DIMACS Technical Report 2002-46
October 2002

The PAK suite: Protocols for Password-Authenticated Key Exchange

by

Philip MacKenzie[1]
Bell Laboratories
Lucent Technologies
Murray Hill, NJ 07974 USA
philmac@lucent.com

[1]Permanent Member

DIMACS is a partnership of Rutgers University, Princeton University, AT&T Labs-Research, Bell Labs, NEC Research Institute and Telcordia Technologies (formerly Bellcore).

DIMACS was founded as an NSF Science and Technology Center, and also receives support from the New Jersey Commission on Science and Technology.

ABSTRACT

In this paper we give a detailed formal description of the PAK password-authenticated key exchange protocol and some variants, and provide provide complete proofs of security which we believe are more straightforward than the original proofs. We also show a new general method (called the *Z-method*) for making these protocols resilient to server-compromise, so as to not allow an attacker that obtains password verification data from a server to then impersonate a user. When this method is applied to PAK, we call the resulting protocol PAK-Z. Finally, we discuss the current state-of-the-art in password-authenticated key exchange, with respect to both theory and practice.

1 Introduction

Given the current interest in using and standardizing strong password-authenticated key exchange protocols, we feel it is appropriate to develop a comprehensive document that (1) discusses the current state of the art in password-authenticated key exchange protocols, and (2) provides a comprehensive description of the PAK protocol and its variants, with full explanations of all design choices, and with proofs of security that are as simple and straightforward as possible. The hope is to give practitioners and standards bodies more information with which to implement and standardize these protocols. For completeness, in the appendix we also provide precise descriptions of the EKE protocol [7, 2] and some variants, and provide a complete proof for the basic EKE protocol.[1]

We assume the reader is familiar with the basic idea of strong password-authenticated key exchange, in which two parties share only a password (i.e., a short secret),[2] and want to run a protocol to compute a cryptographically strong shared secret key, using the password for authentication purposes in the protocol. The protocol should be strong in the sense that it should not allow an attacker to obtain any information about the password through simple eavesdropping, and only allow the attacker to gain information about one password per protocol session in an active attack.[3] Basically, this implies that the attacker is not able to obtain data with which to perform an *offline dictionary attack*, in which the attacker would run through a dictionary of possible passwords offline, checking each one for consistency with the data. A very good introduction and discussion of this problem may be found in Jablon [24] or Wu [41]. The seminal work in the field was the development of Encrypted Key Exchange (EKE) by Bellovin and Merritt [7, 8], and there has been a great deal of work since then, e.g., [2, 18, 23, 22, 40, 24, 25, 28, 29, 31, 32, 36, 41] (see `http://www.integritysciences.com` for more references). The PAK protocol and some variants (PPK, PAK-X) were originally developed in Boyko, MacKenzie, and Patel [12]. More PAK variants (PAK-R, PAK-EC, PAK-XTR, PAK-Y) were developed in [33]. The PAK protocol may be thought of as a type of encrypted key exchange with a very simple encryption function.

1.1 Current State of the Art

The past three years have seen a major growth in the field of password-authentication key exchange, both in theory and practice.

1.1.1 Theory

We begin our discussion on the theoretical side of the field. The original EKE protocols by Bellovin and Merritt [7] were groundbreaking in their ideas, but only included very rough security arguments. Certain instantiations were later shown to be insecure by Patel [34].

[1] The basic EKE protocol is slightly modified from the one from [2]. We were not able to obtain the security result claimed in [2] without the modifications.

[2] In particular, neither party knows a public key belonging to the other party.

[3] This may be generalized in some cases to obtaining information about a constant number of passwords, rather than just one.

Many subsequent protocols (e.g. [31, 36] and an early version of [24]), suffered the same fate. They were based on non-rigorous security arguments, and some instantiations were shown to be insecure. Eventually, protocol designers began to design protocols that were able to withstand all known attacks. Notable examples include SPEKE [24] and SRP [41].

Soon after these were published, sound theoretical models for password-authenticated key exchange were developed. The two main models are the *simulation model* and the *direct model*. These models are discussed in Appendix A.[4] Along with the development of sound theoretical models for password-authenticated key exchange came the development of some protocols that could be rigorously proven secure in these models. A precise specification of the EKE protocol (called EKE2) was given in [2], along with a sketch of a security proof. The PAK protocol and some variants were proven secure in [12]. The SNAPI protocol was proven secure in [32]. Some more PAK variants were proven secure in [33]. We caution that although these proofs are rigorous in some sense, they actually make a simplifying assumption that the hash functions behave like ideal random functions, or *random oracles* [3]. (The proof for EKE2 additionally relies on ideal ciphers.) For this reason, some prefer to call these proofs "arguments." However, we will continue to use the word proof, since they are actually proofs in the random oracle model, and they show that the protocols can only be broken by breaking an underlying cryptographic assumption, or by using some information about the hash function. So far, cryptographic schemes proven secure in the random oracle model have been very useful in practice.

Two protocols have been proven secure without relying on random oracles. The protocol in [28] only relies on a reasonably short public random string that is produced before the protocol begins. The protocol in [18] does not rely on a public random string or on random oracles, but is only proven secure when protocols sessions are *not* run concurrently.

Some of the protocols above provide a form of protection against server compromise, so that an attacker that obtains password verification data on the server does not obtain the actual password, and cannot then impersonate the user without performing an offline dictionary attack on the password verification data. SRP [41] and AMP [29] provide this inherently. Enhancements may be made to some of the other protocols to provide this, e.g., B-SPEKE [25], PAK-X [12], SNAPI-X [32], PAK-Y [33], and PAK-Z (this paper). Only the PAK and SNAPI variants have proofs of security (in the random oracle model).

1.1.2 Practice

There have been major developments in the practical side of the field also. The protocol that has been the most widely accepted so far has been SRP [41]. Versions of `telnet` and `ftp` using SRP have been available for a few years, and two IETF RFCs have been approved: RFC 2944 and RFC 2945. SPEKE [24] and PAK [12] have been used to a lesser extent, with PAK being used in Lucent Technologies' Plan9 operating system. All of these protocols, along with AMP [29], are currently being considered in the IEEE P1363.2 proposed standard for password-based techniques in public-key cryptography.

[4] Note: the names for these models are not universally recognized.

The issue of patents and licensing has played a role in the current makeup of the field, and probably will continue to play a role for the foreseeable future. The main patents in the field are two by Bellovin and Merrit [9, 10] and one by Jablon [27].

2 Definitions

Let κ be the cryptographic security parameter. Let $G_q \in \mathcal{G}$ denote a finite (cyclic) group of order q, where $|q| = \kappa$. Let g be a generator of G_q, and assume it is included in the description of G_q. We will assume the Computational Diffie-Hellman (CDH) assumption holds over G_q (see Section 6). Let t_{exp} be the time required to perform an exponentiation in G_q.

Notation. We denote by Ω the set of all functions H from $\{0,1\}^*$ to $\{0,1\}^\infty$. This set is provided with a probability measure by saying that a random H from Ω assigns to each $x \in \{0,1\}^*$ a sequence of bits each of which is selected uniformly at random. As shown in [3], this sequence of bits may be used to define the output of H in a specific set, and thus we will assume that we can specify that the output of a random oracle H be interpreted as a (random) element of G_q. See Section 5 for instantiations of this. Access to any public random oracle $H \in \Omega$ is given to all algorithms; specifically, it is given to the protocol P and the adversary $\mathcal{A}$. Assume that secret keys are drawn from $\{0,1\}^\kappa$.

A function $f : \mathbb{Z} \to [0,1]$ is negligible if for all $\alpha > 0$ there exists an $\kappa_\alpha > 0$ such that for all $\kappa > \kappa_\alpha$, $f(\kappa) < |\kappa|^{-\alpha}$. We say a multi-input function is negligible if it is negligible with respect to each of its inputs.

Signature schemes. We will use a signature scheme in the PAK-Z protocol, so we define signature schemes here. We will assume the signature scheme is existentially unforgeable against adaptive chosen message attacks (probably too strong) (see Section 6).

A *digital signature scheme* $\mathcal{S}$ is a triple $(\mathsf{Gen}_{\mathcal{S}}, \mathsf{Sig}, \mathsf{Verify})$ of algorithms, the first two being probabilistic, and all running in expected polynomial time. $\mathsf{Gen}_{\mathcal{S}}$ takes as input 1^κ and outputs a public key pair (pk, sk), i.e., $(pk, sk) \leftarrow \mathsf{Gen}_{\mathcal{S}}(1^\kappa)$. Sign takes a message m and a secret key sk as input and outputs a signature σ for m, i.e., $\sigma \leftarrow \mathsf{Sig}_{sk}(m)$. Verify takes a message m, a public key pk, and a candidate signature σ' for m as input and returns the bit $b = 1$ if σ' is a valid signature for m for the corresponding private key, and otherwise returns the bit $b = 0$. That is, $b \leftarrow \mathsf{Verify}_{pk}(m, \sigma')$. Naturally, if $\sigma \leftarrow \mathsf{Sig}_{sk}(m)$, then $\mathsf{Verify}_{pk}(m, \sigma) = 1$.

We assume that for the signature scheme used in PAK-Z, each secret key has a single κ-bit representation, and that a signature is of size $\kappa' \geq \kappa$, and such that the fraction of strings of length κ' that are valid signatures is at most $2^{-\kappa}$.

3 Model

For our proofs of security we use the model of [2] (which builds on [4] and [6], and is also used by [28]). This model is designed for the problem of authenticated key exchange (ake) between two parties, a client and a server, that share a secret. The goal is for them to engage in a protocol such that after the protocol is completed, they each hold a session key that is known to nobody but the two of them. This model is also extended to include explicit authentication between the client and server. We extend the model further to the case in which the server only stores some one-way function of the shared secret, and one who obtains this function of the secret would not be able to impersonate the client without performing an offline dictionary attack. The original model we call the *balanced* model, and our extension we call the *augmented* model.[5]

In the following, we will assume some familiarity with the model of [2].

Protocol participants. Let ID be a nonempty set of principals, each of which is either a client or a server. Thus $ID \stackrel{\text{def}}{=} Clients \cup Servers$, where $Clients$ and $Servers$ are finite, disjoint, nonempty sets. We assume each principal $U \in ID$ is labeled by a string, and we simply use U to denote this string.

Each client $C \in Clients$ has a secret password π_C and each server $S \in Servers$ has a vector $\pi_S = \langle \pi_S[C] \rangle_{C \in Clients}$. Entry $\pi_S[C]$ is the *password record*. In the balanced case we will have $\pi_S[C] = f(\pi_C)$, for some deterministic function f.[6] Let $Password_C$ be a (possibly small) set from which passwords for client C are selected. We will assume that $\pi_C \stackrel{R}{\leftarrow} Password_C$ (but our results easily extend to other password distributions). Clients and servers are modeled as probabilistic poly-time algorithms with an input tape and an output tape.

Execution of the protocol. A protocol P is an algorithm that determines how principals behave in response to inputs from their environment. In the real world, each principal is able to execute P multiple times with different partners, and we model this by allowing unlimited number of *instances* of each principal. Instance i of principal $U \in ID$ is denoted Π_i^U.

To describe the security of the protocol, we assume there is an adversary $\mathcal{A}$ that has complete control over the environment (mainly, the network), and thus provides the inputs to instances of principals. Formally, the adversary is a probabilistic algorithm with a distinguished query tape. Queries written to this tape are responded to by principals according to P; the allowed queries are formally defined in [2] and summarized here:

Send (U, i, M)**:** causes message M to be sent to instance Π_i^U. The instance computes what the protocol says to, state is updated, and the output of the computation is given to $\mathcal{A}$. If this query causes Π_i^U to accept or terminate, this will also be shown to $\mathcal{A}$.[7] To initiate a session between client C and server S, the adversary should send a message containing the server name S to an unused instance of C.

Execute (C, i, S, j)**:** causes P to be executed to completion between Π_i^C (where $C \in Clients$) and Π_j^S (where $S \in Servers$), and outputs the transcript of the execution.

[5]In [2], these models are called *symmetric* and *asymmetric*, but we use the terminology of P1363.2.

[6]This implies that the shared secret is really $f(\pi_C)$, instead of π_C. This generalization will be useful in our protocols, and in particular, will allow for some practical efficiency improvements.

[7]Recall that accepting implies generating a triple (pid, sid, sk), terminating implies accepting and no more messages will be output. To indicate the protocol not sending any more messages, but not terminating, *state* is set to DONE, but *term* is set to FALSE.

This query captures the intuition of a passive adversary who simply eavesdrops on the execution of P.

Reveal $(U,\ i)$**:** causes the output of the session key held by Π_i^U.

Test $(U,\ i)$**:** causes Π_i^U to flip a bit b. If $b = 1$ the session key sk_U^i is output; otherwise, a string is drawn uniformly from the space of session keys and output. A Test query may be asked at any time during the execution of P, but may only be asked once.

Corrupt (U)**:** If $U \in \mathit{Servers}$, this returns $\langle \pi_U[C] \rangle_{C \in \mathit{Clients}}$, and otherwise returns π_U.[8]

Partnering. A client or server instance that accepts holds a partner-id pid, session-id sid, and a session key sk. Then instances Π_i^C (with $C \in \mathit{Clients}$) and Π_j^S (with $S \in \mathit{Servers}$) are said to be *partnered* if both accept, they hold (pid, sid, sk) and (pid', sid', sk'), respectively, with $pid = S$, $pid' = C$, $sid = sid'$, and $sk = sk'$, and no other instance accepts with session-id equal to sid.

Freshness. We define two notions of freshness, as in [2]. Specifically, an instance Π_i^U is nfs-fresh (fresh with no requirement for forward secrecy) unless either (1) a $\mathsf{Reveal}\,(U,\ i)$ query occurs, (2) a $\mathsf{Reveal}\,(U',\ j)$ query occurs where $\Pi_{U'}^j$ is the partner of Π_i^U, or (3) a $\mathsf{Corrupt}\,(U')$ query occurs. (For convenience, when we do not make a requirement for forward secrecy, we simply disallow $\mathsf{Corrupt}$ queries.) An instance Π_i^U is fs-fresh (fresh with forward secrecy) unless either (1) a $\mathsf{Reveal}\,(U,\ i)$ query occurs, (2) a $\mathsf{Reveal}\,(U',\ j)$ query occurs where $\Pi_j^{U'}$ is the partner of Π_i^U, or (3) a $\mathsf{Corrupt}\,(U')$ query occurs before the Test query and a $\mathsf{Send}\,(U,\ i,\ M)$ query occurs for some string M.

Advantage of the adversary. We now formally define the authenticated key exchange (ake) advantage of the adversary against protocol P. Let $\mathsf{Succ}_P^{\mathrm{ake}}(\mathcal{A})$ be the event that $\mathcal{A}$ makes a single Test query directed to some fresh instance Π_i^U that has terminated, and eventually outputs a bit b', where $b' = b$ for the bit b that was selected in the Test query. The ake advantage of $\mathcal{A}$ attacking P is defined to be

$$\mathsf{Adv}_P^{\mathrm{ake}}(\mathcal{A}) \stackrel{\mathrm{def}}{=} 2\Pr\left[\mathsf{Succ}_P^{\mathrm{ake}}(\mathcal{A})\right] - 1.$$

When necessary to distinguish between the two notions of freshness, we use ake-nfs and ake-fs in place of ake.

As in [2], we also define the notions of client-to-server authentication, server-to-client authentication, and mutual authentication. We define $\mathsf{Adv}_P^{\mathrm{c2s}}(\mathcal{A})$ to be the probability that a server oracle terminates before any $\mathsf{Corrupt}$ query without having a partner oracle. We define $\mathsf{Adv}_P^{\mathrm{s2c}}(\mathcal{A})$ to be the probability that a client oracle terminates before any $\mathsf{Corrupt}$ query without having a partner oracle. We define $\mathsf{Adv}_P^{\mathrm{ma}}(\mathcal{A})$ to be the probability that some oracle terminates before any $\mathsf{Corrupt}$ query without having a partner oracle.

The following fact is easily verified.

[8] This is the weak corruption model of [2]. Unfortunately, the strong corruption model (at least as defined in [2]) seems impossible to achieve, at least with two round protocols like PPK, but possibly also with any protocols.

Fact 3.1

$$\Pr(\mathsf{Succ}_P^{\text{ake}}(\mathcal{A})) = \Pr(\mathsf{Succ}_{P'}^{\text{ake}}(\mathcal{A})) + \epsilon \qquad \Longleftrightarrow \qquad \mathsf{Adv}_P^{\text{ake}}(\mathcal{A}) = \mathsf{Adv}_{P'}^{\text{ake}}(\mathcal{A}) + 2\epsilon.$$

4 PPK, PAK, and PAK-Z Protocols

In this section we describe the PPK, PAK, and PAK-Z protocols. The PPK protocol is shown in Figure 1. The PAK protocol is shown in Figure 2. The PAK-Z protocol is shown in Figure 3. The function ACCEPTABLE($\cdot$) must be predefined for a specific abelian group $\overline{G}$ where G_q is a subgroup of $\overline{G}$. Then ACCEPTABLE(v) returns true if and only if $v \in \overline{G}$.

The following are some remarks and comments about the protocols, including reasoning for design decisions, instantiation hints, and some discussion of security properties achieved.

1. The ACCEPTABLE(m) test ensures that all group operations are valid and result in group elements. In particular, when $\overline{G}$ is a multiplicative group, this disallows the use of $m = 0$, which would force $\sigma = 0$. We make this generalization because it can be more efficient to test for $m \in \overline{G}$ rather than $m \in G_q$. (The ACCEPTABLE(μ) in PPK provides similar properties.)

2. The hash functions have subscripts, which basically means that when they are instantiated by some particular hash function, they need to be differentiated by some agreed upon parameters. For instance, $H_i(x)$ could possibly be defined $H(ASCII(i) \,\|\, x)$, where $H(\cdot)$ is then instantiated using SHA-1 as in [3].

3. In PPK, H_1 and H_2 output values in G_q, whereas H_3 outputs κ-bit values. In PAK, H_1 outputs values in G_q, whereas H_2, H_3, and H_4 output κ-bit values. In PAK-Z, H_1 outputs values in G_q, whereas H_2, H_3, H_4, and H_5 output κ-bit values, and H_6 outputs κ'-bit values, where κ' is the length of a signature. The instantiations for hash functions have an effect on the efficiency of the protocol. See Section 5 for details.

4. As opposed to the original model of [12], in this model the participants are split into clients and servers, with servers being allowed to store a password file. This allows us to improve the efficiency of the server by having the server store not π_C, but $\langle (H_1(\pi_C))^{-1} \rangle$ (for PAK and PAK-Z) and $\langle (H_1(\pi_C))^{-1}, H_2(\pi_C) \rangle$ (for PPK).[9] This also allows us to unify (somewhat) the implementations of PPK, PAK, and PAK-Z, since π_C obviously cannot be stored by the server in the PAK-Z protocol.

5. In PPK and PAK, as opposed to the original definitions of PPK and PAK, we only include the password in the $H_1(\cdot)$ query. This simplifies the protocol without reducing the security, as shown in the new proof. In PAK-Z, however, when considering the possibility of server compromise, we include the user identity in the $H_1(\cdot)$ query and the $H_2(\cdot)$ query to prevent a single offline dictionary attack on multiple users. (Similarly,

[9] In particular, when the group G_q is a subgroup of Z_p^*, there is an implicit exponentiation by $(p-1)/q$ in the computation of $H_1(\pi_C)$ (and $H_2(\pi_C)$ in PPK) which is avoided by the server.

we could use a salt value.) This allows us to achieve a better security bound, and is a prudent thing to do in practice. In fact, in PAK-Z one may also consider using the server identity in $H_1(\cdot)$ queries. Without the server identity, an attacker who compromises one server may impersonate another server (for which the client uses the same password), without having to perform an offline dictionary attack. Note that in our model, this is not an issue, since we only worry about an attacker impersonating a client after a server compromise, not impersonating another server. However, this may be an issue in practice. To model this, we would need to change the definition of freshness.

6. Along with the Diffie-Hellman secret value σ, we include both identities, both Diffie-Hellman exchanged values (m and μ), and $(H_1(\pi_C))^{-1}$ (part of the server-stored secret $\pi_S[C]$) in the confirmation and session key generation hashes.

   The reasons for the inclusion of each value in the hashes are as follows:

   - Without the server identity, an adversary that knows that the passwords of a client C at two different servers S_1 and S_2 are the same may fool the client into starting a session with S_1, when the client believes it is starting a session with S_2. This would break the protocol according to our model. In particular, the associated instances of C and S_1 would not be partnered, so the adversary could reveal the session key at the instance of S_1, and be able to answer a Test query to the instance of C. We discuss this further in a subsequent comment.
   - Without the client identity, an adversary that knows that two clients C_1 and C_2 share the same password may fool a server into starting a session with C_1, when the server believes it is starting a session with C_2. This would break the protocol according to our model, as above. (We would like our protocols to be as secure as possible, even if client's passwords are correlated.)
   - Without m, since the server does not necessarily test if $m \in G_q$, the adversary could, for instance, change the m value sent by a client C to some $m' \notin G_q$, such that with non-negligible probability, the σ values produced by the client and server are equal.[10] However, the *sid* values (which we define to include m) would be different, so the instances would not be partnered, and this would break the protocol (as above).[11]
   - Without μ the protocol would break similar to the way the protocol breaks if m is not included.[12]

[10] For instance, if $G_q \subseteq \mathbb{Z}_p^\star$, let $m' \leftarrow -m \bmod p$.

[11] We could define *sid* differently, but then other changes to the protocol would be required. Anyway, *sid* is generally defined as a concatentation of flows, or at least the "important" public parameters of the protocol and flows.

[12] Note that we could use α and β in place of m and μ in the hash functions, similar to the EKE2 protocol in [2], but since we include the critical part of $\pi_S[C]$, this does not seem to make any difference.

- We include part of $\pi_S[C]$ simply to make the security proof tighter.[13]

7. Even though PPK is very similar to the EKE2 protocol described in [2], and the EKE2 protocol satisfies forward secrecy, it seems to be difficult to prove that PPK satisfies forward secrecy with advantage linear in the advantage of CDH (see Appendix C). We leave this as an open problem. PPK does satisfy "weak forward secrecy" as defined in [2], but we do not include a proof of that, since it does not seem to us to be a useful notion. We do show that the PAK protocol, with its explicit key confirmation, does satisfy forward secrecy. This gives some evidence as to the importance of explicit key confirmation in password protocols.

8. The PAK-Z protocol is really a generalization of the PAK-Y protocol, in which the client gives a non-interactive proof of knowledge of the secret key to a public key. In PAK-Y the public key was the shared secret. In PAK-Z, there is a shared secret as in PAK, but also a public/secret key pair, in which the secret key is encrypted with the password (in this case, by an exclusive-or with a hash of the password, although one could also imagine using a block cipher keyed with the password). The PAK-Z protocol was inspired by the SecureStore system built using PAK by Eric Grosse. In that system, PAK was used to exchange a session key, and then an encrypted key store was downloaded to the client, where this key store was encrypted using a different function of the password than the PAK shared secret. The idea then is that by proving the ability to decrypt the key store, the client can prove knowledge of the actual password.

9. In our protocols we assume each server has a unique identity, known to the client. However, in modern networks servers often have many aliases, and primary identity strings may actually change. To handle this, our protocols and the definition of security could be generalized. In the easiest case, we could simply not include server identities, in a sense, redefining everything (partnering, *sid*'s, etc.) as if all server identities were simply the empty string. When there are multiple servers then this implies that a client may be partnered to any one of them, so this case may be the most useful when there is only a single server. More generally, a server could include its primary identity string in its first protocol message to the client, and the client could verify that it is acceptable (e.g., by querying the user). Then all subsequent references to the server identity would be to this string. Even more generally, we could develop protocols and definitions of security that would allow a client to connect to "any server from a given set of servers." However, for simplicity, we stick to the basic case in this paper.

5 Instantiating the protocol over a specific G_q

Here we show how to instantiate the protocol and hash functions over any subgroup of $\mathbb{Z}_p^\star$. For information on instantiating PAK and PPK over other groups, such as elliptic curve

[13] Including $\pi_S[C]$ makes a difference even if α and β were used in place of m and μ.

Client C		Server S
Input: S, π		$\pi_S[C] = \langle (H_1(\pi_C))^{-1}, H_2(\pi_C) \rangle$
$x \xleftarrow{R} \mathbb{Z}_q$		
$\alpha \leftarrow g^x$		
$\gamma_1 \leftarrow H_1(\pi)$		
$\gamma_2 \leftarrow H_2(\pi)$		
$m \leftarrow \alpha \cdot \gamma_1$		
	$\xrightarrow{\langle C, m \rangle}$	
		Abort if $\neg$ACCEPTABLE(m)
		$y \xleftarrow{R} \mathbb{Z}_q$
		$\beta \leftarrow g^y$
		$\langle \gamma_1', \gamma_2 \rangle \leftarrow \pi_S[C]$
		$\alpha \leftarrow m \cdot \gamma_1'$
		$\sigma \leftarrow \alpha^y$
		$\mu \leftarrow \beta \cdot \gamma_2$
	$\xleftarrow{\mu}$	
Abort if $\neg$ACCEPTABLE(μ)		
$\beta \leftarrow \mu \cdot (\gamma_2)^{-1}$		
$\gamma_1' \leftarrow (\gamma_1)^{-1}$		
$\sigma \leftarrow \beta^x$		

Figure 1: PPK Protocol. Session ID is $sid = C \parallel S \parallel m \parallel \mu$. Partner ID for C is $pid_C = S$, and partner ID for S is $pid_S = C$. Shared session key is $sk = H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle)$.

Client C		Server S
Input: S, π		$\pi_S[C] = \langle (H_1(\pi_C))^{-1} \rangle$
$x \stackrel{R}{\leftarrow} \mathbb{Z}_q$		
$\alpha \leftarrow g^x$		
$\gamma \leftarrow H_1(\pi)$		
$m \leftarrow \alpha \cdot \gamma$		
	$\xrightarrow{\langle C, m \rangle}$	
		Abort if $\neg$ACCEPTABLE(m)
		$y \stackrel{R}{\leftarrow} \mathbb{Z}_q$
		$\mu \leftarrow g^y$
		$\gamma' \leftarrow \pi_S[C]$
		$\alpha \leftarrow m \cdot \gamma'$
		$\sigma \leftarrow \alpha^y$
		$k \leftarrow H_2(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$
		$k'' \leftarrow H_3(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$
	$\xleftarrow{\langle \mu, k \rangle}$	
$\sigma \leftarrow \mu^x$		
$\gamma' \leftarrow (\gamma)^{-1}$		
Abort if $k \neq H_2(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$		
$k' \leftarrow H_3(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$		
	$\xrightarrow{k'}$	
		Abort if $k' \neq k''$

Figure 2: PAK Protocol. Session ID is $sid = C \parallel S \parallel m \parallel \mu$. Partner ID for C is $pid_C = S$, and partner ID for S is $pid_S = C$. Shared session key is $sk = H_4(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$.

Client C		Server S
Input: S, π		$\pi_S[C] =$
		$\langle (H_1(\langle C, \pi_C \rangle))^{-1}, W, H_2(\langle C, \pi_C \rangle) \oplus V \rangle$
$x \stackrel{R}{\leftarrow} \mathbb{Z}_q$		
$\alpha \leftarrow g^x$		
$\gamma \leftarrow H_1(\langle C, \pi \rangle)$		
$m \leftarrow \alpha \cdot \gamma$		
	$\xrightarrow{\langle C,m \rangle}$	
		Abort if $\neg$ACCEPTABLE(m)
		$y \stackrel{R}{\leftarrow} \mathbb{Z}_q$
		$\mu \leftarrow g^y$
		$\langle \gamma', W, V' \rangle \leftarrow \pi_S[C]$
		$\alpha \leftarrow m \cdot \gamma'$
		$\sigma \leftarrow \alpha^y$
		$a' \leftarrow H_5(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$
		$a \leftarrow a' \oplus V'$
		$s'' \leftarrow H_6(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$
		$k \leftarrow H_3(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$
	$\xleftarrow{\langle \mu,k,a \rangle}$	
$\sigma \leftarrow \mu^x$		
$\gamma' \leftarrow \gamma^{-1}$		
Abort if $k \neq H_3(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$		
$a' \leftarrow H_5(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$		
$V' \leftarrow a' \oplus a$		
$V \leftarrow H_2(\langle C, \pi \rangle) \oplus V'$		
Abort if $\neg$VALID(V)		
$s \leftarrow \mathsf{sig}_V(\mu)$		
$s'' \leftarrow H_6(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$		
$s' \leftarrow s'' \oplus s$		
	$\xrightarrow{s'}$	
		$s \leftarrow s'' \oplus s'$
		Abort if $\mathsf{Verify}_W(\mu, s) = 0$

Figure 3: PAK-Z Protocol. Session ID is $sid = C \parallel S \parallel m \parallel \mu$. Partner ID for C is $pid_C = S$, and partner ID for S is $pid_S = C$. Shared session key is $sk = H_4(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$.

groups and XTR groups [30], see [33].

Let G_q be a q-order subgroup of $\mathbb{Z}_p^\star$, where p is prime.

The test ACCEPTABLE(x) returns true if and only if $x \not\equiv 0 \bmod p$, i.e., if $x \in \mathbb{Z}_p^\star$.

To define hash functions that output elements of G_q, (H_1 in PAK and PAK-Z, and H_1 and H_2 in PPK), we could, for instance, use a random oracle $H : \{0,1\}^* \to \{0,1\}^{|p|+\kappa}$, and set $H_i(x) \leftarrow (H(ASCII(i) \parallel x))^{(p-1)/q}$. To simulate H_i, we can simulate H in the following way. So that $H_i(x)$ outputs X (with high probability), we set $H(ASCII(i) \parallel x)$ as follows, with $w = 2^{|p|+\kappa}$: Generate $r \xleftarrow{R} \mathbb{Z}_w$. If $r > w - (w \bmod p)$ output r, else output $(X^{q/(p-1)} r^q \bmod p) + \lfloor r/p \rfloor p$.

An alternative formulation (similar to one in the IEEE P1363.2 document) would be to set $H_i(x) \leftarrow g \cdot (g')^{H(ASCII(i) \parallel x) \bmod q} \bmod p$. where $g' \leftarrow (H(ASCII(i)))^{(p-1)/q} \bmod p$. This works if for all inputs x, we want $H_i(x)$ to output gX^c, for some $X \in G_q$, $c \in \mathbb{Z}_q$, with c random. We first set $H(ASCII(i))$ as follows, with $w = 2^{|p|+\kappa}$: Generate $r \xleftarrow{R} \mathbb{Z}_w$ and $r' \xleftarrow{R} \mathbb{Z}_q$. If $r > w - (w \bmod p)$ output r, else output $(X^{q/(p-1)} r^q \bmod p) + \lfloor r/p \rfloor p$. Then so that $H_i(x)$ outputs gX^c, we set $H(ASCII(i) \parallel x)$ to c.

6 Security of the Protocols

Here we state the CDH assumption, and define security for signature schemes. Following that we prove that the protocol P is secure, based on the CDH assumption.

Computational Diffie-Hellman Here we formally state the CDH assumption. Let G_q be as in Section 2, with generator g. For two values X and Y, if ACCEPTABLE(X) and $Y = g^y$, let $\mathsf{DH}(X,Y) = X^y$, else if $X = g^x$ and ACCEPTABLE(Y), let $\mathsf{DH}(X,Y) = Y^x$. (Note that if $X = g^x$ and $Y = g^y$, then by this definition $\mathsf{DH}(X,Y) = g^{xy}$.) Let $\mathcal{A}$ be an algorithm with input (X,Y). Let

$$\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(\mathcal{A}) \stackrel{\text{def}}{=} \Pr\left[(x,y) \xleftarrow{R} \mathbb{Z}_q;\, X \leftarrow g^x;\, Y \leftarrow g^y :\ \mathsf{DH}(X,Y) \in \mathcal{A}(X,Y)\right]$$

Let $\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t,n) = \max_{\mathcal{A}}\left\{\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(\mathcal{A})\right\}$, where the maximum is taken over all adversaries of time complexity at most t that output a list containing at most n elements of G_q. The CDH assumption states that for t and n polynomial in κ, $\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t,n)$ is negligible.

Security for signature schemes. We specify existential unforgeability versus chosen message attacks [21] for a signature scheme $\mathcal{S} = (\mathsf{Gen}_{\mathcal{S}}, \mathsf{Sig}, \mathsf{Verify})$. A forger $\mathcal{F}$ is given pk, where $(pk, sk) \leftarrow \mathsf{Gen}_{\mathcal{S}}(1^\kappa)$, and tries to forge signatures with respect to pk. It is allowed to query a signature oracle (with respect to sk) on messages of its choice. It succeeds if after this it can output a valid forgery (m, σ) such that $\mathsf{Verify}_{pk}(m,\sigma) = 1$, where m was not one of the messages signed by the signature oracle. We say $\mathsf{Succ}_{\mathcal{S},\kappa}^{\text{eu-cma}}(\mathcal{F}) = \Pr(\mathcal{F}\text{ succeeds})$, and $\mathsf{Succ}_{\mathcal{S},\kappa}^{\text{eu-cma}}(t,u) = \max_{\mathcal{F}}\left\{\mathsf{Succ}_{\mathcal{S},\kappa}^{\text{eu-cma}}(\mathcal{F})\right\}$, where the maximum is taken over all forgers of time complexity t that make u queries to the signature oracle. A signature scheme $\mathcal{S}$ is existentially unforgeable versus chosen message attacks if for t and u polynomial in κ, $\mathsf{Succ}_{\mathcal{S},\kappa}^{\text{eu-cma}}(t,u)$ is negligible.

Practical examples of signature schemes based on the hardness of the discrete logarithm problem that are existentially unforgeable versus chosen message attacks can be found in [35, 37].

6.1 PPK Protocol

Here we prove that the PPK protocol is secure, in the sense that an adversary attacking the system cannot determine session keys of fresh instances with greater advantage than that of an online dictionary attack.

Theorem 6.1 *Let P be the protocol described in Figure 1 (and formally described in Appendix B), using group G_q, and with a password dictionary of size N. Fix an adversary $\mathcal{A}$ that runs in time t, and makes $n_{\mathrm{se}}, n_{\mathrm{ex}}, n_{\mathrm{re}}$ queries of type* Send, Execute, Reveal*, respectively, and n_{ro} queries to the random oracles. Then for $t' = O(t + ((n_{\mathrm{ro}})^2 + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$:*

$$\mathsf{Adv}_P^{\text{ake-nfs}}(\mathcal{A}) \quad = \quad \frac{n_{\mathrm{se}}}{N} + O\left(\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', (n_{\mathrm{ro}})^2) + \frac{(n_{\mathrm{se}} + n_{\mathrm{ex}})(n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})}{q}\right).$$

Proof: Our proof will proceed by introducing a series of protocols $P_0, P_1, \ldots, P_7$ related to P, with $P_0 = P$. In P_7, $\mathcal{A}$ will be reduced to a simple online guessing attack that will admit a straightforward analysis. We describe these protocols informally in Figure 4.

For each i from 1 to 7, we will prove that the advantage of $\mathcal{A}$ attacking protocol P_{i-1} is at most negligibly more than the advantage of $\mathcal{A}$ attacking protocol P_i. We sketch these proofs in Figure 5.

Now we proceed to the detailed proof.

We use the terminology "in a Client Action i query to Π_i^C" to mean "in a Send query to Π_i^C that results in the Client Action i procedure being executed," and "in a Server Action i query to Π_j^S" to mean "in a Send query to Π_j^S that results in the Server Action i procedure being executed,"

We assume without loss of generality that n_{ro} and $n_{\mathrm{se}} + n_{\mathrm{ex}}$ are both at least 1. We make the standard assumption that random oracles are built "on the fly," that is, each new query to a random oracle is answered with a fresh random output, and each query that is not new is answered consistently with the previous queries. We also assume that the $H_1(\cdot)$ and $H_2(\cdot)$ queries are answered in the following way:

1. In an $H_1(\pi)$ query, output $g^{\psi_1[\pi]}$, where $\psi_1[\pi] \stackrel{R}{\leftarrow} \mathbb{Z}_q$.[14]

2. In an $H_2(\pi)$ query, output $g^{\psi_2[\pi]}$, where $\psi_2[\pi] \stackrel{R}{\leftarrow} \mathbb{Z}_q$.

[14] Note that we do this by fixing the underlying $H(\cdot)$ query. See Section 5.

P_0 The original protocol P.

P_1 If honest parties randomly choose m or μ values seen previously in the execution of the protocol, the protocol halts and the adversary fails.

P_2 The protocol answers Send and Execute queries without making any random oracle queries. Subsequent random oracle queries by the adversary are backpatched, as much as possible, to be consistent with the responses to the Send and Execute queries. (This is a standard technique for proofs involving random oracles.)

P_3 If an $H_3(\cdot)$ query is made, it is not checked for consistency against Execute queries. That is, instead of backpatching to maintain consistency with an Execute query, the protocol responds with a random output.

P_4 If a correct password guess is made against a client instance or server instance (determined by an $H_3(\cdot)$ query using the correct inputs to compute a session key), the protocol halts and the adversary automatically succeeds.

P_5 If the adversary makes two password guesses against the same server instance, the protocol halts and the adversary fails.

P_6 If the adversary makes two password guesses against the same client instance, the protocol halts and the adversary fails.

P_7 The protocol uses an internal password oracle that holds all passwords and only accepts simple queries that test whether a given password is the correct password for a given client/server pair. The test for correct password guesses (from P_4) is changed so that whenever the adversary makes a password guess, a query is submitted to the oracle to determine if it is correct.

Figure 4: Informal description of protocols P_0 through P_7

$P_0 \to P_1$ This is straightforward.

$P_1 \to P_2$ By inspection, the two protocols are indistinguishable unless the adversary makes an $H_3(\langle C, S, \cdot, \cdot, \cdot, \gamma' \rangle)$ query where $\gamma' = H_1(\pi_C)$, but the adversary has not actually made the $H_1(\pi_C)$ query. However, the probability of this is negligible.

$P_2 \to P_3$ This can be shown using a standard reduction from CDH. On input (X, Y), we plug in X multiplied by random powers of g for the clients' m values, and Y multiplied by random powers of g for the servers' μ values. Then we plug in random powers of g for the outputs of $H_1(\cdot)$ and $H_2(\cdot)$ queries, so that from a correct $H_3(\cdot)$ query, we can compute $\mathsf{DH}(X, Y)$.

$P_3 \to P_4$ This is obvious.

$P_4 \to P_5$ This can be shown using a reduction from CDH. On input (X, Y), we randomly plug in X or 1 multiplied by random powers of g into the outputs of $H_1(\cdot)$ queries, and Y multiplied by random powers of g for the servers' μ values. Then we plug in random powers of g for the outputs of $H_2(\cdot)$ queries. Finally, for pairs of $H_3(\cdot)$ queries corresponding to password guesses using particular m and μ values, we can divide out the common $\mathsf{DH}(m, Y)$ value and compute $\mathsf{DH}(X, Y)$ (with probability $\frac{1}{2}$).

$P_5 \to P_6$ This is a reduction from CDH, similar to the previous one. On input (X, Y), we randomly plug in Y or 1 multiplied by random powers of g into the outputs of $H_2(\cdot)$ queries, and X multiplied by random powers of g for the clients' m values. Then we plug in random powers of g for the outputs of $H_1(\cdot)$ queries. Finally, for pairs of $H_3(\cdot)$ queries corresponding to password guesses using particular m and μ values, we can divide out the common $\mathsf{DH}(X, \mu)$ value and compute $\mathsf{DH}(X, Y)$ (with probability $\frac{1}{2}$).

$P_6 \to P_7$ By inspection, these two protocols are indistinguishable.

Figure 5: Proof sketches of negligible advantage gain from P_{i-1} to P_i

That is, we assume we know the discrete logs of the outputs of $H_1(\cdot)$ and $H_2(\cdot)$ queries. Finally, we assume that for each $H_1(\pi)$ query, the corresponding $H_2(\pi)$ query is made automatically, and vice-versa. Both queries are considered to be made by $\mathcal{A}$, though $\mathcal{A}$ only sees the output of the original one.

We now define some events, corresponding to the adversary making a password guess against a client instance, against a server instance, and against a client instance and server instance that are partnered in an Execute query. In each case, we also define an associated value for the event, and we note that the associated value is actually fixed by the protocol before the event occurs.

- testpw(C, i, S, π): for some m, μ, γ_1', and γ_2, $\mathcal{A}$ makes an $H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle)$ query, a Client Action 0 query to a client instance Π_i^C with input S and output $\langle C, m \rangle$, a Client Action 1 query to Π_i^C with input μ, an $H_1(\pi)$ query returning $(\gamma_1')^{-1}$, and an $H_2(\pi)$ query returning γ_2, where the latest query is either the $H_3(\cdot)$ query or the Client Action 1 query, $\sigma = DH(\alpha, \beta)$, $m = \alpha \cdot (\gamma_1')^{-1}$, $\mu = \beta \cdot \gamma_2$, and $\text{ACCEPTABLE}(\mu)$. The associated value of this event is $sk_C^i = H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle)$.

- testpw(S, j, C, π): for some m, μ, γ_1', and γ_2, $\mathcal{A}$ makes an $H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle)$ query, and previously $\mathcal{A}$ made a Server Action 1 query to a server instance Π_j^S with input $\langle C, m \rangle$ and output μ, an $H_1(\pi)$ query returning $(\gamma_1')^{-1}$, and an $H_2(\pi)$ query returning γ_2, where $\sigma = DH(\alpha, \beta)$, $m = \alpha \cdot (\gamma_1')^{-1}$, $\mu = \beta \cdot \gamma_2$, and $\text{ACCEPTABLE}(m)$. The associated value of this event is $sk_S^j = H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle)$.

- testexecpw(C, i, S, j, π): for some m, μ, γ_1', and γ_2, $\mathcal{A}$ makes an $H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle)$ query, and previously $\mathcal{A}$ made an Execute $(C,\ i,\ S,\ j)$ query that generates m and μ, an $H_1(\pi)$ query returning $(\gamma_1')^{-1}$, and an $H_2(\pi)$ query returning γ_2, where $\sigma = DH(\alpha, \beta)$, $m = \alpha \cdot (\gamma_1')^{-1}$, and $\mu = \beta \cdot \gamma_2$. The associated value of this event is $sk_C^i = sk_S^j = H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle)$.

- correctpw: *either* a testpw(C, i, S, π_C) event occurs for some C, i, and S, *or* a testpw(S, j, C, π_C) event occurs for some S, j, and C.

- correctpwexec: a testexecpw(C, i, S, j, π_C) event occurs for some C, i, S, and j.

- doublepwserver: both a testpw(S, j, C, π) event and a testpw(S, j, C, π') event occur, for some S, j, C, π and π', with $\pi \neq \pi'$.

- doublepwclient: both a testpw(C, i, S, π) event and a testpw$(C, i, S, , \pi')$ event occur, for some C, i, S, π and π', with $\pi \neq \pi'$.

Say a client instance Π_i^C is *paired with* a server instance Π_j^S if there is a Client Action 0 query to Π_i^C with input S and output $\langle C, m \rangle$, there is a Server Action 1 query to Π_j^S with input $\langle C, m \rangle$ and output μ, and there is a Client Action 1 query to Π_i^C with input μ. Say a server instance Π_j^S is *paired with* a client instance Π_i^C if there is a Client Action

0 query to Π_i^C with input S and output $\langle C, m\rangle$, there is a Server Action 1 query to Π_j^S with input $\langle C, m\rangle$.

Protocol P_1. Let E_1 be the event that an m value generated in a Client Action 0 or Execute query is equal to an m value generated in a previous Client Action 0 or Execute query, an m value sent as input in a previous Server Action 1 query, or an m value in a previous $H_3(\cdot)$ query (made by the adversary). Let E_2 be the event that a μ value generated in a Server Action 1 or Execute query is equal to a μ value generated in a previous Server Action 1 or Execute query, a μ value sent as input in a previous Client Action 1 query, or a μ value in a previous $H_3(\cdot)$ query (made by the adversary). Let $E = E_1 \vee E_2$. Let P_1 be a protocol that is identical to P_0 except that if E occurs, the protocol aborts (and thus the adversary fails).

Claim 6.2 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}_{P_0}^{\mathrm{ake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_1}^{\mathrm{ake}}(\mathcal{A}) + \frac{O((n_{\mathrm{se}} + n_{\mathrm{ex}})(n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}}))}{q}.$$

Proof: Straightforward. ▌

Protocol P_2. Let P_2 be a protocol that is identical to P_1 except that Send and Execute queries are answered without making any random oracle queries, and subsequent random oracle queries by the adversary are backpatched, as much as possible, to be consistent with the responses to the Send and Execute queries.

Specifically, the queries in P_2 are changed as follows:

- In an Execute $(C,\ i,\ S,\ j)$ query, $m \leftarrow g^{\tau[i,C]}$, where $\tau[i,C] \xleftarrow{R} \mathbb{Z}_q$, $\mu \leftarrow g^{\tau[j,S]}$, where $\tau[j,S] \xleftarrow{R} \mathbb{Z}_q$, and $sk_C^i \leftarrow sk_S^j \xleftarrow{R} \{0,1\}^\kappa$.
- In a Client Action 0 query to instance Π_i^C, $m \leftarrow g^{\tau[i,C]}$, where $\tau[i,C] \xleftarrow{R} \mathbb{Z}_q$.
- In a Server Action 1 query to instance Π_j^S, $\mu \leftarrow g^{\tau[j,S]}$, where $\tau[j,S] \xleftarrow{R} \mathbb{Z}_q$, and $sk_S^j \xleftarrow{R} \{0,1\}^\kappa$.
- In a Client Action 1 query to instance Π_i^C, if Π_i^C is paired with an instance Π_j^S, $sk_C^i \leftarrow sk_S^j$, else if this query causes a $\mathsf{testpw}(C,i,S,\pi_C)$ event to occur, set sk_C^i to the associated value of that event, else $sk_C^i \xleftarrow{R} \{0,1\}^\kappa$.
- In an $H_3(\langle C, S, m, \mu, \sigma, \gamma_1'\rangle)$ query, if this $H_3(\cdot)$ query causes a $\mathsf{testpw}(C,i,S,\pi_C)$, $\mathsf{testpw}(S,j,C,\pi_C)$, or $\mathsf{testexecpw}(C,i,S,j,\pi_C)$ event to occur, output the associated value of that event, else output a random value from $\{0,1\}^\kappa$.

Note that we can determine whether the appropriate events occur using the ψ_1, ψ_2, and τ values. Also note that by P_1 and the fact that a client instance that is paired with a server instance copies the session key of that server instance, there will never be more than one associated value that needs to be considered in the $H_3(\cdot)$ query.

Claim 6.3 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}_{P_1}^{\mathrm{ake}}(\mathcal{A}) = \mathsf{Adv}_{P_2}^{\mathrm{ake}}(\mathcal{A}) + \frac{O(n_{\mathrm{ro}})}{q}.$$

Proof: One can see that in P_1, a server instance Π_j^S, creates a session key sk_S^j that is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since the $H_3(\cdot)$ query that determines sk_S^j is new. Also in P_1, for any client instance Π_i^C, either

1. exactly one instance Π_j^S is paired with Π_i^C, in which case $sk_C^i = sk_S^j$, or
2. no instance is paired with Π_i^C, in which case either a $\mathsf{testpw}(C, i, S, \pi_C)$ event occurs, and sk_C^i is the value associated with that event (i.e., the output of the previous $H_3(\cdot)$ query associated with that event), or sk_C^i is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since the $H_3(\cdot)$ query that determines sk_C^i is new.

Finally, for any $H_3(\langle C, S, \cdot, \cdot, \cdot, (\gamma_1')\rangle)$ query, either (1) it causes a $\mathsf{testpw}(C, i, S, \pi_C)$, $\mathsf{testpw}(S, j, C, \pi_C)$, or $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event to occur, in which case the output is the associated value of that event (i.e., the session key associated with the particular event that occurs), (2) $\gamma_1' = H_1(\pi_C)$, but the adversary has not made an $H_1(\pi_C)$ query, or (3) the output of $H_3(\cdot)$ is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since this is a new $H_3(\cdot)$ query.

The total probability of an $H_3(\cdot)$ query causing the second case above can be easily shown to be bounded by $\frac{n_{\mathrm{ro}}}{q}$. If the second case never occurs, then P_2 is consistent with P_1. The claim follows. ▮

Protocol P_3. Let P_3 be a protocol that is identical to P_2 except that in an $H_3(\langle C, S, \cdot, \cdot, \cdot, \cdot\rangle)$ query there is no check for a $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event.

Claim 6.4 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + (n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$ such that

$$\mathsf{Adv}_{P_2}^{\mathrm{ake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_3}^{\mathrm{ake}}(\mathcal{A}) + 2\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', n_{\mathrm{ro}}).$$

Proof: Let E be the event that a $\mathsf{correctpwexec}$ event occurs. Obviously, if E does not occur, then P_2 and P_3 are indistinguishable. Let ϵ be the probability that E occurs when

$\mathcal{A}$ is running against protocol P_1. Then $\Pr(\mathsf{Succ}^{\mathrm{ake}}_{P_2}(\mathcal{A})) \le \Pr(\mathsf{Succ}^{\mathrm{ake}}_{P_3}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}^{\mathrm{ake}}_{P_2}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{ake}}_{P_3}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts to solve CDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y), D simulates P_3 for $\mathcal{A}$ with these changes:

1. In an $\mathsf{Execute}\,(C,\ i,\ S,\ j)$ query, set $m \leftarrow X g^{\rho_{i,C}}$ and $\mu \leftarrow Y g^{\rho'_{j,S}}$, where $\rho_{i,C}, \rho'_{j,S} \overset{R}{\leftarrow} \mathbb{Z}_q$.
2. When $\mathcal{A}$ finishes, for every $H_3(\langle C, S, m, \mu, \sigma, \gamma'_1\rangle)$ query, where m and μ were generated in an $\mathsf{Execute}\,(C,\ i,\ S,\ j)$ query and an $H_1(\pi_C)$ query returned $(\gamma'_1)^{-1}$, add
$$\sigma X^{-\rho'_{j,S}} Y^{-\rho_{i,C}} g^{-\rho_{i,C}\rho'_{j,S}} m^{\psi_2[\pi_C]} \mu^{\psi_1[\pi_C]} g^{-\psi_1[\pi_C]\psi_2[\pi_C]}$$
to the list of possible values for $\mathsf{DH}(X, Y)$.

This simulation is perfectly indistinguishable from P_2 until E occurs, and in this case, D adds the correct $\mathsf{DH}(X, Y)$ to the list. After E occurs the simulation may be distinguishable from P_2, but this does not change the fact that E occurs with probability ϵ. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulator can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguishes the simulation from P_2.

D creates a list of size n_{ro}, and its advantage is ϵ. Let t' be the running time of D, and note that $t' = O(t + (n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$. The claim follows from the fact that $\mathsf{Adv}^{\mathrm{CDH}}_{G_q}(D) \le \mathsf{Adv}^{\mathrm{CDH}}_{G_q}(t', n_{\mathrm{ro}})$. ▮

Protocol P_4. Let P_4 be a protocol that is identical to P_3 except that if correctpw occurs then the protocol halts and the adversary automatically succeeds. Note that the protocol already checks for a correctpw event, specifically, in the Client Action 1 query to determine if the session key has already been determined, and in the $H_3(\cdot)$ query, to see if the output has already been determined.

Claim 6.5 For any adversary $\mathcal{A}$,
$$\mathsf{Adv}^{\mathrm{ake}}_{P_3}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{ake}}_{P_4}(\mathcal{A}).$$

Proof: Obvious. ▮

Note that in P_4, until correctpw occurs, an $H_3(\langle C, S, \cdot, \cdot, \cdot, \cdot\rangle)$ query will output a value uniformly chosen from $\{0,1\}^\kappa$, and the session key for an unpaired client instance will be uniformly chosen from $\{0,1\}^\kappa$.

Protocol P_5. Let P_5 be a protocol that is identical to P_4 except that if doublepwserver occurs, the protocol halts and the adversary fails. We assume that when a query is made, the test for doublepwserver occurs before the test for correctpw.

Claim 6.6 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t+((n_{\rm ro})^2+n_{\rm se}+n_{\rm ex})t_{\sf exp})$ such that

$$\mathsf{Adv}^{\rm ake}_{P_4}(\mathcal{A}) \leq \mathsf{Adv}^{\rm ake}_{P_5}(\mathcal{A}) + 4\mathsf{Adv}^{\rm CDH}_{G_q}(t', (n_{\rm ro})^2).$$

Proof: Let ϵ be the probability that $\mathsf{doublepwserver}$ occurs when $\mathcal{A}$ is running against protocol P_4. Then $\Pr(\mathsf{Succ}^{\rm ake}_{P_4}(\mathcal{A})) \leq \Pr(\mathsf{Succ}^{\rm ake}_{P_5}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}^{\rm ake}_{P_4}(\mathcal{A}) \leq \mathsf{Adv}^{\rm ake}_{P_5}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts to solve CDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y), D simulates P_4 for $\mathcal{A}$ with these changes:

1. In an $H_1(\pi)$ query, output $X^{\psi_1[\pi]}g^{\psi_1'[\pi]}$, where $\psi_1[\pi] \stackrel{R}{\leftarrow} \{0,1\}$ and $\psi_1'[\pi] \stackrel{R}{\leftarrow} \mathbb{Z}_q$.
2. In a Server Action 1 query to a server instance Π_j^S with input $\langle C, m\rangle$ where $\textsc{acceptable}(m)$ is true, set $\mu \leftarrow Yg^{\rho'_{j,S}}$.
3. Tests for $\mathsf{correctpw}$ (from P_4) are not made.
4. When $\mathcal{A}$ finishes, for every pair of queries $H_3(\langle C, S, m, \mu, \sigma, \gamma_1'\rangle)$ and $H_3(\langle C, S, m, \mu, \hat\sigma, \hat\gamma_1'\rangle)$ where $\textsc{acceptable}(\sigma)$ and $\textsc{acceptable}(\hat\sigma)$ are true, and there was a Server Action 1 query to a server instance Π_j^S with input $\langle C, m\rangle$ and output μ (and thus $\textsc{acceptable}(m)$ is true), an $H_1(\pi)$ query that returned $(\gamma_1')^{-1}$, an $H_1(\hat\pi)$ query that returned $(\hat\gamma_1')^{-1}$, and $\psi_1[\pi] \neq \psi_1[\hat\pi]$, add
$$\left(\sigma\hat\sigma^{-1}m^{\psi_2[\pi]-\psi_2[\hat\pi]}(\gamma_1')^{\rho'_{j,S}-\psi_2[\pi]}(\hat\gamma_1')^{-\rho'_{j,S}+\psi_2[\pi]}Y^{\psi_1'[\pi]-\psi_1'[\hat\pi]}\right)^{\psi_1[\hat\pi]-\psi_1[\pi]}$$
to the list of possible values for $\mathsf{DH}(X, Y)$.

This simulation is perfectly indistinguishable from P_4 until a $\mathsf{doublepwserver}$ event or a $\mathsf{correctpw}$ event occurs. If a $\mathsf{doublepwserver}$ event occurs, then with probability $\frac{1}{2}$ it occurs for two passwords π and $\hat\pi$ with $\psi_1[\pi] \neq \psi_1[\hat\pi]$, and in this case D adds the correct $\mathsf{DH}(X, Y)$ to the list.[15] If a $\mathsf{correctpw}$ event occurs before a $\mathsf{doublepwserver}$ event occurs, then the $\mathsf{doublepwserver}$ event would never have occurred in P_4, since P_4 would halt. Note that in this case, the simulation may be distinguishable from P_4, but this does not change the fact that a $\mathsf{doublepwserver}$ event will occur with probability at least ϵ in the simulation. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulator can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguishes the simulation from P_4.

[15] The correctness of this calculation relies on the fact that $\textsc{acceptable}(m)$ is true, and that $\overline{G}$ is an abelian group.

D creates a list of size $(n_{\mathrm{ro}})^2$, and its advantage is $\frac{\epsilon}{2}$. Let t' be the running time of D, and note that $t' = O(t + ((n_{\mathrm{ro}})^2 + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$. The claim follows from the fact that $\mathsf{Adv}^{\mathrm{CDH}}_{G_q}(D) \leq \mathsf{Adv}^{\mathrm{CDH}}_{G_q}(t', (n_{\mathrm{ro}})^2)$. ∎

Protocol P_6. Let P_6 be a protocol that is identical to P_5 except that if $\mathsf{doublepwclient}$ occurs, the protocol halts and the adversary fails. We assume that when a query is made, the test for $\mathsf{doublepwclient}$ occurs before the tests for $\mathsf{doublepwserver}$ and $\mathsf{correctpw}$.

Claim 6.7 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t+((n_{\mathrm{ro}})^2+n_{\mathrm{se}}+n_{\mathrm{ex}})t_{\mathsf{exp}})$, such that

$$\mathsf{Adv}^{\mathrm{ake}}_{P_5}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{ake}}_{P_6}(\mathcal{A}) + 4\mathsf{Adv}^{\mathrm{CDH}}_{G_q}(t', (n_{\mathrm{ro}})^2).$$

Proof: Let ϵ be the probability that $\mathsf{doublepwclient}$ occurs when $\mathcal{A}$ is running against protocol P_5. Then $\Pr(\mathsf{Succ}^{\mathrm{ake}}_{P_5}(\mathcal{A})) \leq \Pr(\mathsf{Succ}^{\mathrm{ake}}_{P_6}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}^{\mathrm{ake}}_{P_5}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{ake}}_{P_6}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts to solve CDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y), D simulates P_5 for $\mathcal{A}$ with these changes:

1. In a $H_2(\pi)$ query, output $Y^{\psi_2[\pi]}g^{\psi_2'[\pi]}$, where $\psi_2[\pi] \xleftarrow{R} \{0,1\}$ and $\psi_2'[\pi] \xleftarrow{R} \mathbb{Z}_q$.
2. In a Client Action 0 query, say to a client instance Π_i^C, with input S, set $m \leftarrow Xg^{\rho_{i,C}}$.
3. Tests for $\mathsf{correctpw}$ (from P_4), and $\mathsf{doublepwserver}$ (from P_5) are not made.
4. When $\mathcal{A}$ finishes, for every pair of queries $H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle)$ and $H_3(\langle C, S, m, \mu, \hat{\sigma}, \hat{\gamma}_1' \rangle)$ where $\textsc{acceptable}(\mu)$, $\textsc{acceptable}(\sigma)$ and $\textsc{acceptable}(\hat{\sigma})$ are true, there was a Client Action 0 query to a client instance Π_i^C with input S and output $\langle C, m \rangle$, a Client Action 1 query to Π_i^C with input μ, an $H_1(\pi)$ query that returned $(\gamma_1')^{-1}$, an $H_1(\hat{\pi})$ query that returned $(\hat{\gamma}_1')^{-1}$, and $\psi_2[\pi] \neq \psi_2[\hat{\pi}]$, let $\gamma_2 = H_2(\pi)$ and $\hat{\gamma}_2 = H_2(\hat{\pi})$ and add

$$\left(\sigma\hat{\sigma}^{-1}\mu^{\psi_1[\pi]-\psi_1[\hat{\pi}]}(\gamma_2)^{-\rho_{i,C}+\psi_1[\pi]}(\hat{\gamma}_2)^{\rho_{i,C}-\psi_1[\pi]}X^{\psi_2'[\pi]-\psi_2'[\hat{\pi}]}\right)^{\psi_2[\hat{\pi}]-\psi_2[\pi]}$$

   to the list of possible values for $\mathsf{DH}(X, Y)$.

This simulation is perfectly indistinguishable from P_5 until a $\mathsf{doublepwclient}$ event, a $\mathsf{doublepwserver}$ event, or a $\mathsf{correctpw}$ event occurs. If a $\mathsf{doublepwclient}$ event occurs, then with probability $\frac{1}{2}$ it occurs for two passwords π and $\hat{\pi}$ with $\psi_2[\pi] \neq \psi_2[\hat{\pi}]$, and in this case D adds the correct $\mathsf{DH}(X, Y)$ to the list. If a $\mathsf{doublepwserver}$ event or a $\mathsf{correctpw}$ event occurs before a $\mathsf{doublepwclient}$ event occurs, then the $\mathsf{doublepwclient}$ event would never have occurred in P_5, since P_5 would halt. Note that in this case, the simulation may be distinguishable from

P_5, but this does not change the fact that a $\mathsf{doublepwclient}$ event will occur with probability at least ϵ in the simulation. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulator can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguishes the simulation from P_5.

D creates a list of size $(n_{\mathrm{ro}})^2$, and its advantage is $\frac{\epsilon}{2}$. Let t' be the running time of D, and note that $t' = O(t + ((n_{\mathrm{ro}})^2 + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$. The claim follows from the fact that $\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(D) \leq \mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', (n_{\mathrm{ro}})^2)$. ▮

In P_6, it is easy to see that the total number of $\mathsf{testpw}(C, i, S, \pi)$ and $\mathsf{testpw}(S, j, C, \pi)$ events is at most n_{se}, and more precisely, it is at most the number of client and server instances which received a Send query.

Protocol P_7. Let P_7 be a protocol that is identical to P_6 except that there is a new internal oracle (i.e., not available to the adversary) that handles passwords, called a *password oracle*. This oracle generates all passwords during initialization. Then it accepts queries of the form $\mathsf{testpw}(C, \pi)$ and returns TRUE if $\pi = \pi_C$, and FALSE otherwise. The protocol is changed only in the method for determining $\mathsf{correctpw}$. Specifically, to test if $\mathsf{correctpw}$ occurs, whenever a $\mathsf{testpw}(C, i, S, \pi)$ event or $\mathsf{testpw}(S, j, C, \pi)$ event occurs, a $\mathsf{testpw}(C, \pi)$ query is made to the password oracle to see if $\pi = \pi_C$.

Claim 6.8 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}_{P_6}^{\mathrm{ake}}(\mathcal{A}) = \mathsf{Adv}_{P_7}^{\mathrm{ake}}(\mathcal{A}).$$

Proof: By inspection, P_6 and P_7 are perfectly indistinguishable. ▮

The probability of the adversary $\mathcal{A}$ succeeding in P_7 is bounded by

$$\Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A})) \quad \leq \quad \Pr(\mathsf{correctpw}) + \Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A}) | \neg\mathsf{correctpw}) \Pr(\neg\mathsf{correctpw}).$$

First, since there are at most n_{se} queries to the password oracle, and passwords are chosen uniformly from a dictionary of size N, $\Pr(\mathsf{correctpw}) \leq \frac{n_{\mathrm{se}}}{N}$.

Now we compute $\Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A}) | \neg\mathsf{correctpw})$. If $\mathsf{correctpw}$ does not occur, then $\mathcal{A}$ succeeds by making a Test query to a fresh instance Π_i^U and guessing the bit used in that Test query. We will show that the view of the adversary is independent of sk_U^i, and thus the probability of success is exactly $\frac{1}{2}$.

First we examine Reveal queries. Recall that since Π_i^U is fresh, there could be no $\mathsf{Reveal}\ (U,\ i)$ query, and if $\Pi_j^{U'}$ is partnered with Π_i^U, no $\mathsf{Reveal}\ (U',\ j)$ query. Second note that since *sid* includes m and μ values, if more than a single client instance and a single server instance accept with the same *sid*, $\mathcal{A}$ fails (see P_1). Thus the output of Reveal queries is independent of sk_U^i.

Second we examine $H_3(\cdot)$ queries. As noted in the discussion following the description of P_4, an $H_3(\cdot)$ query returns random values independent of anything that previously occurred. Thus any $H_3(\cdot)$ queries that occur after sk_U^i is set are independent of sk_U^i. But consider the following cases. (1) If $U \in \mathit{Servers}$, sk_U^i is chosen independently of anything that previously occurred (see P_2). (2) If $U \in \mathit{Clients}$ and is unpaired, sk_U^i is chosen independently of anything that previously occurred (see the discussion after P_4). (3) If $U \in \mathit{Clients}$ and is paired, then $sk_U^i \leftarrow sk_{U'}^j$, where $\Pi_j^{U'}$ is the partner of Π_i^U and $sk_{U'}^j$ is chosen independently of anything that previously occurred (see P_2). This implies that the view of the adversary is independent of sk_U^i, and thus the probability of success is exactly $\frac{1}{2}$.

Since $\Pr(\neg\mathsf{correctpw}) = 1 - \Pr(\mathsf{correctpw})$, we have that

$$\begin{aligned}
\Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A})) &\leq \Pr(\mathsf{correctpw}) + \Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A})|\neg\mathsf{correctpw})(1 - \Pr(\mathsf{correctpw})) \\
&\leq \Pr(\mathsf{correctpw}) + \frac{1}{2}(1 - \Pr(\mathsf{correctpw})) \\
&= \frac{1}{2} + \frac{\Pr(\mathsf{correctpw})}{2} \\
&\leq \frac{1}{2} + \frac{n_{\mathrm{se}}}{2N}.
\end{aligned}$$

Therefore $\mathsf{Adv}_{P_7}^{\mathrm{ake}}(\mathcal{A}) \leq \frac{n_{\mathrm{se}}}{N}$. The theorem follows from this and Claims 6.2 through 6.8. ∎

6.2 PAK Protocol

Here we prove that the PAK protocol is secure, in the sense that an adversary attacking the system cannot determine session keys of fresh instances with greater advantage than that of an online dictionary attack.

Theorem 6.9 *Let P be the protocol described in Figure 2 (and formally described in Appendix B), using group G_q, and with a password dictionary of size N. Fix an adversary $\mathcal{A}$ that runs in time t, and makes $n_{\mathrm{se}}, n_{\mathrm{ex}}, n_{\mathrm{re}}, n_{\mathrm{co}}$ queries of type* Send, Execute, Reveal, Corrupt, *respectively, and n_{ro} queries to the random oracles. Then for $t' = O(t+((n_{\mathrm{ro}})^2+n_{\mathrm{se}}+n_{\mathrm{ex}})t_{\mathsf{exp}})$:*

$$\mathsf{Adv}_P^{\mathrm{ake\text{-}fs}}(\mathcal{A}) = \frac{n_{\mathrm{se}}}{N} + O\left(n_{\mathrm{se}}\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', (n_{\mathrm{ro}})^2) + \frac{(n_{\mathrm{se}} + n_{\mathrm{ex}})(n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})}{q}\right).$$

Using essentially the same arguments, we can also show that

$$\mathsf{Adv}_P^{\mathrm{ma}}(\mathcal{A}) = \frac{n_{\mathrm{se}}}{N} + O\left(n_{\mathrm{se}}\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', (n_{\mathrm{ro}})^2) + \frac{(n_{\mathrm{se}} + n_{\mathrm{ex}})(n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})}{q}\right).$$

Proof: Our proof will proceed by introducing a series of protocols $P_0, P_1, \ldots, P_7$ related to P, with $P_0 = P$. In P_7, $\mathcal{A}$ will be reduced to a simple online guessing attack that will admit a straightforward analysis. We describe these protocols informally in Figure 6. For each i from

P_0 The original protocol P.

P_1 If honest parties randomly choose m or μ values seen previously in the execution of the protocol, the protocol halts and the adversary fails.

P_2 The protocol answers Send and Execute queries without making any random oracle queries. Subsequent random oracle queries by the adversary are backpatched, as much as possible, to be consistent with the responses to the Send and Execute queries. (This is a standard technique for proofs involving random oracles.)

P_3 If an $H_\ell(\cdot)$ query is made, for $\ell \in \{2, 3, 4\}$, it is not checked for consistency against Execute queries. That is, instead of backpatching to maintain consistency with an Execute query, the protocol responds with a random output.

P_4 If before a Corrupt query, a correct password guess is made against a client instance or server instance (determined by an $H_\ell(\cdot)$ query, for $\ell \in \{2, 3, 4\}$, using the correct inputs to compute a session key, k, or k' value), the protocol halts and the adversary automatically succeeds.

P_5 If the adversary makes a password guess against client and server instances that are partnered, the protocol halts and the adversary fails.

P_6 If the adversary makes two password guesses against the same server instance the protocol halts and the adversary fails.

P_7 The protocol uses an internal password oracle that holds all passwords and accepts queries that test whether a given password is the correct password for a given client/server pair. (It also accepts Corrupt (U) queries and returns $\langle \pi_U[C] \rangle_{C \in Clients}$ if $U \in Servers$, and otherwise returns π_U.) The test for correct password guesses (from P_4) is changed so that whenever the adversary makes a password guess, a query is submitted to the oracle to determine if it is correct.

Figure 6: Informal description of protocols P_0 through P_7

$P_0 \to P_1$ This is straightforward.

$P_1 \to P_2$ By inspection, the two protocols are indistinguishable unless the adversary makes an $H_\ell(\langle C, S, \cdot, \cdot, \cdot, \gamma' \rangle)$ query, for $\ell \in \{2, 3, 4\}$, where $\gamma' = H_1(\pi_C)$, but the adversary has not actually made the $H_1(\pi_C)$ query, or the adversary makes a Client Action 1 query (resp. Server Action 2 query) with a k (resp. k') value that is not the output of an $H_2(\cdot)$ (resp. $H_3(\cdot)$) query that would be a correct password guess. However, the probability of these is negligible.

$P_2 \to P_3$ This can be shown using a standard reduction from CDH. On input (X, Y), we plug in X multiplied by random powers of g for the clients' m values, and Y multiplied by random powers of g for the servers' μ values. Then we plug in random powers of g for the outputs of $H_1(\cdot)$, so that from a correct $H_\ell(\cdot)$ query, for $\ell \in \{2, 3, 4\}$, we can compute $\mathsf{DH}(X, Y)$.

$P_3 \to P_4$ This is obvious.

$P_4 \to P_5$ This is a reduction from CDH, similar to the one for Execute queries. On input (X, Y), we plug in X for a certain client's m value, and Y multiplied by random powers of g for the servers' μ values associated with that client. Then we plug in random powers of g for the outputs of $H_1(\cdot)$, so that from a correct $H_\ell(\cdot)$ query, for $\ell \in \{2, 3, 4\}$, we can compute $\mathsf{DH}(X, Y)$.

$P_5 \to P_6$ This can be shown using a reduction from CDH. On input (X, Y), we randomly plug in X or 1 multiplied by random powers of g into the outputs of $H_1(\cdot)$ queries, and Y multiplied by random powers of g for the servers' μ values. Finally, for pairs of $H_\ell(\cdot)$ queries, for $\ell \in \{2, 3, 4\}$ corresponding to password guesses using particular m and μ values, we can divide out the common $\mathsf{DH}(m, Y)$ value and compute $\mathsf{DH}(X, Y)$ (with probability $\frac{1}{2}$).

$P_6 \to P_7$ By inspection, these two protocols are indistinguishable.

Figure 7: Proof sketches of negligible advantage gain from P_{i-1} to P_i

1 to 7, we will prove that the advantage of $\mathcal{A}$ attacking protocol P_{i-1} is at most negligibly more than the advantage of $\mathcal{A}$ attacking protocol P_i.

We sketch these proofs in Figure 7.

Now we proceed to the detailed proof.

We use the terminology "in a CLIENT ACTION i query to Π_i^C" to mean "in a Send query to Π_i^C that results in the CLIENT ACTION i procedure being executed," and "in a SERVER ACTION i query to Π_j^S" to mean "in a Send query to Π_j^S that results in the SERVER ACTION i procedure being executed,"

We assume without loss of generality that n_{ro} and $n_{\mathrm{se}} + n_{\mathrm{ex}}$ are both at least 1. We make the standard assumption that random oracles are built "on the fly," that is, each new query to a random oracle is answered with a fresh random output, and each query that is not new is answered consistently with the previous queries. We also assume that in P_0 that $H_1(\cdot)$ queries are answered in the following way: In an $H_1(\pi)$ query, output $g^{\psi_1[\pi]}$, where $\psi_1[\pi] \xleftarrow{R} \mathbb{Z}_q$.[16] That is, we assume we know the discrete logs of the outputs of $H_1(\cdot)$ queries. Finally, we assume that for each $H_\ell(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query made by $\mathcal{A}$ for $\ell \in \{2, 3, 4\}$, the corresponding $H_{\ell'}(\cdot)$ and $H_{\ell''}(\cdot)$ queries are made automatically, where $\{\ell', \ell''\} = \{2, 3, 4\} \setminus \{\ell\}$. All queries are considered to be made by $\mathcal{A}$, though $\mathcal{A}$ only sees the output of the original one.

Say a client instance Π_i^C is *paired with* a server instance Π_j^S if there is a CLIENT ACTION 0 query to Π_i^C with input S and output $\langle C, m \rangle$, there is a SERVER ACTION 1 query to Π_j^S with input $\langle C, m \rangle$ and output $\langle \mu, k \rangle$, and there is a CLIENT ACTION 1 query to Π_i^C with input $\langle \mu, k \rangle$. Say a server instance Π_j^S is *paired with* a client instance Π_i^C if there is a CLIENT ACTION 0 query to Π_i^C with input S and output $\langle C, m \rangle$, there is a SERVER ACTION 1 query to Π_j^S with input $\langle C, m \rangle$ and output $\langle \mu, k \rangle$, and if there is a SERVER ACTION 2 query to Π_j^S with input k', then there was a CLIENT ACTION 1 query to Π_i^C with input $\langle \mu, k \rangle$ and output k'. Note that a client instance paired to a server instance will have terminated, and will be partnered with that server instance. On the other hand, a server instance may be paired to a client instance but still not have terminated. However, if it is paired to a client instance after the SERVER ACTION 2 query, then it will have terminated, and be partnered with that client instance.

We now define some events, corresponding to the adversary making a password guess against a client instance, against a server instance, and against a client instance and server instance that are partnered, respectively.

- testpw(C, i, S, π, ℓ): for some m, μ, γ', and k, $\mathcal{A}$ makes an $H_\ell(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query, a CLIENT ACTION 0 query to a client instance Π_i^C with input S and output $\langle C, m \rangle$, a CLIENT ACTION 1 query to Π_i^C with input $\langle \mu, k \rangle$ and an $H_1(\pi)$ query returning $(\gamma')^{-1}$, where the latest query is either the $H_\ell(\cdot)$ query or the CLIENT ACTION 1

[16] Note that we do this by fixing the underlying $H(\cdot)$ query. See Section 5.

query, $\sigma = DH(\alpha, \mu)$ and $m = \alpha \cdot (\gamma')^{-1}$. The associated value of this event is the output of the $H_\ell(\cdot)$ query, or the k, k', or sk_C^i value (respectively, for $\ell = 2$, 3, or 4) generated by Π_i^C, whichever was set first.

- $\mathsf{testpw!}(C, i, S, \pi)$: for some k, a Client Action 1 query with input $\langle \mu, k \rangle$ causes a $\mathsf{testpw}(C, i, S, \pi, 2)$ event to occur, with associated value k.

- $\mathsf{testpw}(S, j, C, \pi, \ell)$: for some m, μ, γ', and k, $\mathcal{A}$ makes an $H_\ell(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query, and previously made a Server Action 1 query to a server instance Π_j^S with input $\langle C, m \rangle$ and output $\langle \mu, k \rangle$, and an $H_1(\pi)$ query returning $(\gamma')^{-1}$, where $\sigma = DH(\alpha, \mu)$, $m = \alpha \cdot (\gamma'_1)^{-1}$, and ACCEPTABLE$(m)$. The associated value of this event is k, k'', or sk_S^j (respectively, for $\ell = 2$, 3, or 4) generated by Π_j^S.

- $\mathsf{testpw!}(S, j, C, \pi)$: a Server Action 2 query to Π_j^S is made with input k', and previously a $\mathsf{testpw}(S, j, C, \pi, 3)$ event occurs with associated value k'.

- $\mathsf{testpw}^*(S, j, C, \pi)$: $\mathsf{testpw}(S, j, C, \pi, \ell)$ occurs for some $\ell \in \{2, 3, 4\}$.

- $\mathsf{testpw}(C, i, S, j, \pi)$: for some $\ell \in \{2, 3, 4\}$, both a $\mathsf{testpw}(C, i, S, \pi, \ell)$ event and a $\mathsf{testpw}(S, j, C, \pi, \ell)$ event occur, where Π_i^C is paired with Π_j^S and Π_j^S is paired with Π_i^C after its Server Action 1 query.

- $\mathsf{testexecpw}(C, i, S, j, \pi)$: for some m, μ, and γ', $\mathcal{A}$ makes an $H_\ell(\langle C, S, m, \mu, \sigma, \gamma'_1 \rangle)$ query, for $\ell \in \{2, 3, 4\}$, and previously made an $\mathsf{Execute}$ $(C,\ i,\ S,\ j)$ query that generates m and μ, and an $H_1(\pi)$ query returning $(\gamma')^{-1}$, where $\sigma = DH(\alpha, \mu)$, and $m = \alpha \cdot (\gamma')^{-1}$.

- $\mathsf{correctpw}$: before any $\mathsf{Corrupt}$ query, either a $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs for some C, i, and S, or a $\mathsf{testpw}^*(S, j, C, \pi_C)$ event occurs for some S, j, and C.

- $\mathsf{correctpwexec}$: a $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event occurs for some C, i, S, and j.

- $\mathsf{doublepwserver}$: before any $\mathsf{Corrupt}$ query, both a $\mathsf{testpw}^*(S, j, C, \pi)$ event and a $\mathsf{testpw}^*(S, j, C, \pi')$ event occur, for some S, j, C, π and π', with $\pi \neq \pi'$.

- $\mathsf{pairedpwguess}$: a $\mathsf{testpw}(C, i, S, j, \pi_C)$ event occurs, for some C, i, S, and j.

Protocol P_1. Let E_1 be the event that an m value generated in a Client Action 0 or $\mathsf{Execute}$ query is equal to an m value generated in a previous Client Action 0 or $\mathsf{Execute}$ query, an m value sent as input in a previous Server Action 1 query, or an m value in a previous $H_\ell(\cdot)$ query (made by the adversary), for $\ell \in \{2, 3, 4\}$. Let E_2 be the event that a μ value generated in a Server Action 1 or $\mathsf{Execute}$ query is equal to a μ value generated in a previous Server Action 1 or $\mathsf{Execute}$ query, a μ value sent as input in a previous Client Action 1 query, or a μ value in a previous $H_\ell(\cdot)$ query (made by the adversary), for $\ell \in \{2, 3, 4\}$.

Let $E = E_1 \vee E_2$. Let P_1 be a protocol that is identical to P_0 except that if E occurs, the protocol aborts (and thus the adversary fails).

Claim 6.10 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathrm{ake}}_{P_0}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{ake}}_{P_1}(\mathcal{A}) + \frac{O((n_{\mathrm{se}}+n_{\mathrm{ex}})(n_{\mathrm{ro}}+n_{\mathrm{se}}+n_{\mathrm{ex}}))}{q}.$$

Proof: Straightforward. ▌

Protocol P_2. Let P_2 be a protocol that is identical to P_1 except that Send and Execute queries are answered without making any random oracle queries, and subsequent random oracle queries by the adversary are backpatched, as much as possible, to be consistent with the responses to the Send and Execute queries.

Specifically, the queries in P_2 are changed as follows:

- In an Execute $(C,\ i,\ S,\ j)$ query, $m \leftarrow g^{\tau[i,C]}$, where $\tau[i,C] \stackrel{R}{\leftarrow} \mathbb{Z}_q$, $\mu \leftarrow g^{\tau[j,S]}$, where $\tau[j,S] \stackrel{R}{\leftarrow} \mathbb{Z}_q$, $k, k' \stackrel{R}{\leftarrow} \{0,1\}^\kappa$, and $sk_C^i \leftarrow sk_S^j \stackrel{R}{\leftarrow} \{0,1\}^\kappa$.
- In a Client Action 0 query to instance Π_i^C, $m \leftarrow g^{\tau[i,C]}$, where $\tau[i,C] \stackrel{R}{\leftarrow} \mathbb{Z}_q$.
- In a Server Action 1 query to instance Π_j^S, $\mu \leftarrow g^{\tau[j,S]}$, where $\tau[j,S] \stackrel{R}{\leftarrow} \mathbb{Z}_q$, and $sk_S^j, k, k'' \stackrel{R}{\leftarrow} \{0,1\}^\kappa$.
- In a Client Action 1 query to instance Π_i^C, do the following.
  - If this query causes a $\mathsf{testpw!}(C,i,S,\pi_C)$ event to occur, then set k' to associated value of the $\mathsf{testpw}(C,i,S,\pi_C,3)$ event, and set sk_C^i to associated value of the $\mathsf{testpw}(C,i,S,\pi_C,4)$ event.
  - Otherwise, if Π_i^C is paired with a server instance Π_j^S, $sk_C^i \leftarrow sk_S^j$. Then $k' \stackrel{R}{\leftarrow} \{0,1\}^\kappa$.
  - Otherwise, Π_i^C aborts.
- In a Server Action 2 query to instance Π_j^S, if this query causes a $\mathsf{testpw!}(S,i,C,\pi_C)$ event to occur, or if Π_j^S is paired with a client instance Π_i^C, terminate. Otherwise, Π_j^S aborts.
- In an $H_\ell(\langle C,S,m,\mu,\sigma,\gamma'\rangle)$ query, for $\ell \in \{2,3,4\}$, if this $H_\ell(\cdot)$ query causes a $\mathsf{testpw}(S,j,C,\pi_C,\ell)$, or $\mathsf{testexecpw}(C,i,S,j,\pi_C)$ event to occur, then output the associated value of that event, and otherwise output a random value from $\{0,1\}^\kappa$.

Claim 6.11 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathrm{ake}}_{P_1}(\mathcal{A}) = \mathsf{Adv}^{\mathrm{ake}}_{P_2}(\mathcal{A}) + \frac{O(n_{\mathrm{ro}}+n_{\mathrm{se}})}{q}.$$

Proof: One can see that in P_1, a server instance Π_j^S in a SERVER ACTION 1 query creates a session key sk_S^j and k and k'' values that are uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since the $H_2(\cdot)$, $H_3(\cdot)$, and $H_4(\cdot)$ queries that determine these values are new. Then in a SERVER ACTION 2 query, if a $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs, or Π_j^S is paired, the instance terminates, and if Π_j^S is unpaired and no $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs, then either the instance terminates or aborts. It is easy to show that the total probability of any instance terminating in this case is at most $\frac{n_{\text{se}}}{2^\kappa}$

Also in P_1, for any client instance Π_i^C, either

1. a $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs, and then k' and sk_C^i are set to the values associated with the $\mathsf{testpw}(C, i, S, \pi_C, 3)$ and $\mathsf{testpw}(C, i, S, \pi_C, 4)$ events, respectively, which are guaranteed to occur by our original assumption that all queries to $H_2(\cdot)$, $H_3(\cdot)$, and $H_4(\cdot)$ using the same inputs are made simultaneously, or

2. no $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs, but exactly one instance Π_j^S is paired with Π_i^C, in which case $sk_C^i = sk_S^j$, and k' is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred (since no $\mathsf{testpw}(C, i, S, \pi_C, 3)$ event could have occurred in this case), or

3. no $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs and no instance is paired with Π_i^C, then either the instance terminates or aborts. It is easy to show that the total probability of any instance terminating in this case is at most $\frac{n_{\text{se}}}{2^\kappa}$

For any $H_2(\langle C, S, \cdot, \cdot, \cdot, \gamma' \rangle)$ query, either (1) it causes a $\mathsf{testpw}(S, j, C, \pi_C, 2)$, or $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event to occur, in which case the output is the associated value of that event (i.e., the k value associated with the particular event that occurs), (2) it does not cause a $\mathsf{testpw}(C, i, S, j, \pi_C)$ event, but does cause a $\mathsf{testpw}(C, i, S, \pi_C, 2)$ event to occur, where the CLIENT ACTION 1 query of the event had input $\langle \mu, k \rangle$ for some μ, in which case either Π_i^C terminated and the output is k, or Π_i^C aborted and the output is uniformly chosen from $\{0,1\}^\kappa \setminus \{k\}$, (3) $\gamma' = H_1(\pi_C)$, but the adversary has not made an $H_1(\pi_C)$ query, or (4) the output of $H_2(\cdot)$ is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since this is a new $H_2(\cdot)$ query. The total probability of an $H_2(\cdot)$ query causing the third case above can be easily shown to be bounded by $\frac{n_{\text{ro}}}{q}$. The second case above where the output is fixed can only occur when an unpaired client instance terminated with no $\mathsf{testpw!}(C, i, S, \pi_C)$ event.

For any $H_3(\langle C, S, \cdot, \cdot, \cdot, \gamma' \rangle)$ query, either (1) it causes a $\mathsf{testpw}(S, j, C, \pi_C, 3)$, or $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event to occur, in which case the output is the associated value of that event (i.e., the k' or k'' value associated with the particular event that occurs), (2) it does not cause a $\mathsf{testpw}(C, i, S, j, \pi_C)$ event, but it causes a $\mathsf{testpw}(C, i, S, \pi_C, 3)$ event to occur, where Π_i^C terminated, (3) $\gamma' = H_1(\pi_C)$, but the adversary has not made an $H_1(\pi_C)$ query, or (4) the output of $H_3(\cdot)$ is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since this is a new $H_3(\cdot)$ query. The total probability of an $H_3(\cdot)$ query causing the third case above can be easily shown to be bounded by $\frac{n_{\rm ro}}{q}$. The second case can only occur when an $H_2(\cdot)$ query causes a second case where its output is fixed (see above).

For any $H_4(\langle C, S, \cdot, \cdot, \cdot, \gamma' \rangle)$ query, either (1) it causes a $\mathsf{testpw}(S, j, C, \pi_C, 4)$, or $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event to occur, in which case the output is the associated value of that event (i.e., the sk_S^j value associated with the particular event that occurs), (2) it does not cause a $\mathsf{testpw}(C, i, S, j, \pi_C)$ event, but it causes a $\mathsf{testpw}(C, i, S, , \pi_C, 4)$ event to occur, where Π_i^C terminated, (3) $\gamma' = H_1(\pi_C)$, but the adversary has not made an $H_1(\pi_C)$ query, or (4) the output of $H_4(\cdot)$ is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since this is a new $H_4(\cdot)$ query. The total probability of an $H_4(\cdot)$ query causing the third case above can be easily shown to be bounded by $\frac{n_{\rm ro}}{q}$. The second case can only occur when an $H_2(\cdot)$ query causes a second case where its output is fixed (see above).

If an unpaired client instance Π_i^C never terminates without a $\mathsf{testpw!}(C, i, S, \pi_C)$ event, an unpaired server instance Π_j^S never terminates without a $\mathsf{testpw!}(S, j, C, \pi_C)$ event, and the third case of the $H_2(\cdot)$, $H_3(\cdot)$, and $H_4(\cdot)$ queries never occurs, then P_2 is consistent with P_1. The claim follows. ▌

Protocol P_3. Let P_3 be a protocol that is identical to P_2 except that in an $H_\ell(\langle C, S, \cdot, \cdot, \cdot, \cdot \rangle)$ query, for $\ell \in \{2, 3, 4\}$, there is no check for a $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event.

Claim 6.12 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + (n_{\rm ro} + n_{\rm se} + n_{\rm ex}) t_{\mathsf{exp}})$ such that

$$\mathsf{Adv}^{\rm ake}_{P_2}(\mathcal{A}) \le \mathsf{Adv}^{\rm ake}_{P_3}(\mathcal{A}) + 2\mathsf{Adv}^{\rm CDH}_{G_q}(t', n_{\rm ro}).$$

Proof: Let E be the event that a $\mathsf{correctpwexec}$ event occurs. Obviously, if E does not occur, then P_2 and P_3 are indistinguishable. Let ϵ be the probability that E occurs when $\mathcal{A}$ is running against protocol P_1. Then $\Pr(\mathsf{Succ}^{\rm ake}_{P_2}(\mathcal{A})) \le \Pr(\mathsf{Succ}^{\rm ake}_{P_3}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}^{\rm ake}_{P_2}(\mathcal{A}) \le \mathsf{Adv}^{\rm ake}_{P_3}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts to solve CDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y), D simulates P_2 for $\mathcal{A}$ with these changes:

1. In an $\mathsf{Execute}$ $(C,\ i,\ S,\ j)$ query, set $m \leftarrow X g^{\rho_{i,C}}$ and $\mu \leftarrow Y g^{\rho'_{j,S}}$, where $\rho_{i,C}, \rho'_{j,S} \stackrel{R}{\leftarrow} \mathbb{Z}_q$.
2. When $\mathcal{A}$ finishes, for every $H_\ell(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query, for $\ell \in \{2, 3, 4\}$, where m and μ were generated in an $\mathsf{Execute}$ $(C,\ i,\ S,\ j)$ query, and an $H_1(\pi_C)$ query returned $(\gamma')^{-1}$, add
$$\sigma X^{-\rho'_{j,S}} Y^{-\rho_{i,C}} g^{-\rho_{i,C}\rho'_{j,S}} m^{\psi_2[\pi_C]} \mu^{\psi_1[\pi_C]} g^{-\psi_1[\pi_C]\psi_2[\pi_C]}$$
to the list of possible values for $\mathsf{DH}(X, Y)$.

This simulation is perfectly indistinguishable from P_2 until E occurs, and in this case, D adds the correct $\mathsf{DH}(X,Y)$ to the list. After E occurs the simulation may be distinguishable from P_2, but this does not change the fact that E occurs with probability ϵ. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulator can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguishes the simulation from P_2.

D creates a list of size n_{ro}, and its advantage is ϵ. Let t' be the running time of D, and note that $t' = O(t + (n_{\text{ro}} + n_{\text{se}} + n_{\text{ex}})t_{\mathsf{exp}})$. The claim follows from the fact that $\mathsf{Adv}_{G_q}^{\text{CDH}}(D) \leq \mathsf{Adv}_{G_q}^{\text{CDH}}(t', n_{\text{ro}})$. ▮

Protocol P_4. Let P_4 be a protocol that is identical to P_3 except that if $\mathsf{correctpw}$ occurs then the protocol halts and the adversary automatically succeeds. Note that this involves the following changes:

1. In a Client Action 1 query to Π_i^C, if a $\mathsf{testpw!}(C,i,S,\pi_C)$ event occurs and no $\mathsf{Corrupt}$ query has been made, halt and say the adversary automatically succeeds.
2. In an $H_\ell(\cdot)$ query, for $\ell \in \{2,3,4\}$, if a $\mathsf{testpw}^*(S,j,C,\pi_C)$ event occurs and no $\mathsf{Corrupt}$ query has been made, halt and say the adversary automatically succeeds.

Claim 6.13 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}_{P_3}^{\text{ake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_4}^{\text{ake}}(\mathcal{A}).$$

Proof: Obvious. ▮

Note that in P_4, until $\mathsf{correctpw}$ or a $\mathsf{Corrupt}$ query occurs, no unpaired client or server instance will terminate.

Protocol P_5. Let P_5 be a protocol that is identical to P_4 except that if a $\mathsf{pairedpwguess}$ event occurs, the protocol halts and the adversary fails. We assume that when a query is made, the test for $\mathsf{pairedpwguess}$ occurs before the test for $\mathsf{correctpw}$. Note that this involves the following change: if a $\mathsf{testpw}(C,i,S,\pi,\ell)$ event occurs (this should be checked in a Client Action 1 query, or an $H_\ell(\cdot)$ query) for $\ell \in \{2,3,4\}$, check if a $\mathsf{testpw}(C,i,S,j,\pi)$ event also occurs.

Claim 6.14 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t+(n_{\text{ro}}+n_{\text{se}}+n_{\text{ex}})t_{\mathsf{exp}})$ such that

$$\mathsf{Adv}_{P_4}^{\text{ake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_5}^{\text{ake}}(\mathcal{A}) + 2n_{\text{se}} \cdot \mathsf{Adv}_{G_q}^{\text{CDH}}(t', n_{\text{ro}}).$$

Proof: Obviously, if $\mathsf{pairedpwguess}$ does not occur, then P_4 and P_5 are indistinguishable. Let ϵ be the probability that $\mathsf{pairedpwguess}$ occurs when $\mathcal{A}$ is running against protocol P_4. Then $\Pr(\mathsf{Succ}^{\mathrm{ake}}_{P_4}(\mathcal{A})) \le \Pr(\mathsf{Succ}^{\mathrm{ake}}_{P_5}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}^{\mathrm{ake}}_{P_4}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{ake}}_{P_5}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts to solve CDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y), D chooses a random $d \in \{1, \ldots, n_{\mathrm{se}}\}$ and simulates P_4 for $\mathcal{A}$ with these changes:

1. In the dth Client Action 0 query, say to a client instance $\Pi^C_{i'}$, with input S, set $m \leftarrow X$.
2. In a Server Action 1 query to a server instance Π^S_j with input $\langle C, m\rangle$ where there was a Client Action 0 query to $\Pi^C_{i'}$ (i.e., the instance with the dth Client Action 0 query) with input S and output $\langle C, m\rangle$, set $\mu \leftarrow Y g^{\rho'_{j,S}}$.
3. In a Client Action 1 query to $\Pi^C_{i'}$, if $\Pi^C_{i'}$ is unpaired, D outputs 0 and halts.
4. In a Server Action 2 query to Π^S_j, if Π^S_j was paired with $\Pi^C_{i'}$ after its Server Action 1 query, but is not now paired with $\Pi^C_{i'}$, no test for $\mathsf{correctpw}$ is made, and Π^S_j aborts.
5. When $\mathcal{A}$ finishes, for every $H_\ell(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ query, for $\ell \in \{2, 3, 4\}$ where m and μ were generated by $\Pi^C_{i'}$ and a server instance Π^S_j, respectively, where Π^S_j was paired with $\Pi^C_{i'}$ after its Server Action 1 query, and an $H_1(\pi)$ query returned $(\gamma')^{-1}$, add
$$\sigma X^{-\rho'_{j,S}} \mu^{\psi_1[\pi]}$$
to the list of possible values for $\mathsf{DH}(X, Y)$.

This simulation is perfectly indistinguishable from P_4 until (1) a $\mathsf{testpw}(S, j, C, \pi)$ event occurs, where Π^S_j was paired with $\Pi^C_{i'}$ after the Server Action 1 query, or (2) $\Pi^C_{i'}$ is not paired with a server instance when the Client Action 1 query is made. Note that the probability of a $\mathsf{pairedpwguess}$ event occurring for $\Pi^C_{i'}$ is at least $\frac{\epsilon}{n_{\mathrm{se}}}$, and this is at most the probability of an event of type (1) occurring, since an event of type (2) implies that $\mathsf{pairedpwguess}$ would never have occurred in P_4 for $\Pi^C_{i'}$ (see P_1). If an event of type (1) occurs, D adds the correct $\mathsf{DH}(X, Y)$ to the list.

Note that in either case, the simulation may be distinguishable from P_4, but this does not change the fact that a $\mathsf{pairedpwguess}$ event will occur for $\Pi^C_{i'}$ with probability at least $\frac{\epsilon}{n_{\mathrm{se}}}$ in the simulation. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulator can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguishes the simulation from P_4.

D creates a list of size n_{ro}, and its advantage is $\frac{\epsilon}{n_{\mathrm{se}}}$. Let t' be the running time of D, and note that $t' = O(t + (n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$. The claim follows from the fact that $\mathsf{Adv}^{\mathrm{CDH}}_{G_q}(D) \leq \mathsf{Adv}^{\mathrm{CDH}}_{G_q}(t', n_{\mathrm{ro}})$. ∎

Protocol P_6. Let P_6 be a protocol that is identical to P_5 except that if $\mathsf{doublepwserver}$ occurs, the protocol halts and the adversary fails. We assume that when a query is made, the test for $\mathsf{doublepwserver}$ occurs before the test for $\mathsf{pairedpwguess}$ or $\mathsf{correctpw}$.

Claim 6.15 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + ((n_{\mathrm{ro}})^2 + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$ such that

$$\mathsf{Adv}^{\mathrm{ake}}_{P_5}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{ake}}_{P_6}(\mathcal{A}) + 2\mathsf{Adv}^{\mathrm{CDH}}_{G_q}(t', (n_{\mathrm{ro}})^2).$$

Proof: Let ϵ be the probability that $\mathsf{doublepwserver}$ occurs when $\mathcal{A}$ is running against protocol P_5. Then $\Pr(\mathsf{Succ}^{\mathrm{ake}}_{P_5}(\mathcal{A})) \leq \Pr(\mathsf{Succ}^{\mathrm{ake}}_{P_6}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}^{\mathrm{ake}}_{P_5}(\mathcal{A}) \leq \mathsf{Adv}^{\mathrm{ake}}_{P_6}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts to solve CDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y), D simulates P_5 for $\mathcal{A}$ with these changes:

1. In an $H_1(\pi)$ query, output $X^{\psi_1[\pi]} g^{\psi'_1[\pi]}$, where $\psi_1[\pi] \xleftarrow{R} \{0, 1\}$ and $\psi'_1[\pi] \xleftarrow{R} \mathbb{Z}_q$.
2. In a SERVER ACTION 1 query to a server instance Π^S_j with input $\langle C, m\rangle$ where $\textsc{acceptable}(m)$ is true, set $\mu \leftarrow Y g^{\rho'_{j,S}}$.
3. Tests for $\mathsf{correctpw}$ (from P_4) and $\mathsf{pairedpwguess}$ (from P_5) are not made. In particular, unpaired client instances that receive a CLIENT ACTION 1 query abort, and unpaired server instances that receive a SERVER ACTION 2 query abort. Also, $H_\ell(\cdot)$ queries always return values uniformly chosen from $\{0, 1\}^\kappa$.
4. When $\mathcal{A}$ finishes, for every pair of queries $H_\ell(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ and $H_{\hat{\ell}}(\langle C, S, m, \mu, \hat{\sigma}, \hat{\gamma}'\rangle)$, for $\ell, \hat{\ell} \in \{2, 3, 4\}$, where $\textsc{acceptable}(\sigma)$ and $\textsc{acceptable}(\hat{\sigma})$ are true, there was a SERVER ACTION 1 query to a server instance Π^S_j with input $\langle C, m\rangle$ and output $\langle \mu, k\rangle$ (and thus $\textsc{acceptable}(m)$ is true), an $H_1(\pi)$ query that returned $(\gamma')^{-1}$, an $H_1(\hat{\pi})$ query that returned $(\hat{\gamma}')^{-1}$, and $\psi_1[\pi] \neq \psi_1[\hat{\pi}]$, add

$$\left(\sigma\hat{\sigma}^{-1}(\gamma')^{\rho'_{j,S}}(\hat{\gamma}')^{-\rho'_{j,S}} Y^{\psi'_1[\pi] - \psi'_1[\hat{\pi}]}\right)^{\psi_1[\hat{\pi}] - \psi_1[\pi]}$$

to the list of possible values for $\mathsf{DH}(X, Y)$.

This simulation is perfectly indistinguishable from P_5 until a $\mathsf{doublepwserver}$ event, a $\mathsf{pairedpwguess}$ event, or a $\mathsf{correctpw}$ event occurs, or $\mathcal{A}$ makes a $\mathsf{Corrupt}$ query. If a

$\mathsf{doublepwserver}$ event occurs, then with probability $\frac{1}{2}$ it occurs for two passwords π and $\hat{\pi}$ with $\psi_1[\pi] \neq \psi_1[\hat{\pi}]$, and in this case D adds the correct $\mathsf{DH}(X,Y)$ to the list. If a $\mathsf{correctpw}$ event or a $\mathsf{pairedpwguess}$ event occurs, then the $\mathsf{doublepwserver}$ event would never have occurred in P_5, since P_5 would halt. Also, if a $\mathsf{Corrupt}$ query is made before a $\mathsf{doublepwserver}$ event, then a $\mathsf{doublepwserver}$ event would never occur (by definition). Note that in either of these cases, the simulation may be distinguishable from P_5, but this does not change the fact that a $\mathsf{doublepwserver}$ event will occur with probability at least ϵ in the simulation. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulator can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguishes the simulation from P_5.

D creates a list of size $(n_{\mathrm{ro}})^2$, and its advantage is $\frac{\epsilon}{2}$. Let t' be the running time of D, and note that $t' = O(t + ((n_{\mathrm{ro}})^2 + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$. The claim follows from the fact that $\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(D) \leq \mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', (n_{\mathrm{ro}})^2)$. ▮

Protocol P_7. Let P_7 be a protocol that is identical to P_6 except that there is a new internal oracle (i.e., not available to the adversary) that handles passwords, called a *password oracle*. This oracle generates all passwords during initialization. Then it accepts queries of the form $\mathsf{testpw}(C, \pi)$ and returns TRUE if $\pi = \pi_C$, and FALSE otherwise. It also accepts $\mathsf{Corrupt}\,(U)$ queries and returns $\langle \pi_U[C] \rangle_{C \in \mathit{Clients}}$ if $U \in \mathit{Servers}$, and otherwise returns π_U. When a $\mathsf{Corrupt}\,(U)$ query is received in the protocol, it is answered using a $\mathsf{Corrupt}\,(U)$ query to the password oracle. The protocol is also changed in the method for determining $\mathsf{correctpw}$. Specifically, to test if $\mathsf{correctpw}$ occurs, whenever the first $\mathsf{testpw}(C, i, S, \pi)$ event occurs for an instance Π_i^C and password π, or the first $\mathsf{testpw}(S, j, C, \pi)$ event occurs for an instance Π_j^S and password π, a $\mathsf{testpw}(C, \pi)$ query is made to the password oracle to see if $\pi = \pi_C$.

Claim 6.16 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}_{P_6}^{\mathrm{ake}}(\mathcal{A}) = \mathsf{Adv}_{P_7}^{\mathrm{ake}}(\mathcal{A}).$$

Proof: By inspection, P_6 and P_7 are perfectly indistinguishable. ▮

The probability of the adversary $\mathcal{A}$ succeeding in P_7 is bounded by

$$\Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A})) \leq \Pr(\mathsf{correctpw}) + \Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A}) | \neg \mathsf{correctpw}).$$

First, since there are at most n_{se} queries to the password oracle before a $\mathsf{Corrupt}$ query, and passwords are chosen uniformly from a dictionary of size N, $\Pr(\mathsf{correctpw}) \leq \frac{n_{\mathrm{se}}}{N}$.

Now we compute $\Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A}) | \neg \mathsf{correctpw})$. If $\mathsf{correctpw}$ does not occur, then $\mathcal{A}$ succeeds by making a Test query to a fresh instance Π_i^U and guessing the bit used in that Test query. We will show that the view of the adversary is independent of sk_U^i, and thus the probability of success is exactly $\frac{1}{2}$.

First we examine Reveal queries. Recall that since Π_i^U is fresh, there could be no $\mathsf{Reveal}\,(U,\ i)$ query, and if $\Pi_j^{U'}$ is partnered with Π_i^U, no $\mathsf{Reveal}\,(U',\ j)$ query. Second note that since *sid* includes m and μ values, if more than a single client instance and a single server instance accept with the same *sid*, $\mathcal{A}$ fails (see P_1). Thus the output of Reveal queries is independent of sk_U^i.

Second we examine $H_4(\cdot)$ queries. As noted in the discussion following the description of P_4, until a $\mathsf{correctpw}$ event or a $\mathsf{Corrupt}$ query, no unpaired client or server instance will terminate, and thus an instance may only be fresh and receive a Test query if it is partnered. However, an $H_4(\cdot)$ query will never reveal sk_U^i if Π_i^U is partnered (see P_5).

This implies that the view of the adversary is independent of sk_U^i, and thus the probability of success is exactly $\frac{1}{2}$.

Since $\Pr(\neg\mathsf{correctpw}) = 1 - \Pr(\mathsf{correctpw})$, we have that

$$
\begin{aligned}
\Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A})) &\leq \Pr(\mathsf{correctpw}) + \Pr(\mathsf{Succ}_{P_7}^{\mathrm{ake}}(\mathcal{A})|\neg\mathsf{correctpw})(1 - \Pr(\mathsf{correctpw})) \\
&\leq \Pr(\mathsf{correctpw}) + \frac{1}{2}(1 - \Pr(\mathsf{correctpw})) \\
&= \frac{1}{2} + \frac{\Pr(\mathsf{correctpw})}{2} \\
&\leq \frac{1}{2} + \frac{n_{\mathrm{se}}}{2N}.
\end{aligned}
$$

Therefore $\mathsf{Adv}_{P_7}^{\mathrm{ake}}(\mathcal{A}) \leq \frac{n_{\mathrm{se}}}{N}$. The theorem follows from this and Claims 6.10 through 6.16. ∎

6.3 PAK-Z Protocol

Here we prove that the PAK-Z protocol is secure, in the sense that an adversary attacking the system cannot determine session keys of fresh instances with greater advantage than that of an online dictionary attack, and cannot determine session keys of semi-fresh instances with greater advantage than that of an offline dictionary attack.

Semi-freshness. Here we modify the two notions of freshness defined earlier to deal with systems that are designed to be resilient to server compromise. Specifically, we will define the notion of semi-freshness that may be applied to server instances, even if some server has been compromised. This is because even after server compromise, the attacker should not be able to authenticate to other servers (at least not unless the attacker runs an offline dictionary attack).

An instance Π_i^U is semi-nfs-fresh (semi-fresh with no requirement for forward secrecy) unless either (1) a $\mathsf{Reveal}\,(U,\ i)$ query occurs, (2) a $\mathsf{Reveal}\,(U',\ j)$ query occurs where $\Pi_{U'}^j$ is the partner of Π_i^U, (3) a $\mathsf{Corrupt}\,(C)$ query occurs, or (4) $U \in$ *Clients* and a $\mathsf{Corrupt}\,(S)$ query occurs. (For convenience, when we do not make a requirement for forward secrecy, we simply disallow $\mathsf{Corrupt}$ queries to clients.) An instance Π_i^U is semi-fs-fresh (semi-fresh with forward secrecy) unless either (1) a $\mathsf{Reveal}\,(U,\ i)$ query occurs, (2) a $\mathsf{Reveal}\,(U',\ j)$ query occurs where $\Pi_j^{U'}$ is the partner of Π_i^U, (3) a $\mathsf{Corrupt}\,(C)$ query occurs before the Test query and a $\mathsf{Send}\,(U,\ i,\ M)$ query occurs for some string M, or (4) $U \in \mathit{Clients}$, a $\mathsf{Corrupt}\,(S)$ query occurs before the Test query and a $\mathsf{Send}\,(U,\ i,\ M)$ query occurs for some string M,

For semi-freshness, we define ake-nfs.s and ake-fs.s similarly to ake-nfs and ake-fs.

We define $\mathsf{Adv}_P^{\mathrm{c2s.s}}(\mathcal{A})$ to be the probability that a server oracle terminates before any $\mathsf{Corrupt}$ query to a client without having a partner oracle.

Theorem 6.17 *Let P be the protocol described in Figure 3 (and formally described in Appendix B), using group G_q and signature scheme $\mathcal{S}$, and with a password dictionary of size N. Fix an adversary $\mathcal{A}$ that runs in time t, and makes $n_{\mathrm{se}}, n_{\mathrm{ex}}, n_{\mathrm{re}}, n_{\mathrm{co}}$ queries of type* $\mathsf{Send}, \mathsf{Execute}, \mathsf{Reveal}, \mathsf{Corrupt}$*, respectively, and n_{ro} queries to the random oracles. Then for $t' = O(t + ((n_{\mathrm{ro}})^2 + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$:*

$$\mathsf{Adv}_P^{\mathrm{ake\text{-}fs}}(\mathcal{A}) \;=\; \frac{n_{\mathrm{se}}}{N} + O\left(n_{\mathrm{se}}\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', (n_{\mathrm{ro}})^2) + \frac{(n_{\mathrm{se}} + n_{\mathrm{ex}})(n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})}{q}\right),$$

and

$$\mathsf{Adv}_P^{\mathrm{ake\text{-}fs.s}}(\mathcal{A}) \;=\; \frac{n_{\mathrm{ro}}[n_{\mathrm{co}} > 0]}{N} + \mathsf{Adv}_P^{\mathrm{ake\text{-}fs}}(\mathcal{A}).$$

Using essentially the same arguments, we can also show that

$$\mathsf{Adv}_P^{\mathrm{ma}}(\mathcal{A}) \;=\; \frac{n_{\mathrm{se}}}{N} + O\left(n_{\mathrm{se}}\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', (n_{\mathrm{ro}})^2) + \frac{(n_{\mathrm{se}} + n_{\mathrm{ex}})(n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})}{q}\right)$$

and

$$\mathsf{Adv}_P^{\mathrm{c2s.s}}(\mathcal{A}) \;=\; \frac{n_{\mathrm{ro}}[n_{\mathrm{co}} > 0]}{N} + \mathsf{Adv}_P^{\mathrm{ma}}(\mathcal{A}).$$

Proof: Our proof will proceed by introducing a series of protocols $P_0, P_1, \ldots, P_8$ related to P, with $P_0 = P$. In P_8, $\mathcal{A}$ will be reduced to a simple online guessing attack that will admit a straightforward analysis. We describe these protocols informally in Figure 8. For each i from 1 to 8, we will prove that the advantage of $\mathcal{A}$ attacking protocol P_{i-1} is at most negligibly more than the advantage of $\mathcal{A}$ attacking protocol P_i.

We sketch these proofs in Figure 9.

Now we proceed to the detailed proof.

We use the terminology "in a Client Action i query to Π_i^C" to mean "in a Send query to Π_i^C that results in the Client Action i procedure being executed," and "in a Server Action i query to Π_j^S" to mean "in a Send query to Π_j^S that results in the Server Action i procedure being executed,"

We assume without loss of generality that n_{ro} and $n_{\mathrm{se}} + n_{\mathrm{ex}}$ are both at least 1. We make the standard assumption that random oracles are built "on the fly," that is, each new query

P_0 The original protocol P.

P_1 If honest parties randomly choose m or μ values seen previously in the execution of the protocol, the protocol halts and the adversary fails.

P_2 The protocol answers Send and $\mathsf{Execute}$ queries without making any random oracle queries. Subsequent random oracle queries by the adversary are backpatched, as much as possible, to be consistent with the responses to the Send and $\mathsf{Execute}$ queries. (This is a standard technique for proofs involving random oracles.)

P_3 If an $H_\ell(\cdot)$ query is made, for $\ell \in \{3,4,5,6\}$, it is not checked for consistency against $\mathsf{Execute}$ queries. That is, instead of backpatching to maintain consistency with an $\mathsf{Execute}$ query, the protocol responds with a random output.

P_4 If before a $\mathsf{Corrupt}$ query, a correct password guess is made against a client instance or server instance (determined by an $H_\ell(\cdot)$ query, for $\ell \in \{3,4,5,6\}$, using the correct inputs to compute a session key, k, a', or s'' value), the protocol halts and the adversary automatically succeeds. For semi-freshness, add the following: If before a $\mathsf{Corrupt}$ query to a client, a correct password guess is made against a server instance (determined by an $H_1(\cdot)$ or $H_2(\cdot)$ query with the correct password and a $\mathsf{Corrupt}$ query to a server) the protocol halts and the adversary automatically succeeds.

P_5 If the adversary makes a password guess against client and server instances that are partnered, the protocol halts and the adversary fails.

P_6 If the adversary is able to produce a valid signature without making an $H_2(\cdot)$ query with the correct password and a $\mathsf{Corrupt}$ query to a server, the protocol halts and the adversary fails.

P_7 If the adversary makes two password guesses against the same server instance the protocol halts and the adversary fails.

P_8 The protocol uses an internal password oracle that holds all passwords and accepts queries that test whether a given password is the correct password for a given client/server pair. (It also accepts $\mathsf{Corrupt}$ (U) queries and returns $\langle \pi_U[C] \rangle_{C \in \mathit{Clients}}$ if $U \in \mathit{Servers}$, and otherwise returns π_U.) The test for correct password guesses (from P_4) is changed so that whenever the adversary makes a password guess, a query is submitted to the oracle to determine if it is correct.

Figure 8: Informal description of protocols P_0 through P_8

$P_0 \to P_1$ This is straightforward.

$P_1 \to P_2$ By inspection, the two protocols are indistinguishable unless the adversary makes an $H_\ell(\langle C, S, \cdot, \cdot, \cdot, \gamma' \rangle)$ query, for $\ell \in \{3, 4, 5, 6\}$, where $\gamma' = H_1(\langle C, \pi_C \rangle)$, but the adversary has not actually made the $H_1(\langle C, \pi_C \rangle)$ query, or the adversary makes a Client Action 1 query (resp. Server Action 2 query) with a k (resp. s') value that is not the output of an $H_3(\cdot)$ query (resp. the correct signature when xor'ed with an $H_6(\cdot)$ query) that would be a correct password guess. However, the probability of these is negligible.

$P_2 \to P_3$ This can be shown using a standard reduction from CDH. On input (X, Y), we plug in X multiplied by random powers of g for the clients' m values, and Y multiplied by random powers of g for the servers' μ values. Then we plug in random powers of g for the outputs of $H_1(\cdot)$, so that from a correct $H_\ell(\cdot)$ query, for $\ell \in \{3, 4, 5, 6\}$, we can compute $\mathsf{DH}(X, Y)$.

$P_3 \to P_4$ This is obvious.

$P_4 \to P_5$ This is a reduction from CDH, similar to the one for Execute queries. On input (X, Y), we plug in X for a certain client's m value, and Y multiplied by random powers of g for the servers' μ values associated with that client. Then we plug in random powers of g for the outputs of $H_1(\cdot)$, so that from a correct $H_\ell(\cdot)$ query, for $\ell \in \{3, 4, 5, 6\}$, we can compute $\mathsf{DH}(X, Y)$.

$P_5 \to P_6$ This can be shown using the existential unforgeability of the signature scheme against chosen message attacks. Given a public key and signature oracle, we plug in the public key for one client/server pair, and use the signature oracle to compute signatures for clients (that may be fooled by an attacker impersonating a server after that attacker has corrupted the server). Note that before an $H_2(\cdot)$ query with the correct password, and a Corrupt query to a server, the adversary would get no information about the secret key for the given client/server pair.

$P_6 \to P_7$ This can be shown using a reduction from CDH. On input (X, Y), we randomly plug in X or 1 multiplied by random powers of g into the outputs of $H_1(\cdot)$ queries, and Y multiplied by random powers of g for the servers' μ values. Finally, for pairs of $H_\ell(\cdot)$ queries, for $\ell \in \{3, 4, 5, 6\}$ corresponding to password guesses using a particular m value, we can divide out the common $\mathsf{DH}(m, Y)$ value and compute $\mathsf{DH}(X, Y)$ (with probability $\frac{1}{2}$).

$P_7 \to P_8$ By inspection, these two protocols are indistinguishable.

Figure 9: Proof sketches of negligible advantage gain from P_{i-1} to P_i

to a random oracle is answered with a fresh random output, and each query that is not new is answered consistently with the previous queries. We also assume that in P_0 that $H_1(\cdot)$ queries are answered in the following way: In an $H_1(\langle C, \pi \rangle)$ query, output $g^{\psi_1[C,\pi]}$, where $\psi_1[C,\pi] \stackrel{R}{\leftarrow} \mathbb{Z}_q$.[17] That is, we assume we know the discrete logs of the outputs of $H_1(\cdot)$ queries. Finally, we assume that for each $H_\ell(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query made by $\mathcal{A}$ for $\ell \in \{3,4,5,6\}$, the corresponding $H_{\ell'}(\cdot)$, $H_{\ell''}(\cdot)$, and $H_{\ell'''}(\cdot)$ queries are made automatically, where $\{\ell', \ell'', \ell'''\} = \{3,4,5,6\} \setminus \{\ell\}$. All queries are considered to be made by $\mathcal{A}$, though $\mathcal{A}$ only sees the output of the original one.

Say a client instance Π_i^C is *paired with* a server instance Π_j^S if there is a CLIENT ACTION 0 query to Π_i^C with input S and output $\langle C, m \rangle$, there is a SERVER ACTION 1 query to Π_j^S with input $\langle C, m \rangle$ and output $\langle \mu, k, a \rangle$, and there is a CLIENT ACTION 1 query to Π_i^C with input $\langle \mu, k, a \rangle$. Say a server instance Π_j^S is *paired with* a client instance Π_i^C if there is a CLIENT ACTION 0 query to Π_i^C with input S and output $\langle C, m \rangle$, there is a SERVER ACTION 1 query to Π_j^S with input $\langle C, m \rangle$ and output $\langle \mu, k, a \rangle$, and if there is a SERVER ACTION 2 query to Π_j^S with input s', then there was a CLIENT ACTION 1 query to Π_i^C with input $\langle \mu, k, a \rangle$ and output s'. Note that a client instance paired to a server instance will have terminated, and will be partnered with that server instance. On the other hand, a server instance may be paired to a client instance but still not have terminated. However, if it is paired to a client instance after the SERVER ACTION 2 query, then it will have terminated, and be partnered with that client instance.

We now define some events, corresponding to the adversary making a password guess against a client instance, against a server instance, and against a client instance and server instance that are partnered, respectively.

- $\mathsf{testpw}(C, i, S, \pi, \ell)$: for some m, μ, γ', k, and a, $\mathcal{A}$ makes an $H_\ell(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query, a CLIENT ACTION 0 query to a client instance Π_i^C with input S and output $\langle C, m \rangle$, a CLIENT ACTION 1 query to Π_i^C with input $\langle \mu, k, a \rangle$ and an $H_1(\langle C, \pi \rangle)$ query returning $(\gamma')^{-1}$, where the latest query is either the $H_\ell(\cdot)$ query or the CLIENT ACTION 1 query, $\sigma = DH(\alpha, \mu)$ and $m = \alpha \cdot (\gamma')^{-1}$. The associated value of this event is the output of the $H_\ell(\cdot)$ query, or the k, sk_C^i, a', or s'' value (respectively, for $\ell = 3$, 4, 5, or 6) generated by Π_i^C, whichever was set first.

- $\mathsf{testpw!}(C, i, S, \pi)$: for some k and a, a CLIENT ACTION 1 query with input $\langle \mu, k, a \rangle$ causes a $\mathsf{testpw}(C, i, S, \pi, 2)$ event to occur, with associated value k.

- $\mathsf{testpw}(S, j, C, \pi, \ell)$: for some m, μ, γ', and k, $\mathcal{A}$ makes an $H_\ell(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query, and previously made a SERVER ACTION 1 query to a server instance Π_j^S with input $\langle C, m \rangle$ and output $\langle \mu, k, a \rangle$, and an $H_1(\langle C, \pi \rangle)$ query returning $(\gamma')^{-1}$, where $\sigma = DH(\alpha, \mu)$, $m = \alpha \cdot (\gamma_1')^{-1}$, and ACCEPTABLE$(m)$. The associated value of this event is k, sk_S^j, a', or s'' (respectively, for $\ell = 3$, 4, 5, or 6) generated by Π_j^S.

[17]Note that we do this by fixing the underlying $H(\cdot)$ query. See Section 5.

- $\mathsf{testpw!}(S, j, C, \pi)$: a SERVER ACTION 2 query to Π_j^S is made with input s', and previously a $\mathsf{testpw}(S, j, C, \pi, 7)$ event occurs, with associated value s'', and for $s = s'' \oplus s'$, W the public key stored in $\pi_S[C]$, and μ returned in the associated SERVER ACTION 1 query, $\mathsf{Verify}_W(\mu, s) = 1$.

- $\mathsf{testpw}^*(S, j, C, \pi)$: $\mathsf{testpw}(S, j, C, \pi, \ell)$ occurs for some $\ell \in \{3, 4, 5, 6\}$.

- $\mathsf{testpw}(C, i, S, j, \pi)$: for some $\ell \in \{3, 4, 5, 6\}$, both a $\mathsf{testpw}(C, i, S, \pi, \ell)$ event and a $\mathsf{testpw}(S, j, C, \pi, \ell)$ event occur, where Π_i^C is paired with Π_j^S and Π_j^S is paired with Π_i^C after its SERVER ACTION 1 query.

- $\mathsf{testexecpw}(C, i, S, j, \pi)$: for some m, μ, and γ', $\mathcal{A}$ makes an $H_\ell(\langle C, S, m, \mu, \sigma, \gamma'_1\rangle)$ query, for $\ell \in \{3, 4, 5, 6\}$, and previously made an $\mathsf{Execute}$ $(C,\ i,\ S,\ j)$ query that generates m and μ, and an $H_1(\langle C, \pi\rangle)$ query returning $(\gamma')^{-1}$, where $\sigma = DH(\alpha, \mu)$, and $m = \alpha \cdot (\gamma')^{-1}$.

- $\mathsf{testpwoffline}(C, \pi)$: a $\mathsf{Corrupt}$ query has been made to a server, and an $H_\ell(\langle C, \pi\rangle)$ query, for $\ell \in \{1, 2\}$, has been made.

- $\mathsf{correctpw}$: before any $\mathsf{Corrupt}$ query, either a $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs for some C, i, and S, or a $\mathsf{testpw}^*(S, j, C, \pi_C)$ event occurs for some S, j, and C. For semi-freshness, add the following: before any $\mathsf{Corrupt}$ query to a client, a $\mathsf{testpwoffline}(C, \pi_C)$ event occurs.

- $\mathsf{correctpwexec}$: a $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event occurs for some C, i, S, and j.

- $\mathsf{forgesig}$: before any $\mathsf{Corrupt}$ query to a client, a $\mathsf{testpw!}(S, j, C, \pi_C)$ event occurs without a previous $H_2(\langle C, \pi_C\rangle)$ query occurring.

- $\mathsf{doublepwserver}$: before any $\mathsf{Corrupt}$ query, both a $\mathsf{testpw}^*(S, j, C, \pi)$ event and a $\mathsf{testpw}^*(S, j, C, \pi')$ event occur, for some S, j, C, π and π', with $\pi \neq \pi'$.

- $\mathsf{pairedpwguess}$: a $\mathsf{testpw}(C, i, S, j, \pi_C)$ event occurs, for some C, i, S, and j.

Protocol P_1. Let E_1 be the event that an m value generated in a CLIENT ACTION 0 or $\mathsf{Execute}$ query is equal to an m value generated in a previous CLIENT ACTION 0 or $\mathsf{Execute}$ query, an m value sent as input in a previous SERVER ACTION 1 query, or an m value in a previous $H_\ell(\cdot)$ query (made by the adversary), for $\ell \in \{3, 4, 5, 6\}$. Let E_2 be the event that a μ value generated in a SERVER ACTION 1 or $\mathsf{Execute}$ query is equal to a μ value generated in a previous SERVER ACTION 1 or $\mathsf{Execute}$ query, a μ value sent as input in a previous CLIENT ACTION 1 query, or a μ value in a previous $H_\ell(\cdot)$ query (made by the adversary), for $\ell \in \{3, 4, 5, 6\}$.

Let $E = E_1 \vee E_2$. Let P_1 be a protocol that is identical to P_0 except that if E occurs, the protocol aborts (and thus the adversary fails).

Claim 6.18 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathrm{ake}}_{P_0}(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{ake}}_{P_1}(\mathcal{A}) + \frac{O((n_{\mathrm{se}}+n_{\mathrm{ex}})(n_{\mathrm{ro}}+n_{\mathrm{se}}+n_{\mathrm{ex}}))}{q}.$$

Proof: Straightforward. ▮

Protocol P_2. Let P_2 be a protocol that is identical to P_1 except that Send and Execute queries are answered without making any random oracle queries, and subsequent random oracle queries by the adversary are backpatched, as much as possible, to be consistent with the responses to the Send and Execute queries.

Specifically, the queries in P_2 are changed as follows:

- In an Execute $(C,\ i,\ S,\ j)$ query, $m \leftarrow g^{\tau[i,C]}$, where $\tau[i,C] \stackrel{R}{\leftarrow} \mathbb{Z}_q$, $\mu \leftarrow g^{\tau[j,S]}$, where $\tau[j,S] \stackrel{R}{\leftarrow} \mathbb{Z}_q$, $k \stackrel{R}{\leftarrow} \{0,1\}^\kappa$, $a' \stackrel{R}{\leftarrow} \{0,1\}^\kappa$, $s'' \stackrel{R}{\leftarrow} \{0,1\}^{\kappa'}$, and $sk^i_C \leftarrow sk^j_S \stackrel{R}{\leftarrow} \{0,1\}^\kappa$.
- In a Client Action 0 query to instance Π^C_i, $m \leftarrow g^{\tau[i,C]}$, where $\tau[i,C] \stackrel{R}{\leftarrow} \mathbb{Z}_q$.
- In a Server Action 1 query to instance Π^S_j, $\mu \leftarrow g^{\tau[j,S]}$, where $\tau[j,S] \stackrel{R}{\leftarrow} \mathbb{Z}_q$, and $sk^j_S, k, a' \stackrel{R}{\leftarrow} \{0,1\}^\kappa$ and $s'' \stackrel{R}{\leftarrow} \{0,1\}^{\kappa'}$.
- In a Client Action 1 query to instance Π^C_i, do the following.
  - If this query causes a $\mathsf{testpw!}(C,i,S,\pi_C)$ event to occur, then set sk^i_C, a', s'' to associated value of the $\mathsf{testpw}(C,i,S,\pi_C,\ell)$ event, for $\ell \in \{4,5,6\}$, respectively.
  - Otherwise, if Π^C_i is paired with a server instance Π^S_j, $sk^i_C \leftarrow sk^j_S$. Then $a' \stackrel{R}{\leftarrow} \{0,1\}^\kappa$ and $s'' \stackrel{R}{\leftarrow} \{0,1\}^{\kappa'}$.
  - Otherwise, Π^C_i aborts.
- In a Server Action 2 query to instance Π^S_j, if this query causes a $\mathsf{testpw!}(S,i,C,\pi_C)$ event to occur, or if Π^S_j is paired with a client instance Π^C_i, terminate. Otherwise, Π^S_j aborts.
- In an $H_\ell(\langle C,S,m,\mu,\sigma,\gamma'\rangle)$ query, for $\ell \in \{3,4,5,6\}$, if this $H_\ell(\cdot)$ query causes a $\mathsf{testpw}(S,j,C,\pi_C,\ell)$, or $\mathsf{testexecpw}(C,i,S,j,\pi_C)$ event to occur, then output the associated value of that event, and otherwise output a random value from $\{0,1\}^\kappa$ (for $\ell \in \{3,4,5\}$) or $\{0,1\}^{\kappa'}$ (for $\ell = 6$).

Claim 6.19 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathrm{ake}}_{P_1}(\mathcal{A}) = \mathsf{Adv}^{\mathrm{ake}}_{P_2}(\mathcal{A}) + \frac{O(n_{\mathrm{ro}}+n_{\mathrm{se}})}{q}.$$

Proof: One can see that in P_1, a server instance Π_j^S in a SERVER ACTION 1 query creates a session key sk_S^j and k and a' values that are uniformly chosen from $\{0,1\}^\kappa$, and an s'' value that is uniformly chosen from $\{0,1\}^{\kappa'}$, independent of anything that previously occurred, since the $H_\ell(\cdot)$ queries that determine these values are new. Then in a SERVER ACTION 2 query, if a $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs, or Π_j^S is paired, the instance terminates, and if Π_j^S is unpaired and no $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs, then either the instance terminates or aborts. It is easy to show that the total probability of any instance terminating in this case is at most $\frac{n_{\mathrm{se}}}{2^\kappa}$

Also in P_1, for any client instance Π_i^C, either

1. a $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs, and then sk_C^i, a' and s'' are set to the values associated with the $\mathsf{testpw}(C, i, S, \pi_C, \ell)$ events, for $\ell \in \{4, 5, 6\}$, respectively, which are guaranteed to occur by our original assumption that all queries to $H_\ell(\cdot)$, for $\ell \in \{3, 4, 5, 6\}$, using the same inputs are made simultaneously, or

2. no $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs, but exactly one instance Π_j^S is paired with Π_i^C, in which case $sk_C^i = sk_S^j$, and a' is uniformly chosen from $\{0,1\}^\kappa$ and s'' is uniformly chosen from $\{0,1\}^{\kappa'}$, independent of anything that previously occurred (since no $\mathsf{testpw}(C, i, S, \pi_C, \ell)$ event, for $\ell \in \{5, 6\}$ could have occurred in this case), or

3. no $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs and no instance is paired with Π_i^C, then either the instance terminates or aborts. It is easy to show that the total probability of any instance terminating in this case is at most $\frac{n_{\mathrm{se}}}{2^\kappa}$

For any $H_3(\langle C, S, \cdot, \cdot, \cdot, \gamma' \rangle)$ query, either (1) it causes a $\mathsf{testpw}(S, j, C, \pi_C, 2)$, or $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event to occur, in which case the output is the associated value of that event (i.e., the k value associated with the particular event that occurs), (2) it does not cause a $\mathsf{testpw}(C, i, S, j, \pi_C)$ event, but does cause a $\mathsf{testpw}(C, i, S, \pi_C, \ell)$ event to occur, where the CLIENT ACTION 1 query of the event had input $\langle \mu, k, a \rangle$ for some μ, in which case either Π_i^C terminated and the output is k, or Π_i^C aborted and the output is uniformly chosen from $\{0,1\}^\kappa \setminus \{k\}$, (3) $\gamma' = H_1(\langle C, \pi_C \rangle)$, but the adversary has not made an $H_1(\langle C, \pi_C \rangle)$ query, or (4) the output of $H_3(\cdot)$ is uniformly chosen from $\{0,1\}^\kappa$, independent of anything that previously occurred, since this is a new $H_3(\cdot)$ query. The total probability of an $H_3(\cdot)$ query causing the third case above can be easily shown to be bounded by $\frac{n_{\mathrm{ro}}}{q}$. The second case above where the output is fixed can only occur when an unpaired client instance terminated with no $\mathsf{testpw!}(C, i, S, \pi_C)$ event.

For any $H_\ell(\langle C, S, \cdot, \cdot, \cdot, \gamma' \rangle)$ query, for $\ell \in \{4, 5, 6\}$, either (1) it causes a $\mathsf{testpw}(S, j, C, \pi_C, \ell)$, or $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event to occur, in which case the output is the associated value of that event, (2) it does not cause a $\mathsf{testpw}(C, i, S, j, \pi_C)$ event, but it causes a $\mathsf{testpw}(C, i, S, \pi_C, \ell)$ event to occur, where Π_i^C terminated, (3) $\gamma' = H_1(\langle C, \pi_C \rangle)$, but the adversary has not made an $H_1(\langle C, \pi_C \rangle)$ query, or (4) the output of $H_\ell(\cdot)$ is uniformly chosen from $\{0,1\}^\kappa$ (for

$\ell \in \{4,5\}$) or $\{0,1\}^{\kappa'}$ (for $\ell = 6$), independent of anything that previously occurred, since this is a new $H_\ell(\cdot)$ query. The total probability of an $H_\ell(\cdot)$ query causing the third case above can be easily shown to be bounded by $\frac{n_{\mathrm{ro}}}{q}$. The second case can only occur when an $H_3(\cdot)$ query causes a second case where its output is fixed (see above).

If an unpaired client instance Π_i^C never terminates without a $\mathsf{testpw!}(C, i, S, \pi_C)$ event, an unpaired server instance Π_j^S never terminates without a $\mathsf{testpw!}(S, j, C, \pi_C)$ event, and the third case of the $H_\ell(\cdot)$ queries, for $\ell \in \{3,4,5,6\}$ never occurs, then P_2 is consistent with P_1. The claim follows. ▮

Protocol P_3. Let P_3 be a protocol that is identical to P_2 except that in an $H_\ell(\langle C, S, \cdot, \cdot, \cdot, \cdot\rangle)$ query, for $\ell \in \{3,4,5,6\}$, there is no check for a $\mathsf{testexecpw}(C, i, S, j, \pi_C)$ event.

Claim 6.20 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + (n_{\mathrm{ro}} + n_{\mathsf{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$ such that

$$\mathsf{Adv}_{P_2}^{\mathrm{ake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_3}^{\mathrm{ake}}(\mathcal{A}) + 2\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t').$$

Proof: Let E be the event that a $\mathsf{correctpwexec}$ event occurs. Obviously, if E does not occur, then P_2 and P_3 are indistinguishable. Let ϵ be the probability that E occurs when $\mathcal{A}$ is running against protocol P_2. Then $\Pr(\mathsf{Succ}_{P_2}^{\mathrm{ake}}(\mathcal{A})) \leq \Pr(\mathsf{Succ}_{P_3}^{\mathrm{ake}}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}_{P_2}^{\mathrm{ake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_3}^{\mathrm{ake}}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts to solve CDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y), D simulates P_2 for $\mathcal{A}$ with these changes:

1. In an $\mathsf{Execute}\,(C,\ i,\ S,\ j)$ query, set $m \leftarrow Xg^{\rho_{i,C}}$ and $\mu \leftarrow Yg^{\rho'_{j,S}}$, where $\rho_{i,C}, \rho'_{j,S} \stackrel{R}{\leftarrow} \mathbb{Z}_q$.
2. When $\mathcal{A}$ finishes, for every $H_\ell(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ query, for $\ell \in \{3,4,5,6\}$, where m and μ were generated in an $\mathsf{Execute}\,(C,\ i,\ S,\ j)$ query, and an $H_1(\langle C, \pi_C\rangle)$ query returned $(\gamma')^{-1}$, add
$$\sigma X^{-\rho'_{j,S}} Y^{-\rho_{i,C}} g^{-\rho_{i,C}\rho'_{j,S}} m^{\psi_2[\pi_C]} \mu^{\psi_1[\pi_C]} g^{-\psi_1[\pi_C]\psi_2[\pi_C]}$$
to the list of possible values for $\mathsf{DH}(X, Y)$.

This simulation is perfectly indistinguishable from P_2 until E occurs, and in this case, D adds the correct $\mathsf{DH}(X, Y)$ to the list. After E occurs the simulation may be distinguishable from P_2, but this does not change the fact that E occurs with probability ϵ. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulator can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguishes the simulation from P_2.

D creates a list of size n_{ro}, and its advantage is ϵ. Let t' be the running time of D, and note that $t' = O(t + (n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$. The claim follows from the fact that $\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(D) \leq \mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', n_{\mathrm{ro}})$. ▮

Protocol P_4. Let P_4 be a protocol that is identical to P_3 except that if $\mathsf{correctpw}$ occurs then the protocol halts and the adversary automatically succeeds. Note that this involves the following changes:

1. In a Client Action 1 query to Π_i^C, if a $\mathsf{testpw!}(C, i, S, \pi_C)$ event occurs and no $\mathsf{Corrupt}$ query has been made, halt and say the adversary automatically succeeds.
2. In an $H_\ell(\cdot)$ query, for $\ell \in \{3, 4, 5, 6\}$, if a $\mathsf{testpw}^*(S, j, C, \pi_C)$ event occurs and no $\mathsf{Corrupt}$ query has been made, halt and say the adversary automatically succeeds.
3. For semi-freshness, if a $\mathsf{testpwoffline}(C, S, \pi_C)$ event occurs (this must be tested in $\mathsf{Corrupt}$ queries, $H_1(\cdot)$ queries, and $H_2(\cdot)$ queries) and no $\mathsf{Corrupt}$ query to a client has been made, halt and say the adversary automatically succeeds.

Claim 6.21 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}_{P_3}^{\mathrm{ake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_4}^{\mathrm{ake}}(\mathcal{A}).$$

Proof: Obvious. ▮

Note that in P_4, until $\mathsf{correctpw}$ or a $\mathsf{Corrupt}$ query occurs, no unpaired client or server instance will terminate. Also, for semifreshness, until $\mathsf{correctpw}$ or a $\mathsf{Corrupt}$ query to a client occurs, no unpaired server instance will terminate.

Protocol P_5. Let P_5 be a protocol that is identical to P_4 except that if a $\mathsf{pairedpwguess}$ event occurs, the protocol halts and the adversary fails. We assume that when a query is made, the test for $\mathsf{pairedpwguess}$ occurs before the test for $\mathsf{correctpw}$. Note that this involves the following change: if a $\mathsf{testpw}(C, i, S, \pi, \ell)$ event occurs (this should be checked in a Client Action 1 query, or an $H_\ell(\cdot)$ query) for $\ell \in \{3, 4, 5, 6\}$, check if a $\mathsf{testpw}(C, i, S, j, \pi)$ event also occurs.

Claim 6.22 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + (n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$ such that

$$\mathsf{Adv}_{P_4}^{\mathrm{ake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_5}^{\mathrm{ake}}(\mathcal{A}) + 2n_{\mathrm{se}} \cdot \mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', n_{\mathrm{ro}}).$$

Proof: Obviously, if $\mathsf{pairedpwguess}$ does not occur, then P_4 and P_5 are indistinguishable. Let ϵ be the probability that $\mathsf{pairedpwguess}$ occurs when $\mathcal{A}$ is running against protocol P_4. Then $\Pr(\mathsf{Succ}_{P_4}^{\mathrm{ake}}(\mathcal{A})) \leq \Pr(\mathsf{Succ}_{P_5}^{\mathrm{ake}}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}_{P_4}^{\mathrm{ake}}(\mathcal{A}) \leq \mathsf{Adv}_{P_5}^{\mathrm{ake}}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts to solve CDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y), D chooses a random $d \in \{1, \ldots, n_{\mathrm{se}}\}$ and simulates P_4 for $\mathcal{A}$ with these changes:

1. In the dth Client Action 0 query, say to a client instance $\Pi_{i'}^C$, with input S, set $m \leftarrow X$.

2. In a Server Action 1 query to a server instance Π_j^S with input $\langle C, m\rangle$ where there was a Client Action 0 query to $\Pi_{i'}^C$ (i.e., the instance with the dth Client Action 0 query) with input S and output $\langle C, m\rangle$, set $\mu \leftarrow Y g^{\rho'_{j,S}}$.

3. In a Client Action 1 query to $\Pi_{i'}^C$, if $\Pi_{i'}^C$ is unpaired, D outputs 0 and halts.

4. In a Server Action 2 query to Π_j^S, if Π_j^S was paired with $\Pi_{i'}^C$ after its Server Action 1 query, but is not now paired with $\Pi_{i'}^C$, no test for $\mathsf{correctpw}$ is made, and Π_j^S aborts.

5. When $\mathcal{A}$ finishes, for every $H_\ell(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ query, for $\ell \in \{3, 4, 5, 6\}$ where m and μ were generated by $\Pi_{i'}^C$ and a server instance Π_j^S, respectively, where Π_j^S was paired with $\Pi_{i'}^C$ after its Server Action 1 query, and an $H_1(\pi)$ query returned $(\gamma')^{-1}$, add

$$\sigma X^{-\rho'_{j,S}} \mu^{\psi_1[\pi]}$$

to the list of possible values for $\mathsf{DH}(X, Y)$.

This simulation is perfectly indistinguishable from P_4 until (1) a $\mathsf{testpw}(S, j, C, \pi)$ event occurs, where Π_j^S was paired with $\Pi_{i'}^C$ after the Server Action 1 query, or (2) $\Pi_{i'}^C$ is not paired with a server instance when the Client Action 1 query is made. Note that the probability of a $\mathsf{pairedpwguess}$ event occurring for $\Pi_{i'}^C$ is at least $\frac{\epsilon}{n_{\mathrm{se}}}$, and this is at most the probability of an event of type (1) occurring, since an event of type (2) implies that $\mathsf{pairedpwguess}$ would never have occurred in P_4 for $\Pi_{i'}^C$ (see P_1). If an event of type (1) occurs, D adds the correct $\mathsf{DH}(X, Y)$ to the list.

Note that in either case, the simulation may be distinguishable from P_4, but this does not change the fact that a $\mathsf{pairedpwguess}$ event will occur for $\Pi_{i'}^C$ with probability at least $\frac{\epsilon}{n_{\mathrm{se}}}$ in the simulation. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulator can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguishes the simulation from P_4.

D creates a list of size n_{ro}, and its advantage is $\frac{\epsilon}{n_{\mathrm{se}}}$. Let t' be the running time of D, and note that $t' = O(t + (n_{\mathrm{ro}} + n_{\mathrm{se}} + n_{\mathrm{ex}}) t_{\mathsf{exp}})$. The claim follows from the fact that $\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(D) \leq \mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', n_{\mathrm{ro}})$. ▮

Protocol P_6. Let P_6 be a protocol that is identical to P_4 except that in a Server Action 2 query, if this query causes a $\mathsf{testpw!}(S, i, C, \pi_C)$ event to occur, but there was no $H_2(\langle C, \pi_C\rangle)$ query, the $\mathsf{testpw!}(S, i, C, \pi_C)$ event is ignored (i.e., the server instance aborts).

Claim 6.23 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t+(n_{\rm ro}+n_{\rm se}+n_{\rm ex})t_{\mathsf{exp}})$ such that

$$\mathsf{Adv}^{\rm ake}_{P_6}(\mathcal{A}) \le \mathsf{Adv}^{\rm ake}_{P_4}(\mathcal{A}) + 2n_{\rm se} \cdot \mathsf{Succ}^{\rm eu\text{-}cma}_{\mathcal{S},\kappa}(t', n_{\rm se}) + \frac{O(n_{\rm ro})}{q}.$$

Proof: Let E be the event that a $\mathsf{forgesig}$ event occurs. Obviously, if E does not occur, then P_6 and P_5 are indistinguishable. Let ϵ be the probability that E occurs when $\mathcal{A}$ is running against protocol P_5. Then $\Pr(\mathsf{Succ}^{\rm ake}_{P_6}(\mathcal{A})) \le \Pr(\mathsf{Succ}^{\rm ake}_{P_5}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}^{\rm ake}_{P_6}(\mathcal{A}) \le \mathsf{Adv}^{\rm ake}_{P_5}(\mathcal{A}) + 2\epsilon$.

Now we construct a forger F for the signature scheme $\mathcal{S}$ by running $\mathcal{A}$ on a simulation of the protocol. Given public key pk and a signature oracle for pk, F chooses a random $d \in \{1, \ldots, n_{\rm se}\}$ and simulates P_5 for $\mathcal{A}$ with these changes:

1. In the dth Server Action 1 query, say to a server instance $\Pi^{S'}_{j'}$ with input $\langle C', m\rangle$, if this is the first Server Action 1 query for pair (C', S'), the value W in $\pi_{S'}[C']$ is (permanently) changed to pk, otherwise F halts and fails.
2. If an $H_2(\langle C', \pi_{C'}\rangle)$ query occurs, F halts and fails.
3. In a Client Action 1 query to an unpaired instance $\Pi^{C'}_i$, whose corresponding Client Action 0 query had input S' and for which $\Pi^{C'}_i$ does not abort, compute s using the signature oracle for pk.
4. In an Server Action 2 query to an instance $\Pi^{S'}_j$ (note this could be any instance of S') if a $\mathsf{testpw!}(S', i, C', \pi_{C'})$ event occurs, F outputs (μ, s) and halts (for values μ and s computed by $\Pi^{S'}_j$).
5. If $\mathcal{A}$ finishes, F halts and fails.

Note that the simulation is indistinguishable from P_5 until F outputs a forged signature or the dth Server Action 1 query does not correspond to the first Server Action 1 query for pair (C, S) corresponding to a $\mathsf{forgesig}$ event. Thus F will produce a forged signature with probability at least $\frac{\epsilon}{n_{\rm se}}$.

Let t' be the running time of F, and note that $t' = O(t + (n_{\rm ro} + n_{\rm se} + n_{\rm ex})t_{\mathsf{exp}})$. The success probability of F is

$$\begin{aligned}\mathsf{Succ}^{\rm eu\text{-}cma}_{\mathcal{S},\kappa}(F) &= \Pr[F \text{ outputs valid signature}] \\ &\ge \frac{\epsilon}{n_{\rm se}}.\end{aligned}$$

The claim follows from the fact that $\mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(F) \le \mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(t', n_{\text{se}})$. ∎

Protocol P_7. Let E be the event that both a doublepwserver event occurs. Let P_7 be a protocol that is identical to P_6 except that if doublepwserver occurs, the protocol halts and the adversary fails. We assume that when a query is made, the test for doublepwserver occurs before the test for pairedpwguess or correctpw.

Claim 6.24 For any adversary $\mathcal{A}$ running in time t, there is a $t' = O(t + (n_{\text{ro}} + n_{\text{se}} + n_{\text{ex}})t_{\mathsf{exp}})$ such that

$$\mathsf{Adv}^{\text{ake}}_{P_6}(\mathcal{A}) \le \mathsf{Adv}^{\text{ake}}_{P_7}(\mathcal{A}) + 2\mathsf{Adv}^{\text{CDH}}_{G_q}(t', (n_{\text{ro}})^2).$$

Proof: Let ϵ be the probability that doublepwserver occurs when $\mathcal{A}$ is running against protocol P_6. Then $\Pr(\mathsf{Succ}^{\text{ake}}_{P_6}(\mathcal{A})) \le \Pr(\mathsf{Succ}^{\text{ake}}_{P_7}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}^{\text{ake}}_{P_5}(\mathcal{A}) \le \mathsf{Adv}^{\text{ake}}_{P_7}(\mathcal{A}) + 2\epsilon$.

Now we construct an algorithm D that attempts to solve CDH by running $\mathcal{A}$ on a simulation of the protocol. Given (X, Y), D simulates P_6 for $\mathcal{A}$ with these changes:

1. In an $H_1(\langle C, \pi\rangle)$ query, output $X^{\psi_1[C,\pi]} g^{\psi_1'[C,\pi]}$, where $\psi_1[C,\pi] \xleftarrow{R} \{0,1\}$ and $\psi_1'[C,\pi] \xleftarrow{R} \mathbb{Z}_q$.
2. In a Server Action 1 query to a server instance Π^S_j with input $\langle C, m\rangle$ where $\textsc{acceptable}(m)$ is true, set $\mu \leftarrow Y g^{\rho'_{j,S}}$.
3. Tests for correctpw (from P_4) and pairedpwguess (from P_5) are not made. In particular, unpaired client instances that receive a Client Action 1 query abort, and unpaired server instances that receive a Server Action 2 query abort. Also, $H_\ell(\cdot)$ queries always return values uniformly chosen from $\{0,1\}^\kappa$.
4. When $\mathcal{A}$ finishes, for every pair of queries $H_\ell(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ and $H_{\hat{\ell}}(\langle C, S, m, \mu, \hat{\sigma}, \hat{\gamma}'\rangle)$, for $\ell, \hat{\ell} \in \{3,4,5,6\}$, where $\textsc{acceptable}(\sigma)$ and $\textsc{acceptable}(\hat{\sigma})$ are true, there was a Server Action 1 query to a server instance Π^S_j with input $\langle C, m\rangle$ and output $\langle \mu, k, a\rangle$ (and thus $\textsc{acceptable}(m)$ is true), an $H_1(\langle C, \pi\rangle)$ query that returned $(\gamma')^{-1}$, an $H_1(\langle C, \hat{\pi}\rangle)$ query that returned $(\hat{\gamma}')^{-1}$, and $\psi_1[\pi] \ne \psi_1[\hat{\pi}]$, add

   $$\left(\sigma\hat{\sigma}^{-1}(\gamma')^{\rho'_{j,S}}(\hat{\gamma}')^{-\rho'_{j,S}} Y^{\psi_1'[\pi]-\psi_1'[\hat{\pi}]}\right)^{\psi_1[\hat{\pi}]-\psi_1[\pi]}$$

   to the list of possible values for $\mathsf{DH}(X, Y)$.

This simulation is perfectly indistinguishable from P_6 until a doublepwserver event, a pairedpwguess event, or a correctpw event occurs, or $\mathcal{A}$ makes a Corrupt query. If a doublepwserver event occurs, then with probability $\frac{1}{2}$ it occurs for two passwords π and $\hat{\pi}$ with

$\psi_1[\pi] \neq \psi_1[\hat{\pi}]$, and in this case D adds the correct $\mathsf{DH}(X,Y)$ to the list. If a correctpw event or a pairedpwguess event occurs, then the doublepwserver event would never have occurred in P_6, since P_6 would halt. Also, if a Corrupt query is made before a doublepwserver event, then a doublepwserver event would never occur (by definition). Note that in either of these cases, the simulation may be distinguishable from P_6, but this does not change the fact that a doublepwserver event will occur with probability at least ϵ in the simulation. However, we do make the assumption that $\mathcal{A}$ still follows the appropriate time and query bounds (or at least that the simulator can stop $\mathcal{A}$ from exceeding these bounds), even if $\mathcal{A}$ distinguishes the simulation from P_6.

D creates a list of size $(n_{\rm ro})^2$, and its advantage is $\frac{\epsilon}{2}$. Let t' be the running time of D, and note that $t' = O(t + ((n_{\rm ro})^2 + n_{\rm se} + n_{\rm ex})t_{\mathsf{exp}})$. The claim follows from the fact that $\mathsf{Adv}^{\rm CDH}_{G_q}(D) \leq \mathsf{Adv}^{\rm CDH}_{G_q}(t', (n_{\rm ro})^2)$. ▮

Protocol P_8. Let P_8 be a protocol that is identical to P_7 except that there is a new internal oracle (i.e., not available to the adversary) that handles passwords, called a *password oracle*. This oracle generates all passwords during initialization. Then it accepts queries of the form $\mathsf{testpw}(C, \pi)$ and returns TRUE if $\pi = \pi_C$, and FALSE otherwise. It also accepts Corrupt (U) queries and returns $\langle \pi_U[C] \rangle_{C \in \mathit{Clients}}$ if $U \in \mathit{Servers}$, and otherwise returns π_U. When a Corrupt (U) query is received in the protocol, it is answered using a Corrupt (U) query to the password oracle. The protocol is also changed in the method for determining correctpw. Specifically, to test if correctpw occurs, whenever the first $\mathsf{testpw}(C, i, S, \pi)$ event occurs for an instance Π^C_i and password π, or the first $\mathsf{testpw}(S, j, C, \pi)$ event occurs for an instance Π^S_j and password π, or, for semifreshness, a $\mathsf{testpwoffline}(C, S, \pi)$ event occurs, a $\mathsf{testpw}(C, \pi)$ query is made to the password oracle to see if $\pi = \pi_C$.

Claim 6.25 For any adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\rm ake}_{P_7}(\mathcal{A}) = \mathsf{Adv}^{\rm ake}_{P_8}(\mathcal{A}).$$

Proof: By inspection, P_7 and P_8 are perfectly indistinguishable. ▮

The probability of the adversary $\mathcal{A}$ succeeding in P_8 is bounded by

$$\Pr(\mathsf{Succ}^{\rm ake}_{P_8}(\mathcal{A})) \leq \Pr(\mathsf{correctpw}) + \Pr(\mathsf{Succ}^{\rm ake}_{P_8}(\mathcal{A}) | \neg \mathsf{correctpw}).$$

First, if we only consider freshness, and not semifreshness, there are at most $n_{\rm se}$ queries to the password oracle before a Corrupt query, and passwords are chosen uniformly from a dictionary of size N, so $\Pr(\mathsf{correctpw}) \leq \frac{n_{\rm se}}{N}$. If we also consider semifreshness, then there may be $n_{\rm ro}$ more queries to the password oracle before a Corrupt query to a client, but these only occur if there has also been a Corrupt query to a server, so $\Pr(\mathsf{correctpw}) \leq \frac{n_{\rm se} + n_{\rm ro}[n_{\rm co} > 0]}{N}$.

Now we compute $\Pr(\mathsf{Succ}^{\rm ake}_{P_8}(\mathcal{A}) | \neg \mathsf{correctpw})$. If correctpw does not occur, then $\mathcal{A}$ succeeds by making a Test query to a fresh (or semifresh, if desired) instance Π^U_i and guessing the bit used in that Test query. We will show that the view of the adversary is independent of sk_U^i, and thus the probability of success is exactly $\frac{1}{2}$.

First we examine Reveal queries. Recall that since Π_i^U is fresh, there could be no $\mathsf{Reveal}\ (U,\ i)$ query, and if $\Pi_j^{U'}$ is partnered with Π_i^U, no $\mathsf{Reveal}\ (U',\ j)$ query. Second note that since sid includes m and μ values, if more than a single client instance and a single server instance accept with the same sid, $\mathcal{A}$ fails (see P_1). Thus the output of Reveal queries is independent of sk_U^i.

Second we examine $H_4(\cdot)$ queries. As noted in the discussion following the description of P_4, until a $\mathsf{correctpw}$ event or a $\mathsf{Corrupt}$ query, no unpaired client or server instance will terminate, and for semifreshness, until a $\mathsf{correctpw}$ event or a $\mathsf{Corrupt}$ query to a client occurs, no unpaired server instance will terminate, and thus an instance may only be fresh (or semi-fresh) and receive a Test query if it is partnered. However, an $H_4(\cdot)$ query will never reveal sk_U^i if Π_i^U is partnered (see P_5).

This implies that the view of the adversary is independent of sk_U^i, and thus the probability of success is exactly $\frac{1}{2}$.

Since $\Pr(\neg\mathsf{correctpw}) = 1 - \Pr(\mathsf{correctpw})$, we have that

$$\begin{aligned}
\Pr(\mathsf{Succ}_{P_8}^{\mathrm{ake}}(\mathcal{A})) &\leq \Pr(\mathsf{correctpw}) + \Pr(\mathsf{Succ}_{P_8}^{\mathrm{ake}}(\mathcal{A})|\neg\mathsf{correctpw})(1 - \Pr(\mathsf{correctpw})) \\
&\leq \Pr(\mathsf{correctpw}) + \frac{1}{2}(1 - \Pr(\mathsf{correctpw})) \\
&= \frac{1}{2} + \frac{\Pr(\mathsf{correctpw})}{2}.
\end{aligned}$$

Therefore $\mathsf{Adv}_{P_7}^{\mathrm{ake}}(\mathcal{A}) \leq \Pr(\mathsf{correctpw})$. The theorem follows from this, the bound on $\Pr(\mathsf{correctpw})$ from above, for freshness and semifreshness, and Claims 6.18 through 6.25. ▮

References

[1] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *CRYPTO '98* (LNCS 1462), pp. 26–45, 1998.

[2] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT 2000* (LNCS 1807), pp. 139–155, 2000.

[3] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In 1st *ACM Conference on Computer and Communications Security*, pages 62–73, November 1993.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *CRYPTO '93* (LNCS 773), pp. 232–249, 1993.

[5] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *EUROCRYPT '94* (LNCS 950), pp. 92–111, 1995.

[6] M. Bellare and P. Rogaway. Provably secure session key distribution—the three party case. In *27th ACM Symposium on the Theory of Computing*, pp. 57–66, 1995.

[7] S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symposium on Research in Security and Privacy*, pages 72–84, 1992.

[8] S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *ACM Conference on Computer and Communications Security*, pp. 244–250, 1993.

[9] S. M. Bellovin and M. Merritt. Cryptographic Protocol for Secure Communications. U.S. Patent 5,241,599.

[10] S. M. Bellovin and M. Merritt. Cryptographic Protocol for Remote Authentication. U.S. Patent 5,440,635.

[11] D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium* (LNCS 1423), pp. 48–63, 1998.

[12] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password authentication and key exchange using Diffie-Hellman. In *EUROCRYPT 2000* (LNCS 1807), pp. 156–171, 2000.

[13] R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *CRYPTO '98* (LNCS 1462), pp. 13–25, 1998.

[14] T. Dierks and C. Allen. The TLS protocol, version 1.0, IETF RFC 2246, January 1999.

[15] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22(6):644–654, 1976.

[16] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Trans. Info. Theory*, 31:469–472, 1985.

[17] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *CRYPTO '99* (LNCS 1666), pp. 537–554, 1999.

[18] O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only. In *CRYPTO 2001* (LNCS 2139), pp. 408–432, 2001.

[19] O. Goldreich, S. Micali, and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In *19th ACM Symposium on the Theory of Computing*, pp. 218–229, 1987.

[20] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences* 28:270–299, 1984.

[21] S. Goldwasser, S. Micali, and R. L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing* 17(2):281–308, April 1988.

[22] L. Gong. Optimal authentication protocols resistant to password guessing attacks. In *8th IEEE Computer Security Foundations Workshop*, pages 24–29, 1995.

[23] L. Gong, T. M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, June 1993.

[24] D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communication Review, ACM SIGCOMM*, 26(5):5–20, 1996.

[25] D. Jablon. Extended password key exchange protocols immune to dictionary attack. In *WETICE'97 Workshop on Enterprise Security*, 1997.

[26] D. Jablon Password authentication using multiple servers. In em RSA Conference 2001, Cryptographers' Track (LNCS 2020), pp. 344–360, 2001.

[27] D. Jablon Cryptographic methods for remote authentication. U.S. Patent 6,226,383.

[28] J. Katz, R. Ostrovsky, and M. Yung. Practical password-authenticated key exchange provably secure under standard assumptions. In *Eurocrypt 2001* (LNCS 2045), pp. 475–494, 2001.

[29] T. Kwon. Authentication and Key Agreement via Memorable Passwords. In *2001 Internet Society Network and Distributed System Security Symposium*, 2001.

[30] A. Lenstra and E. Verheul. The XTR public key system. In CRYPTO2000, pages 1–18.

[31] S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Proceedings of the Workshop on Security Protocols*, 1997.

[32] P. MacKenzie, S. Patel, and R. Swaminathan. Password authenticated key exchange based on RSA. In *ASIACRYPT 2000* (LNCS 1976), pp. 599–613, 2000.

[33] P. MacKenzie. More Efficient Password-Authenticated Key Exchange, RSA Conference, Cryptographer's Track (LNCS 2020), pp. 361–377, 2001.

[34] S. Patel. Number theoretic attacks on secure password schemes. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 236–247, 1997.

[35] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *EUROCRYPT '96* (LNCS 1070), pages 387–398, 1996.

[36] M. Roe, B. Christianson, and D. Wheeler. Secure sessions from weak secrets. Technical report, University of Cambridge and University of Hertfordshire, 1998.

[37] C. P. Schnorr. Efficient identification and signatures for smart cards. In *Crypto'89* (LNCS 435), pp. 235–251, 1990.

[38] V. Shoup. On formal models for secure key exchange (version 4). IBM Research Report RZ3120, 1999. Available at: `http://eprint.iacr.org/1999/012/`

[39] SSH Communications Security. `http://www.ssh.fi`, 2001.

[40] M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of encrypted key exchange. *ACM Operating System Review*, 29:22–30, 1995.

[41] T. Wu. The secure remote password protocol. In *1998 Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.

[42] T. Wu. A real-world analysis of Kerberos password security. In *Proceedings of the 1999 Network and Distributed System Security Symposium*, February 1999.

A Proof models

In the simulation model [12], one considers an ideal world in which a trusted host generates session keys for parties, and answers simple password guesses of the form "Is the password between Alice and Bob equal to X?" To prove security of a protocol, one must show that one can construct a simulation of the protocol using the passwords and session keys from the ideal world, such that the simulation is indistinguishable from the real protocol (using real passwords and keys). As is customary, it is assumed that the adversary controls the network, and may forward, modify, delete, insert, etc. messages to and from parties as she wishes.

In the direct model [2], one defines the security of authenticated key exchange directly, and proves the security of the protocol by proving that the advantage obtained in distinguishing real session keys from random keys is directly related (within a negligible factor) to how many protocol sessions have been actively attacked (and thus how many password could have been tested) divided by the number of possible passwords. (Note that concurrent protocol sessions are allowed in both models.)

Although there have been some equivalences shown between versions of the simulation model and direct model for authenticated key exchange using cryptographically strong keys (see [38] and [4]), no relation between the simulation model and direct model for password-authenticated key exchange has yet been shown. Both models seem to be sufficient for proving security. The simulation model follows the more foundational tradition of secure multi-party computation [19] and would most likely allow easier proofs of security for, e.g., the composition of protocols, while the direct model seems to allow more straightforward proofs, and by not requiring explicit knowledge of an attacker's password guesses in the

$sid \leftarrow pid \leftarrow sk \leftarrow \varepsilon \quad acc \leftarrow term \leftarrow \text{FALSE}$

if $state = \text{READY}$ **and** $U \in Clients$ **then** {CLIENT ACTION 0}

  $\langle C \rangle \leftarrow U$

  $\langle S \rangle \leftarrow msg\text{-}in$ where $S \in Servers$

  $x \xleftarrow{R} \mathbb{Z}_q \quad \alpha \leftarrow g^x$

  $\gamma_1 \leftarrow H_1(\pi) \quad \gamma_2 \leftarrow H_2(\pi)$

  $m \leftarrow \alpha \cdot \gamma_1 \quad state \leftarrow \langle S, m, x, \gamma_1, \gamma_2 \rangle \quad msg\text{-}out \leftarrow \langle C, m \rangle$

  return $(msg\text{-}out, acc, term, sid, pid, sk, state)$

elseif $state = \text{READY}$ **and** $U \in Servers$ **then** {SERVER ACTION 1}

  $\langle S \rangle \leftarrow U$

  $\langle C, m \rangle \leftarrow msg\text{-}in$ where $C \in Clients$ and $\text{ACCEPTABLE}(m)$

  $y \xleftarrow{R} \mathbb{Z}_q \quad \beta \leftarrow g^y$

  $\langle \gamma_1', \gamma_2 \rangle \leftarrow \pi_S[C]$

  $\alpha \leftarrow m \cdot \gamma_1'$

  $\mu \leftarrow \beta \cdot \gamma_2 \quad \sigma \leftarrow \alpha^y$

  $state \leftarrow \text{DONE} \quad msg\text{-}out \leftarrow \langle \mu \rangle$

  $sid \leftarrow C \,\|\, S \,\|\, m \,\|\, \mu \quad pid \leftarrow C$

  $sk \leftarrow H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle) \quad acc \leftarrow term \leftarrow \text{TRUE}$

  return $(msg\text{-}out, acc, term, sid, pid, sk, state)$

elseif $state = \langle S, m, x, \gamma_1, \gamma_2 \rangle$ **and** $U \in Clients$ **then** {CLIENT ACTION 1}

  $\langle \mu \rangle \leftarrow msg\text{-}in$ where $\text{ACCEPTABLE}(\mu)$

  $\beta \leftarrow \mu \cdot (\gamma_2)^{-1}$

  $\sigma \leftarrow \beta^x \quad \gamma_1' \leftarrow (\gamma_1)^{-1}$

  $state \leftarrow \text{DONE} \quad msg\text{-}out \leftarrow \varepsilon$

  $sid \leftarrow C \,\|\, S \,\|\, m \,\|\, \mu \quad pid \leftarrow S$

  $sk \leftarrow H_3(\langle C, S, m, \mu, \sigma, \gamma_1' \rangle) \quad acc \leftarrow term \leftarrow \text{TRUE}$

  return $(msg\text{-}out, acc, term, sid, pid, sk, state)$

Figure 10: Specification of the PPK protocol

proofs, may allow more protocols to be proven secure. That is, the simulation model may provide an overly stringent definition of security. In any event, for the discussion below, we will assume both models are acceptable, and not worry about which of the two models is used.

B Formal Specification of the Protocols

See Figures 10, 11, and 12.

C Forward secrecy

In trying to prove PPK is secure in the sense of forward secrecy, the difficult case in the proof is the following: Say Alice and Bob share a password. The adversary pretends to be

$$
\begin{array}{ll}
\mathit{sid} \leftarrow \mathit{pid} \leftarrow \mathit{sk} \leftarrow \varepsilon \quad \mathit{acc} \leftarrow \mathit{term} \leftarrow \text{FALSE} & \\
\textbf{if } \mathit{state} = \text{READY} \textbf{ and } U \in \mathit{Clients} \textbf{ then} & \{\text{Client Action 0}\} \\
\quad \langle C \rangle \leftarrow U & \\
\quad \langle S \rangle \leftarrow \mathit{msg\text{-}in} \text{ where } S \in \mathit{Servers} & \\
\quad x \stackrel{R}{\leftarrow} \mathbb{Z}_q \quad \alpha \leftarrow g^x & \\
\quad \gamma \leftarrow H_1(\pi) & \\
\quad m \leftarrow \alpha \cdot \gamma_1 \quad \mathit{state} \leftarrow \langle S, m, x, \gamma \rangle \quad \mathit{msg\text{-}out} \leftarrow \langle C, m \rangle & \\
\quad \textbf{return } (\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state}) & \\
\textbf{elseif } \mathit{state} = \text{READY} \textbf{ and } U \in \mathit{Servers} \textbf{ then} & \{\text{Server Action 1}\} \\
\quad \langle S \rangle \leftarrow U & \\
\quad \langle C, m \rangle \leftarrow \mathit{msg\text{-}in} \text{ where } C \in \mathit{Clients} \text{ and ACCEPTABLE}(m) & \\
\quad y \stackrel{R}{\leftarrow} \mathbb{Z}_q \quad \mu \leftarrow g^y & \\
\quad \gamma' \leftarrow \pi_S[C] & \\
\quad \alpha \leftarrow m \cdot \gamma' & \\
\quad \sigma \leftarrow \alpha^y & \\
\quad k \leftarrow H_2(\langle C, S, m, \mu, \sigma, \gamma' \rangle) & \\
\quad k'' \leftarrow H_3(\langle C, S, m, \mu, \sigma, \gamma' \rangle) & \\
\quad \mathit{sk} \leftarrow H_4(\langle C, S, m, \mu, \sigma, \gamma' \rangle) & \\
\quad \mathit{sid} \leftarrow C \parallel S \parallel m \parallel \mu \quad \mathit{pid} \leftarrow C & \\
\quad \mathit{acc} \leftarrow \text{TRUE} \quad \mathit{state} \leftarrow \langle k'' \rangle \quad \mathit{msg\text{-}out} \leftarrow \langle \mu, k \rangle & \\
\quad \textbf{return } (\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state}) & \\
\textbf{elseif } \mathit{state} = \langle S, m, x, \gamma \rangle \textbf{ and } U \in \mathit{Clients} \textbf{ then} & \{\text{Client Action 1}\} \\
\quad \langle \mu, k \rangle \leftarrow \mathit{msg\text{-}in} \text{ where ACCEPTABLE}(\mu) & \\
\quad \sigma \leftarrow \mu^x \quad \gamma' \leftarrow \gamma^{-1} & \\
\quad \mathit{state} \leftarrow \text{DONE} \quad \mathit{msg\text{-}out} \leftarrow \varepsilon & \\
\quad \textbf{if } k = H_2(\langle C, S, m, \mu, \sigma, \gamma' \rangle) \textbf{ then} & \\
\qquad k' \leftarrow H_3(\langle C, S, m, \mu, \sigma, \gamma' \rangle) & \\
\qquad \mathit{acc} \leftarrow \mathit{term} \leftarrow \text{TRUE} \quad \mathit{msg\text{-}out} \leftarrow \langle k' \rangle & \\
\qquad \mathit{sid} \leftarrow C \parallel S \parallel m \parallel \mu \quad \mathit{pid} \leftarrow S & \\
\qquad \mathit{sk} \leftarrow H_4(\langle C, S, m, \mu, \sigma, \gamma' \rangle) & \\
\quad \textbf{return } (\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state}) & \\
\textbf{elseif } \mathit{state} = \langle k'' \rangle \textbf{ and } U \in \mathit{Servers} \textbf{ then} & \{\text{Server Action 2}\} \\
\quad \mathit{state} \leftarrow \text{DONE} \quad \mathit{msg\text{-}out} \leftarrow \varepsilon & \\
\quad \langle k' \rangle \leftarrow \mathit{msg\text{-}in} & \\
\quad \textbf{if } k' = k'' \textbf{ then} & \\
\qquad \mathit{term} \leftarrow \text{TRUE} & \\
\quad \textbf{return } (\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state}) &
\end{array}
$$

Figure 11: Specification of the PAK protocol

$sid \leftarrow pid \leftarrow sk \leftarrow \varepsilon \quad acc \leftarrow term \leftarrow \text{FALSE}$

if $state = \text{READY}$ **and** $U \in Clients$ **then** $\{\text{Client Action 0}\}$

$\quad \langle C \rangle \leftarrow U$

$\quad \langle S \rangle \leftarrow msg\text{-}in$ where $S \in Servers$

$\quad x \stackrel{R}{\leftarrow} \mathbb{Z}_q \quad \alpha \leftarrow g^x$

$\quad \gamma \leftarrow H_1(\langle C, \pi \rangle)$

$\quad m \leftarrow \alpha \cdot \gamma \quad state \leftarrow \langle S, m, x, \gamma, \pi \rangle \quad msg\text{-}out \leftarrow \langle C, m \rangle$

$\quad$ **return** $(msg\text{-}out, acc, term, sid, pid, sk, state)$

elseif $state = \text{READY}$ **and** $U \in Servers$ **then** $\{\text{Server Action 1}\}$

$\quad \langle S \rangle \leftarrow U$

$\quad \langle C, m \rangle \leftarrow msg\text{-}in$ where $C \in Clients$ and $\text{ACCEPTABLE}(m)$

$\quad y \stackrel{R}{\leftarrow} \mathbb{Z}_q \quad \mu \leftarrow g^y$

$\quad \langle \gamma', W, V' \rangle \leftarrow \pi_S[C]$

$\quad \alpha \leftarrow m \cdot \gamma'$

$\quad \sigma \leftarrow \alpha^y$

$\quad k \leftarrow H_3(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$

$\quad a' \leftarrow H_5(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$

$\quad a \leftarrow a' \oplus V$

$\quad s'' \leftarrow H_6(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$

$\quad sk \leftarrow H_4(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$

$\quad sid \leftarrow C \parallel S \parallel m \parallel \mu \quad pid \leftarrow C$

$\quad acc \leftarrow \text{TRUE} \quad state \leftarrow \langle s'' \rangle \quad msg\text{-}out \leftarrow \langle \mu, k, a \rangle$

$\quad$ **return** $(msg\text{-}out, acc, term, sid, pid, sk, state)$

elseif $state = \langle S, m, x, \gamma, \pi \rangle$ **and** $U \in Clients$ **then** $\{\text{Client Action 1}\}$

$\quad \langle \mu, k, a \rangle \leftarrow msg\text{-}in$ where $\text{ACCEPTABLE}(\mu)$

$\quad \sigma \leftarrow \mu^x \quad \gamma' \leftarrow \gamma^{-1}$

$\quad state \leftarrow \text{DONE} \quad msg\text{-}out \leftarrow \varepsilon$

$\quad$ **if** $k = H_3(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ **then**

$\quad\quad a' \leftarrow H_5(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$

$\quad\quad V' \leftarrow a' \oplus a \quad V \leftarrow H_2(\langle C, \pi \rangle) \oplus V'$

$\quad\quad$ **if** $\text{VALID}(V)$ **then**

$\quad\quad\quad s \leftarrow \mathsf{sig}_V(\mu) \quad s'' \leftarrow H_6(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$

$\quad\quad\quad s' \leftarrow s'' \oplus s$

$\quad\quad\quad acc \leftarrow term \leftarrow \text{TRUE} \quad msg\text{-}out \leftarrow \langle s' \rangle$

$\quad\quad\quad sid \leftarrow C \parallel S \parallel m \parallel \mu \quad pid \leftarrow S$

$\quad\quad\quad sk \leftarrow H_4(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$

$\quad$ **return** $(msg\text{-}out, acc, term, sid, pid, sk, state)$

elseif $state = \langle s'' \rangle$ **and** $U \in Servers$ **then** $\{\text{Server Action 2}\}$

$\quad state \leftarrow \text{DONE} \quad msg\text{-}out \leftarrow \varepsilon$

$\quad \langle s' \rangle \leftarrow msg\text{-}in$

$\quad s \leftarrow s' \oplus s''$

$\quad$ **if** $\mathsf{Verify}_W(\mu, s)$ **then**

$\quad\quad term \leftarrow \text{TRUE}$

$\quad$ **return** $(msg\text{-}out, acc, term, sid, pid, sk, state)$

Figure 12: Specification of the PAK-Z protocol

Alice and runs the protocol with Bob, but does not attempt to compute the key using the hash function yet. Then the adversary corrupts Bob to obtain the password, and computes the key correctly using the hash function. Our proof technique basically shows that the adversary cannot make two correct hash function computations with two distinct passwords, but here the adversary has only made one. The problem seems to come from the fact that we believe the adversary has committed to a password by running the protocol, but we do not know what that password is. In the EKE2 protocol, the password used by the adversary can be determined (by the simulator in the proof) from the protocol messages and prior queries to the ideal cipher oracle, not just the later hash computation.

Looking deeper into the proof, we could try to do a backtracking argument to attempt to solve a CDH instance if the adversary could guess the correct password after a corruption with more than a certain probability. The problem is that it is impossible to tell (without the simulator solving CDH itself) when the adversary has succeeded, and thus when to backtrack. On the other hand, if we simply backtrack a polynomial number of times, and assuming the advantage of the adversary over online guessing is ϵ, we only seem to be able to solve CDH with probability on the order of ϵ^2, which means that the advantage of the adversary against PPK is linearly related to the square root of advantage of an adversary solving CDH. That is, if an advantage of an adversary solving CDH is bounded by 2^{-80}, then we could (at most) show that the advantage of an adversary against PPK was 2^{-40} over an online dictionary attack. Actually, the advantage would even be significantly greater, due to other factors in the coefficients of the advantages. Therefore, we feel the concrete security of this protocol does not provide practical justification for its use when concerned with forward secrecy.