oracle $\mathcal{O} : \mathbb{G}^2 \to \{0,1\}$ which, on input (N, M), returns 1 if $N^{\log_X Y} = M$ and 0 otherwise. (This $\mathcal{O}$ is what we call $\mathcal{O}'$ in Hybrid 6.)

$\mathcal{W}$, on input (X, Y) and $(h_1, \ldots, h_q)$, samples $k_S \leftarrow [1, q_S]$ and $k_{SS} \leftarrow [1, q_{SS}]$ (as a guess that $\mathsf{ClientImpersonation}$ happens first in the k_{SS}-th sub-session, which is part of the k_S-th session). When generating CRS parameters, $\mathcal{W}$ sets $g_1 \coloneqq X$. On the k_S-th $\mathsf{StorePwdfile}$ message, $\mathcal{W}$ sets $V \coloneqq Y$ (below we only describe $\mathcal{W}$'s behaviro within this session). If $\mathcal{A}$ ever queries $H_1(\mathsf{pw})$ for this session, this implies that $\mathsf{ClientImpersonation}$ does not happen in this session, so $\mathcal{W}$ outputs 0 and halts.

On the i-th unique query to H_2 (by $\mathcal{A}$ or by $\mathcal{W}$ itself), $\mathcal{W}$ returns h_i. Note that $\mathcal{W}$ only uses h_i to answer $H_2(ssid, \dots)$ queries where $ssid$ is the ID for the k_{SS}-th sub-session; for H_2 queries for other sub-sessions, $\mathcal{W}$ answers with lazy sampling.

In every C instance with $\mathsf{pw}' = \mathsf{pw}$ (i.e., the instances for which $\mathcal{W}$ does not know the value of $\pi' = \pi$), C computes $A_1 \coloneqq Y^a$ and generates the rest of the C-to-S message as in Hybrid 6.

When any S instance not for the k_{SS}-th sub-session receives a message, S outputs $SK \leftarrow \{0,1\}^\lambda$.

For the k_{SS}-th sub-session, if it is not part of the k_S-th session, $\mathcal{W}$ outputs 0 and halts. When the S instance receives a message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$, if the message is invalid or if $\mathcal{O}\left(A_3^* A_5^{*-\frac{1}{z_3}}, A_1^*\right) = 0$, $\mathcal{W}$ outputs 0 and halts. Otherwise, let I be the (unique) number for which the input to the I-th unique H_2 query is $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*)$ (such a query must be made by $\mathcal{W}$ to check if the message is valid, so such an I exists). Then $\mathcal{W}$ outputs $(I, (s_1^*, s_2^*, h_I, z_2))$ and halts. (If $\mathcal{A}$ halts before this occurs, $\mathcal{W}$ outputs 0 and halts.)

Description of the intermediate reduction $\mathcal{R}$. Next, we define $\mathcal{R}$, which takes the same inputs as $\mathcal{W}$ and uses the same oracle.

$\mathcal{R}$, on input (X, Y), runs the forking algorithm $\mathcal{F}^{\mathcal{W}}$ on (X, Y), answering $\mathcal{O}$ queries by querying its own oracle. When $\mathcal{F}^{\mathcal{W}}$ outputs $(b, \star, \star)$, if $b = 0$, $\mathcal{R}$ outputs $\mathsf{NotAccept}$ and halts. Otherwise $\mathcal{R}$ parses $\mathcal{F}^{\mathcal{W}}$'s output as $(1, (s_1, s_2, d, z_2), (s_1', s_2', d', z_2))$ and outputs $\frac{s_1' d - s_1 d'}{s_2' d - s_2 d'} + z_2$ (showing that $s_2' d - s_2 d' \neq 0$ is left for the analysis).

Description of the reduction $\mathcal{R}_{DL}$. Finally, we construction the reduction $\mathcal{R}_{\mathrm{DL}}$. On input (X, Y), $\mathcal{R}_{\mathrm{DL}}$ runs $\mathcal{R}$ on (X, Y), answering $\mathcal{O}$ queries with 1 regardless of input. When $\mathcal{R}$ outputs, $\mathcal{R}_{\mathrm{DL}}$ copies its output.

Analysis of $\mathcal{W}$. First, we analyze the behavior of $\mathcal{W}$ when all inputs are uniformly sampled and $\mathcal{O}$ is implemented as specified. By an argument similar to Hybrid 6, $\mathcal{W}$ simulates the behavior of Hybrid 10 correctly unless and until either $\mathcal{W}$ outputs or $\mathsf{ClientImpersonation}$ occurs, and if the first occurrence of $\mathsf{ClientImpersonation}$ is indeed in the k_{SS}-th sub-session, which is in the k_S-th session, $\mathcal{W}$ will output a nonzero value for I. Therefore,

$$\Pr[\mathsf{ClientImpersonation}] \leq q_S q_{SS} \cdot \mathbf{Acc}_{\mathcal{W}},$$

and the forking lemma gives

$$\mathbf{Acc}_{\mathcal{W}} \leq q_S q_{SS} \left(\sqrt{q \cdot \mathbf{Frk}_{\mathcal{F}^{\mathcal{W}}}} + \frac{q}{p} \right).$$

Analysis of $\mathcal{R}$. Next, we analyze the "intermediate reduction" $\mathcal{R}$, again assuming its input is uniformly random and its oracle is implemented as described. Assume that $\mathcal{F}^{\mathcal{W}}$ accepts, i.e., it outputs $(1, \star, \star)$. Note that $\mathcal{W}$'s simulation of

Hybrid 10 implicitly defines $\pi := \log_X Y - z_2$ (which $\mathcal{W}$ does not know). Since $\mathcal{W}$'s random coins are fixed, and its two runs have the same inputs up to the I-th query of H_2, $\mathcal{W}$'s two runs are identical up to that query. Note also that the message which triggers ClientImpersonation is the input to that query, so the conditions for Lemma 1 hold. By Lemma 1,

$$s_1' d - s_1 d' = \pi(s_2' d - s_2 d'),$$

and $s_2' d - s_2 d' \neq 0$ because $A_1 \neq e$, so $\pi = \frac{s_1' d - s_1 d'}{s_2' d - s_2 d'}$ and $\log_X Y = \pi + z_2$, so $\mathcal{R}$'s output is correct as long as $\mathcal{F}^{\mathcal{W}}$ accepts. Therefore,

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

Analysis of $\mathcal{R}_{DL}$. Finally, since $\mathcal{R}$ fails if any $\mathcal{O}$ query returns 0, responding to all queries with 1 cannot reduce the probability of success, so

$$\mathbf{Adv}_{\mathcal{R}} \leq \mathbf{Adv}^{DL}_{\mathcal{R}_{\mathrm{DL}}}.$$

We conclude that (plugging in $q = q_{H_2} + 1$)

$$\mathbf{Dist}^{9,10}_{\mathcal{Z}} \leq \Pr[\mathsf{ClientImpersonation}] \leq q_S q_{SS} \left[\sqrt{(q_{H_2}+1)\mathbf{Adv}^{\mathrm{DL}}_{\mathcal{R}_{\mathrm{DL}}}} + \frac{q_{H_2}+1}{p} \right].$$

Hybrid 11 (*rule out $\mathcal{A}$ predicting hash value without querying or compromising* S). When S receives a valid message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ with $A_1^* = \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi+z_2}$, if $\mathcal{A}$ has neither queried $H_1(\mathsf{pw})$ nor stolen a server file generated with pw, S outputs $SK \leftarrow \{0,1\}^\lambda$. Similarly, when C receives a message B which is the output of an $\mathcal{E}$ query under τ', if $\mathcal{A}$ has neither queried $H_3(\mathsf{pw}')$ nor stolen a server file generated with pw', C outputs $SK' \leftarrow \{0,1\}^\lambda$.

Suppose $\mathcal{A}$ sends a valid message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ with $A_1^* = \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi+z_2}$, before querying $H_1(\mathsf{pw})$ or stealing a server file generated with pw. For any message from $\mathcal{A}$ to S with $A_1^* \neq e$, there is only one value of $\pi = H_1(\mathsf{pw})$ which would make the equality hold. However, note that the behavior of S is independent of $H_1(\mathsf{pw})$ unless and until $\mathcal{A}$ queries $H_1(\mathsf{pw})$ or steals a server file generated by pw, so the probability that this message causes a difference between Hybrids 10 and 11 is at most $\frac{1}{p}$. $\mathcal{A}$'s message to C can be argued about similarly. In total, there are up to $2q_{SS}$ protocol messages sent by $\mathcal{A}$, so

$$\mathbf{Dist}^{10,11}_{\mathcal{Z}} \leq \frac{2q_{SS}}{p}.$$

Comparison between Hybrid 11 and the ideal world. We now claim that Hybrid 11 and the ideal world are identical in $\mathcal{Z}$'s view, where the challenger in Hybrid 11 corresponds to the combination of the simulator SIM and the functionality $\mathcal{F}_{\mathrm{aPAKE}}$ in the ideal world. We compare $\mathcal{Z}$'s views in Hybrid 11 and in the ideal world message by message:

C-*to*-S *message:* In Hybrid 4, C samples $x \leftarrow \mathbb{Z}_p^*$ and computes $A_1 := g_1^x$. The computation of A_2, A_4, A_5, A_6, s_3 never changes in the hybrids, i.e., C samples $k, r_1, r_2, r_3 \leftarrow \mathbb{Z}_p$ and computes $A_2 := g_1^{r_1} g_2^{r_2}$, $A_4 := g_1^{r_2+r_3}$, $A_5 := g_3^k$, $A_6 := g_3^{r_3}$, and $s_3 := k + dr_3$. Then in Hybrid 5, C samples $a' \leftarrow \mathbb{Z}_p$ and computes $A_3 := g_1^{a'+k}$, $s_1 := x - z_2 a' + dr_1$, and $s_2 := a' + dr_2$ (where $d = H_2(A_1, A_2, A_3, A_4, A_5, A_6)$). The definitions of $A_1, A_2, A_3, A_4, A_5, A_6, s_1, s_2, s_3$ match step 3 of SIM.

S-*to*-C *message:* In Hybrid 4, S samples $y \leftarrow \mathbb{Z}_p$ and computes $B := \mathcal{E}(\perp, g_1^y)$. The definition of B matches step 4 of SIM.

C*'s output:* On receiving protocol message B^*,

- If $B^* = B$, i.e., $\mathcal{A}$ forwards the S-to-C message without modification: In Hybrids 8 and 9 C outputs $SK' \leftarrow \{0,1\}^\lambda$. This matches step 5(a) of SIM.
- If $B^* := \mathcal{E}(\tau, \star)$ for some τ: Let pw^* be the unique value satisfying $H_2(\mathsf{pw}^*) = \tau$, if one exists (note that at most one such values exists since H_3 collisions are excluded in Hybrid 2).
  - If pw^* does not exist or if $\mathsf{pw}^* \neq \mathsf{pw}'$, then B^* is not an IC encryption under τ', so in Hybrid 8 C outputs $SK' \leftarrow \{0,1\}^\lambda$. This matches step 5(b), "wrong guess" case, step 5(c), "wrong guess" case, and step 5(d) of SIM.
  - If $\mathsf{pw}^* = \mathsf{pw}'$ and $\mathcal{A}$ has either queried $H_3(\mathsf{pw})$ or stolen a server file for *sid* with $\mathsf{pw} = \mathsf{pw}'$,, then in Hybrid 4 C computes $K' := \mathcal{D}(\tau', B^*)^a$ (where $a = \frac{x}{\pi'+z_2} = \frac{x}{\pi+z_2}$) and $SK' := H_4(A_1, B^*, K')$. This matches step 5(b), "correct guess" case and step 5(c), "correct guess" case of SIM.
  - If $\mathsf{pw}^* = \mathsf{pw}'$ and $\mathcal{A}$ has not queried $H_3(\mathsf{pw})$ or stolen a server file for *sid* with $\mathsf{pw} = \mathsf{pw}'$, then in Hybrid 2 C outputs $SK' \leftarrow \{0,1\}^\lambda$. This matches step 5(d) of SIM.
- If neither of the previous cases happens:
  - If $B^* = \mathcal{E}(\tau^*, \star)$ for some $\tau^* \neq \tau'$, then B^* is not an IC encryption under τ', so in Hybrid 8 C outputs $SK' \leftarrow \{0,1\}^\lambda$. This matches step 5(c)(i), "wrong guess" case of SIM.
  - If $B^* = \mathcal{E}(\tau^*, \star)$ for $\tau^* = \tau'$, then in Hybrid 4 C computes $K' := \mathcal{D}(\tau^*, B^*)^a$ (where $a = \frac{x}{\pi'+z_2} = \frac{x}{H_1(\mathsf{pw}^*)+z_2}$) and $SK' := H_4(A_1, B^*, K')$. This matches step 5(c)(i), "correct guess" case of SIM.
  - If B^* is not the result of an $\mathcal{E}$ query, then B^* is not an IC encryption under τ', so in Hybrid 8 C outputs $SK' \leftarrow \{0,1\}^\lambda$. This matches step 5(c)(ii) of SIM.

We conclude that in all cases, C's behavior on receiving B^* is identical to the ideal world.

S*'s output:* On receiving protocol message $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$,

– If the message S receives is invalid: Then S outputs $\perp$ and halts in the real world, which never changes in the hybrids. This behavior matches step 6(a) of SIM.
– If $A_1^* = A_1$, i.e., $\mathcal{A}$ forwards the C-to-S message A_1 without modification: In Hybrid 7 S outputs $SK \leftarrow \{0,1\}^\lambda$. This matches step 6(b) of SIM.
– If neither of the previous cases happens:
  - If $A_1^* = \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi^*+z_2}$ for some $\pi^* = H_1(\mathsf{pw}^*) \neq \pi$, then in Hybrid 6 S outputs $SK \leftarrow \{0,1\}^\lambda$. This matches step 6(c)(i), "wrong guess" case of SIM.
  - If $A_1^* = \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi+z_2}$ and $\mathcal{A}$ has queried $H_1(\mathsf{pw})$, then in Hybrid 4 S computes $K := (A_1^*)^b$ (where $b = \frac{1}{\pi+z_2}\log_{g_1}\mathcal{D}(\tau, B)$) and $SK := H_4(A_1^*, B, K)$. This matches step 6(c)(i), "correct guess" case of SIM.
  - If $A_1^* = \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi+z_2}$, $\mathcal{A}$ has stolen the server file but has never queried $H_1(\mathsf{pw})$, then the game challenger in Hybrid 10 outputs $\mathsf{ClientImpersonation}$ and halts. This matches step 6(c)(ii) of SIM.
  - If $A_1^* \neq \left(A_3^*(A_5^*)^{-\frac{1}{z_3}}\right)^{\pi^*+z_2}$ for any π^* which is the result of an H_1 query, then in Hybrid 6 S outputs $SK \leftarrow \{0,1\}^\lambda$. This matches step 6(c)(iii) of SIM.

We conclude that in all cases, S's behavior on receiving $(A_1^*, A_2^*, A_3^*, A_4^*, A_5^*, A_6^*, s_1^*, s_2^*, s_3^*)$ is identical to the ideal world.

S's password file: In both Hybrid 11 and the ideal world, when $\mathcal{A}$ sends $(\mathsf{StealPwdfile}, sid)$ to S (assuming a $(\mathsf{StorePwdfile}, sid, \mathsf{pw})$ command has been sent), $\mathcal{A}$ receives $(V = g_1^\pi g_2, \tau)$ where $\pi = H_1(\mathsf{pw})$ and $\tau = H_3(\mathsf{pw})$. The only difference lies in when π and τ are defined: in Hybrid 11 they are defined when $\mathcal{A}$ queries resp. $H_1(\mathsf{pw})$ and $H_3(\mathsf{pw})$, whereas in the ideal world, if the $(\mathsf{StealPwdfile}, sid)$ message comes before the $H_1(\mathsf{pw})$ or $H_3(\mathsf{pw})$ query, then SIM first samples π and τ uniformly at random and later programs $H_1(\mathsf{pw})$ to be π and $H_3(\mathsf{pw})$ to be τ. This makes no external difference in $\mathcal{Z}$'s view.

We conclude that Hybrid 11 is identical to the ideal world, and thus the real world (Hybrid 0) and the ideal world are indistinguishable. This completes the proof.

Acknowledgments

The authors were supported by NSF SaTC Award 2453883.

References

1. Abdalla, M., Barbosa, M., Bradley, T., Jarecki, S., Katz, J., Xu, J.: Universally composable relaxed password authenticated key exchange. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 278–307. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_10
2. Abdalla, M., Barbosa, M., Katz, J., Loss, J., Xu, J.: Algebraic adversaries in the universal composability framework. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part III. LNCS, vol. 13092, pp. 311–341. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92078-4_11
3. Abdalla, M., Bellare, M., Rogaway, P.: The oracle Diffie-Hellman assumptions and an analysis of DHIES. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 143–158. Springer, Berlin, Heidelberg (Apr 2001). https://doi.org/10.1007/3-540-45353-9_12
4. Abdalla, M., Haase, B., Hesse, J.: Security analysis of CPace. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part IV. LNCS, vol. 13093, pp. 711–741. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92068-5_24
5. Bellare, M., Neven, G.: Multi-signatures in the plain public-key model and a general forking lemma. In: Juels, A., Wright, R.N., De Capitani di Vimercati, S. (eds.) ACM CCS 2006. pp. 390–399. ACM Press (Oct / Nov 2006). https://doi.org/10.1145/1180405.1180453
6. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. pp. 139–155. Springer, Berlin, Heidelberg (May 2000). https://doi.org/10.1007/3-540-45539-6_11
7. Bellovin, S.M., Merritt, M.: Encrypted key exchange: Password-based protocols secure against dictionary attacks. In: 1992 IEEE Symposium on Security and Privacy. pp. 72–84. IEEE Computer Society Press (May 1992). https://doi.org/10.1109/RISP.1992.213269
8. Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: New models and constructions. Cryptology ePrint Archive, Report 2013/833 (2013), https://eprint.iacr.org/2013/833
9. Bradley, T., Jarecki, S., Xu, J.: Strong asymmetric PAKE based on trapdoor CKEM. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part III. LNCS, vol. 11694, pp. 798–825. Springer, Cham (Aug 2019). https://doi.org/10.1007/978-3-030-26954-8_26
10. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. Tech. Rep. 260, Institute for Theoretical Computer Science, ETH Zurich (3 1997)
11. Cash, D., Kiltz, E., Shoup, V.: The twin Diffie-Hellman problem and applications. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 127–145. Springer, Berlin, Heidelberg (Apr 2008). https://doi.org/10.1007/978-3-540-78967-3_8
12. Dai, Y., Steinberger, J.P.: Indifferentiability of 8-round Feistel networks. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 95–120. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53018-4_4
13. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 33–62. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96881-0_2

14. Gentry, C., MacKenzie, P., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 142–159 (Aug 2006). https://doi.org/10.1007/11818175_9
15. Gu, Y., Jarecki, S., Krawczyk, H.: KHAPE: Asymmetric PAKE from key-hiding key exchange. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part IV. LNCS, vol. 12828, pp. 701–730. Springer, Cham, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84259-8_24
16. Haase, B., Labrique, B.: AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, Report 2018/286 (2018), https://eprint.iacr.org/2018/286
17. Hanai, S., Tanaka, K., Tezuka, M., Yoshida, Y.: Universally composable relaxed asymmetric password-authenticated key exchange. In: Galdi, C., Phan, D.H. (eds.) SCN 24, Part II. LNCS, vol. 14974, pp. 272–293. Springer, Cham (Sep 2024). https://doi.org/10.1007/978-3-031-71073-5_13
18. Hao, F., Bag, S., Chen, L., van Oorschot, P.C.: Owl: An augmented password-authenticated key exchange scheme. pp. 227–244. LNCS, Springer, Cham (Mar 2024). https://doi.org/10.1007/978-3-031-78679-2_12
19. Hesse, J.: Separating symmetric and asymmetric password-authenticated key exchange. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20. LNCS, vol. 12238, pp. 579–599. Springer, Cham (Sep 2020). https://doi.org/10.1007/978-3-030-57990-6_29
20. Hwang, J.Y., Jarecki, S., Kwon, T., Lee, J., Shin, J.S., Xu, J.: Round-reduced modular construction of asymmetric password-authenticated key exchange. In: Catalano, D., De Prisco, R. (eds.) SCN 18. LNCS, vol. 11035, pp. 485–504 (Sep 2018). https://doi.org/10.1007/978-3-319-98113-0_26
21. Jablon, D.P.: Strong password-only authenticated key exchange. SIGCOMM Comput. Commun. Rev. **26**(5), 5–26 (Oct 1996). https://doi.org/10.1145/242896.242897, https://doi.org/10.1145/242896.242897
22. Jablon, D.P.: Extended password key exchange protocols immune to dictionary attacks. In: 6th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 1997). pp. 248–255. IEEE Computer Society, Cambridge, MA, USA (Jun 18–20, 1997)
23. Januzelli, J., Roy, L., Xu, J.: Under what conditions is encrypted key exchange actually secure? In: EUROCRYPT 2025, Part II. pp. 451–481. LNCS, Springer, Cham (Jun 2025). https://doi.org/10.1007/978-3-031-91124-8_16
24. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 456–486. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_15
25. Jutla, C.S., Roy, A.: Smooth NIZK arguments. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018, Part I. LNCS, vol. 11239, pp. 235–262. Springer, Cham (Nov 2018). https://doi.org/10.1007/978-3-030-03807-6_9
26. Kiefer, F., Manulis, M.: Zero-knowledge password policy checks and verifier-based PAKE. In: Kutylowski, M., Vaidya, J. (eds.) ESORICS 2014, Part II. LNCS, vol. 8713, pp. 295–312. Springer, Cham (Sep 2014). https://doi.org/10.1007/978-3-319-11212-1_17
27. McQuoid, I., Xu, J.: An efficient strong asymmetric PAKE compiler instantiable from group actions. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part VIII. LNCS, vol. 14445, pp. 176–207. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8742-9_6

28. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 31–53. Springer, Berlin, Heidelberg (Aug 1993). https://doi.org/10.1007/3-540-48071-4_3
29. Santos, B.F.D., Gu, Y., Jarecki, S., Krawczyk, H.: Asymmetric PAKE with low computation and communication. In: Dunkelman, O., Dziembowski, S. (eds.) EUROCRYPT 2022, Part II. LNCS, vol. 13276, pp. 127–156. Springer, Cham (May / Jun 2022). https://doi.org/10.1007/978-3-031-07085-3_5
30. Santos, B.F.D., Gu, Y., Jarecki, S., Krawczyk, H.: Asymmetric PAKE with low computation and communication. https://www.youtube.com/watch?v=IYSnLrY8FPk (2022), talk at EUROCRYPT 2022
31. Shoup, V.: Security analysis of SPAKE2+. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part III. LNCS, vol. 12552, pp. 31–60. Springer, Cham (Nov 2020). https://doi.org/10.1007/978-3-030-64381-2_2
32. Xu, J., Zhao, Z.: Just how secure is SRP, really? Cryptology ePrint Archive, Report 2025/1869 (2025), https://eprint.iacr.org/2025/1869

A Proof of Lemma 1

Proof. Dividing equations (2) and (3) yields $g_1^{s_1-s_1'} g_2^{s_2-s_2'} = A_2^{d-d'}$, so $A_2 = g_1^{\frac{s_1-s_1'}{d-d'}} g_2^{\frac{s_2-s_2'}{d-d'}}$ and substituting into (2) gives $A_1 = g_1^{\frac{s_1'd-s_1d'}{d-d'}} g_2^{\frac{s_2'd-s_2d'}{d-d'}}$. Similarly, $A_3 = g_1^{\frac{s_2'd-s_2d'}{d-d'}+\frac{s_3'd-s_3d'}{d-d'}}$ and $A_5 = g_3^{\frac{s_3'd-s_3d'}{d-d'}}$. Substituting these into equation (4) gives

$$g_1^{\frac{s_1'd-s_1d'}{d-d'}} g_2^{\frac{s_2'd-s_2d'}{d-d'}} = \left(g_1^{\frac{s_2'd-s_2d'}{d-d'}+\frac{s_3'd-s_3d'}{d-d'}} g_3^{-\frac{s_3'd-s_3d'}{d-d'}\log_{g_3} g_1} \right)^{\pi+\log_{g_1} g_2}.$$

Raising both sides to the power of $d-d'$ and simplifying gives

$$\begin{aligned} g_1^{s_1'd-s_1d'} g_2^{s_2'd-s_2d'} &= \left(g_1^{s_2'd-s_2d'+s_3'd-s_3d'} g_3^{-(s_3'd-s_3d')\log_{g_3} g_1} \right)^{\pi+\log_{g_1} g_2} \\ &= \left(g_1^{s_2'd-s_2d'+s_3'd-s_3d'} g_1^{-(s_3'd-s_3d')} \right)^{\pi+\log_{g_1} g_2} \\ &= \left(g_1^{s_2'd-s_2d'} \right)^{\pi+\log_{g_1} g_2} \\ &= g_1^{\pi(s_2'd-s_2d')} g_2^{s_2'd-s_2d'} \\ &= g_1^{\pi(s_2'd-s_2d')}. \end{aligned}$$

Taking $\log_{g_1}$ of both sides yields equation (5):

$$s_1'd - s_1d' = \pi(s_2'd - s_2d').$$

B Performance Improvements

The server-side computation of both of our protocols can be reduced in the following manner.

Consider a modified version of our AGM protocol where S, instead of checking that $g_1^{s_1} g_2^{s_2} = A_1 A_2^d$ and subsequently computing $K := A_1^b$, first samples $r \leftarrow \mathbb{Z}_p$ and then computes $K := A_1^b (g_1^{s_1} g_2^{s_2} A_1^{-1} A_2^{-d})^r$. If the NIZK holds, the modified server behaves identically to the normal one, while if it doesn't hold, instead of explicitly aborting, S samples K as a uniformly random group element (by bijection with r, which is used nowhere else), so SK has high min-entropy in the view of the environment, and the protocol remains secure.

Similarly, our non-AGM protocol can be modified by replacing the NIZK checks and normal computation of K by sampling $r_1, r_2, r_3 \leftarrow \mathbb{Z}_p$ and computing $K := A_1^b (g_1^{s_1} g_2^{s_2} A_1^{-1} A_2^{-d})^{r_1} (g_1^{s_2+s_3} A_3^{-1} A_4^{-d})^{r_2} (g_3^{s_3} A_5^{-1} A_6^{-d})^{r_3}$, which remains secure for the same reasons.

The performance improvement from these changes — replacing several separate exponentiations and multi-exponentations in our AGM protocol and our

non-AGM-protocol with a single 4-base and 9-base multi-exponentation, respectively — is difficult to quantify precisely. The computation based on Shamir's trick would suggest that both are roughly equivalent to 1.3 normal exponentiations, but this is questionable: the overhead, especially in the latter case, is significant for some practical values of λ, and other techniques may experience larger discrepancies.