# PAK-Z+

Craig Gentry, Philip MacKenzie, Zulfikar Ramzan
DoCoMo USA Labs
{cgentry,philmac,ramzan}@docomolabs-usa.com

August 15, 2005

**Abstract**

We present a revised version of the of the PAK-Z protocol, called the PAK-Z+ protocol, and give a complete proof of security. The PAK-Z+ protocol is being considered for inclusion in the IEEE P1363.2 standard proposal for Password-Based Public Key Cryptographic Techniques, and this document is presented in support of the inclusion of PAK-Z+ in IEEE P1363.2.

## 1 Introduction

It has been shown that the PAK-Z protocol presented in MacKenzie [34] is not resilient to server compromise when instantiated with some standard signature schemes. We present a modification, called the PAK-Z+ protocol and prove that it is resilient to server compromise.

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),[1] 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.[2] 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 [42]. 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, 41, 24, 25, 28, 29, 31, 32, 37, 42] (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 variant PAK-Z was developed in [34]. PAK-X, PAK-Y, and PAK-Z all assume a client/server model, and are all claimed to be resilient to server compromise. This means that an attacker who compromises a server's password file should not be able to impersonate a client, at least not without running an offline dictionary attack to determine the password. Of these three protocols, PAK-Z provides the most general way

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

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

for "augmenting" the basic PAK protocol to provide resilience to server compromise. Although all three protocols have proofs of security (in the random oracle model), we show there is a flaw in the proof of security for PAK-Z, and in fact, that it is not resilient to server compromise.

In this document, we propose a revision of PAK-Z called PAK-Z+, and provide a proof that it is resilient to server compromise.

## 2 Definitions

Let $\kappa$ 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 5). Let $t_{\mathsf{exp}}$ be the time required to perform an exponentiation in $G_q$.

**Notation.** We denote by $\Omega$ 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 $\Omega$ 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 [34] 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.**

A *digital signature scheme* $\mathcal{S}$ is a triple $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Verify})$ of algorithms, the first two being probabilistic, and all running in expected polynomial time. $\mathsf{Gen}$ takes as input $1^\kappa$ and outputs a public key pair $(pk, sk)$, i.e., $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$. $\mathsf{Sign}$ takes a message $m$ and a secret key $sk$ as input and outputs a signature $\sigma$ for $m$, i.e., $\sigma \leftarrow \mathsf{Sig}_{sk}(m)$. $\mathsf{Verify}$ takes a message $m$, a public key $pk$, and a candidate signature $\sigma'$ for $m$ as input and returns the bit $b = 1$ if $\sigma'$ 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 will assume the signature scheme used in PAK-Z+ is existentially unforgeable against adaptive chosen message attacks (probably too strong) (see Section 5). We also assume that for the signature scheme used in PAK-Z, each secret key has a $\kappa'$-bit representation, where $\kappa' \geq \kappa$.

## 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.[3]

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 $ID \stackrel{\text{def}}{=} \mathit{Clients} \cup \mathit{Servers}$, 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 $C \in \mathit{Clients}$ has a secret password $\pi_{C,S}$ for each $S \in \mathit{Servers}$, and each server $S \in \mathit{Servers}$ has a vector $\pi_S = \langle \pi_S[C] \rangle_{C \in \mathit{Clients}}$. Entry $\pi_S[C]$ is the *password record.* Let *Password* be a (possibly small) set from which passwords are selected. We will assume that $\pi_{C,S} \stackrel{R}{\leftarrow} \mathit{Password}$ (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 $\Pi_i^U$.

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 $\Pi_i^U$. 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 $\Pi_i^U$ to accept or terminate, this will also be shown to $\mathcal{A}$.[4] 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 $\Pi_i^C$ (where $C \in \mathit{Clients}$) and $\Pi_j^S$ (where $S \in \mathit{Servers}$), and outputs the transcript of the execution. 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 $\Pi_i^U$.

**Test** $(U,\ i)$**:** causes $\Pi_i^U$ 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** $(C, S)$**:** This returns $\pi_{C,S}$.

**Corrupt** $(S, C)$**:** This returns $\pi_S[C]$.

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

[4] 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.

Note that our $\mathsf{Corrupt}$ queries correspond to the weak corruption model of [2].

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

**Freshness.** Here we modify the two notions of freshness given in [2] to deal with systems that are designed to be resilient to server compromise. Specifically, we will allow a server instance to be considered fresh, even if that server has been compromised. This is because even after server compromise, the attacker should not be able to authenticate to an instance of this server (at least not unless the attacker runs an offline dictionary attack).

An instance $\Pi_i^U$ with partner-id $U'$ 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 $\Pi_i^U$, (3) $U \in \mathit{Servers}$ and a $\mathsf{Corrupt}\,(U', U)$ query occurs, or (4) $U \in \mathit{Clients}$ and a $\mathsf{Corrupt}\,(U, U')$ or a $\mathsf{Corrupt}\,(U', U)$ query occurs. (For convenience, when we do not make a requirement for forward secrecy, we simply disallow $\mathsf{Corrupt}$ queries to clients.) An instance $\Pi_i^U$ with partner-id $U'$ 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 $\Pi_i^U$, (3) $U \in \mathit{Servers}$ and a $\mathsf{Corrupt}\,(U', U)$ query occurs before the $\mathsf{Test}$ query and a $\mathsf{Send}\,(U,\ i,\ M)$ query occurs for some string $M$, or (4) $U \in \mathit{Clients}$ and a $\mathsf{Corrupt}\,(U, U')$ or a $\mathsf{Corrupt}\,(U', U)$ query occurs before the $\mathsf{Test}$ 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 $\mathsf{Test}$ query directed to some fresh instance $\Pi_i^U$ that has terminated, and eventually outputs a bit $b'$, where $b' = b$ for the bit $b$ that was selected in the $\mathsf{Test}$ 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 instance $\Pi_j^S$ with partner-id $C$ terminates without having a partner oracle before any $\mathsf{Corrupt}\,(C, S)$ query. We define $\mathsf{Adv}_P^{\mathrm{s2c}}(\mathcal{A})$ to be the probability that a client instance $\Pi_i^C$ with partner-id $S$ terminates without having a partner oracle before any $\mathsf{Corrupt}\,(S, C)$ query or $\mathsf{Corrupt}\,(C, S)$ query.

The following fact is easily verified.

**Fact 3.1**

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

## 4 PAK-Z and PAK-Z+ Protocols

In this section we describe the PAK-Z+ protocol. The PAK-Z+ protocol is shown in Figure 1. (For comparison, the PAK-Z protocol is shown in Figure 2.) The function $\textsc{acceptable}(\cdot)$ must be predefined for a specific abelian group $\overline{G}$ where $G_q$ is a subgroup of $\overline{G}$. Then $\textsc{acceptable}(v)$ 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 $\textsc{acceptable}(m)$ 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 $m \in G_q$.

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 $H(ASCII(i) \parallel x)$, where $H(\cdot)$ is then instantiated using SHA-1 as in [3].

3. In PAK-Z+, $H_1$ outputs values in $G_q$, $H_2$ and $H_6$ output $\kappa'$-bit values, where $\kappa'$ is the length of a secret key in a given signature scheme, $H_3$, $H_4$, and $H_5$ output $\kappa$-bit values. The instantiations for hash functions have an effect on the efficiency of the protocol. See [34] for details.

4. In PAK-Z+, the server stores $\langle (H_1(C, S, \pi_{C,S}))^{-1}, W, H_2(C, S, \pi_{C,S}) \oplus V, H_3(V) \rangle$, for public/secret signature key pair $(W, V)$, while in PAK-Z, the last hash is absent.[5]

5. 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, 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 PAK-Z+, we also use 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. (In the model used to prove PAK-Z this was not an issue, since it was only concerned about an attacker impersonating a client after a server compromise, not impersonating another server.)

6. PAK-Z+ has an extra hash of the secret signature key that the client uses to verify that the decrypted secret key is correct. This prevents an attacker from modifying the encryption of the secret key and forcing a client to produce signatures on related keys (and thus potentially determining the actual signature key).

7. PAK-Z+ does not encrypt the final signature, since it was not necessary to do so (according to the proof of security).

## 5 Security of PAK-Z+

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 $\textsc{acceptable}(X)$ and $Y = g^y$, let $\mathsf{DH}(X, Y) = X^y$, else if $X = g^x$ and $\textsc{acceptable}(Y)$, let $\mathsf{DH}(X, Y) = Y^x$. (Note that if $X = g^x$

[5] Note that when the group $G_q$ is a subgroup of $Z_p^*$, there is an implicit exponentiation by $(p-1)/q$ in the computation of $H_1(C, S, \pi_C)$

| Client $C$ |  | Server $S$ |
|---|---|---|
|  |  | $\pi_S[C] = \langle \gamma', W, V', V'' \rangle$ where |
| Input: $S$, $\pi$ |  | $\gamma' = (H_1(\langle C, S, \pi_{C,S} \rangle))^{-1}$ |
|  |  | $V' = H_2(\langle C, S, \pi_{C,S} \rangle) \oplus \langle V \rangle$ |
|  |  | $V'' = H_3(\langle V \rangle)$ |
| $x \overset{R}{\leftarrow} \mathbb{Z}_q$ |  |  |
| $\alpha \leftarrow g^x$ |  |  |
| $\gamma \leftarrow H_1(\langle C, S, \pi \rangle)$ |  |  |
| $m \leftarrow \alpha \cdot \gamma$ |  |  |
|  | $\xrightarrow{\langle C, m \rangle}$ |  |
|  |  | Abort if $\neg$ACCEPTABLE$(m)$ |
|  |  | $y \overset{R}{\leftarrow} \mathbb{Z}_q$ |
|  |  | $\mu \leftarrow g^y$ |
|  |  | $\langle \gamma', W, V', V'' \rangle \leftarrow \pi_S[C]$ |
|  |  | $\alpha \leftarrow m \cdot \gamma'$ |
|  |  | $\sigma \leftarrow \alpha^y$ |
|  |  | $sid \leftarrow \langle C, S, m, \mu \rangle$ |
|  |  | $a' \leftarrow H_6(\langle sid, \sigma, \gamma' \rangle)$ |
|  |  | $a \leftarrow a' \oplus V'$ |
|  |  | $k \leftarrow H_4(\langle sid, \sigma, \gamma' \rangle)$ |
|  | $\xleftarrow{\langle \mu, k, a, V'' \rangle}$ |  |
| $\sigma \leftarrow \mu^x$ |  |  |
| $\gamma' \leftarrow \gamma^{-1}$ |  |  |
| $sid \leftarrow \langle C, S, m, \mu \rangle$ |  |  |
| Abort if $k \neq H_4(\langle sid, \sigma, \gamma' \rangle)$ |  |  |
| $a' \leftarrow H_6(\langle sid, \sigma, \gamma' \rangle)$ |  |  |
| $V' \leftarrow a' \oplus a$ |  |  |
| $\langle V \rangle \leftarrow H_2(\langle C, S, \pi \rangle) \oplus V'$ |  |  |
| Abort if $V'' \neq H_3(\langle V \rangle)$ |  |  |
| $s \leftarrow \mathsf{sig}_V(sid)$ |  |  |
|  | $\xrightarrow{s}$ |  |
|  |  | Abort if $\mathsf{Verify}_W(sid, s) = 0$ |

Figure 1: PAK-Z+ Protocol. Partner ID for $C$ is $pid_C = S$, and partner ID for $S$ is $pid_S = C$. Shared session key is $sk = H_5(\langle sid, \sigma, \gamma' \rangle)$.

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

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

Let $\mathsf{Adv}^{\mathrm{CDH}}_{G_q}(t,n) = \max_{\mathcal{A}}\left\{\mathsf{Adv}^{\mathrm{CDH}}_{G_q}(\mathcal{A})\right\}$, 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 any probabilistic polynomial time $\mathcal{A}$, $\mathsf{Adv}^{\mathrm{CDH}}_{G_q}(\mathcal{A})$ is negligible.

**Security for signature schemes.** We specify existential unforgeability versus chosen message attacks [21] for a signature scheme $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Verify})$. A forger $\mathcal{F}$ is given $pk$, where $(pk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$, 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 $(m,\sigma)$ such that $\mathsf{Verify}_{pk}(m,\sigma) = 1$, where $m$ was not one of the messages signed by the signature oracle. We say $\mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(\mathcal{F}) = \Pr(\mathcal{F} \text{ succeeds})$, and $\mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(t,u) = \max_{\mathcal{F}}\left\{\mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(\mathcal{F})\right\}$, 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 any probabilistic polynomial time $\mathcal{F}$, $\mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(\mathcal{F})$ 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 [36, 38].

### 5.1 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 fresh instances with greater advantage than that of an offline dictionary attack.

**Theorem 5.1** *Let $P$ be the protocol described in Figure 1 (and formally described in Appendix* **??***), 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 $n_{\mathrm{se}}, n_{\mathrm{ex}}, n_{\mathrm{re}}$ queries of type* $\mathsf{Send}, \mathsf{Execute}, \mathsf{Reveal}$*, respectively, and $n_{\mathrm{ro}}$ queries to the random oracles. Let $b_{\mathrm{co}} = 1$ if $\mathcal{A}$ makes a corrupt query to a server, and otherwise $b_{\mathrm{co}} = 0$. Then for $t' = O(t + ((n_{\mathrm{ro}})^2 + n_{\mathrm{se}} + n_{\mathrm{ex}})t_{\mathsf{exp}})$ and $\epsilon = O\left(n_{\mathrm{se}}\mathsf{Adv}^{\mathrm{CDH}}_{G_q}(t', (n_{\mathrm{ro}})^2) + n_{\mathrm{se}}\mathsf{Succ}^{\text{eu-cma}}_{\mathcal{S},\kappa}(t', n_{\mathrm{se}}) + \frac{(n_{\mathrm{se}}+n_{\mathrm{ex}})(n_{\mathrm{ro}}+n_{\mathrm{se}}+n_{\mathrm{ex}})}{q}\right)$:*

$$\mathsf{Adv}^{\text{ake-fs}}_{P}(\mathcal{A}) \leq \frac{n_{\mathrm{se}}(1-b_{\mathrm{co}}) + n_{\mathrm{ro}}b_{\mathrm{co}}}{N} + \epsilon.$$

Using essentially the same arguments, we can also show that

$$\mathsf{Adv}^{\mathrm{s2c}}_{P}(\mathcal{A}) \leq \frac{n_{\mathrm{se}}}{N} + \epsilon$$

and

$$\mathsf{Adv}^{\mathrm{c2s}}_{P}(\mathcal{A}) \leq \frac{n_{\mathrm{se}}(1-b_{\mathrm{co}}) + n_{\mathrm{ro}}b_{\mathrm{co}}}{N} + \epsilon.$$

**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 use the terminology "in a CLIENT ACTION $i$ query to $\Pi_j^C$" to mean "in a $\mathsf{Send}$ query to $\Pi_j^C$ that results in the CLIENT ACTION $i$ procedure being executed," and "in a SERVER ACTION $i$ query to $\Pi_j^S$" to mean "in a $\mathsf{Send}$ query to $\Pi_j^S$ that results in the SERVER ACTION $i$ procedure being executed,"

We assume without loss of generality that $n_{\mathrm{ro}}$ 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 $H_1(\langle C, S, \pi\rangle)$ queries are answered with the value $g^{\psi[C,S,\pi]}$, where $\psi[C, S, \pi] \overset{R}{\leftarrow} \mathbb{Z}_q$. That is, we assume we know the discrete logs of the outputs of $H_1(\cdot)$ queries. WLOG, we assume that for each $H_\ell(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ query made by $\mathcal{A}$ for $\ell \in \{4, 5, 6\}$, the corresponding $H_{\ell'}(\cdot)$ queries are made, for $\ell' \in \{4, 5, 6\} \setminus \{\ell\}$. We refer to this as an $H_{4,5,6}(\cdot)$ query, which outputs a triple of values. Similarly, we define an $H_{1,2}(\cdot)$ query, which outputs a pair of values.

Say a client instance $\Pi_i^C$ is *paired with* a server instance $\Pi_j^S$ if there is a CLIENT ACTION 0 query to $\Pi_i^C$ with input $S$ and output $\langle C, m\rangle$, there is a SERVER ACTION 1 query to $\Pi_j^S$ with input $\langle C, m\rangle$ and output $\langle \mu, k, \cdot, \cdot\rangle$, and there is a CLIENT ACTION 1 query to $\Pi_i^C$ with input $\langle \mu, k, \cdot, \cdot\rangle$. Furthermore, say $\Pi_i^C$ is *fully paired with* a server instance $\Pi_j^S$ if it is paired with $\Pi_j^S$ and the output of the SERVER ACTION 1 query to $\Pi_j^S$ is exactly the same as the input of the CLIENT ACTION 1 query to $\Pi_i^C$. Say a server instance $\Pi_j^S$ is *paired with* a client instance $\Pi_i^C$ if $\Pi_i^C$ is paired with $\Pi_j^S$. Note that if an instance terminates and it is paired with another instance, then it will be partnered with that other 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.

- $\mathsf{testpw}(C, i, S, \pi)$: $\mathcal{A}$ makes a CLIENT ACTION 1 query to $\Pi_i^C$ with input $\langle \mu, k, \cdot, \cdot\rangle$ and previously $\mathcal{A}$ made an $H_{4,5,6}(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ query with $k$ being the first element of the output triple, a CLIENT ACTION 0 query to a client instance $\Pi_i^C$ with input $S$ and output $\langle C, m\rangle$, and an $H_{1,2}(\langle C, S, \pi\rangle)$ query returning $((\gamma')^{-1}, \cdot)$, where $\sigma = DH(\alpha, \mu)$ and $m = \alpha \cdot (\gamma')^{-1}$. The associated triple of this event is the output of the $H_{4,5,6}(\cdot)$ query.

- $\mathsf{impersonateserver}(C, i, S)$: $\mathcal{A}$ makes a CLIENT ACTION 1 query to $\Pi_i^C$ with input $\langle \mu, k, \cdot, \cdot\rangle$, and previously $\mathcal{A}$ made a $\mathsf{Corrupt}$ $(S, C)$ query returning $\langle \gamma', \cdot, \cdot, \cdot\rangle$, an $H_{4,5,6}(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ query returning $(k, h_5, h_6)$, a CLIENT ACTION 0 query to a client instance $\Pi_i^C$ with input $S$ and output $\langle C, m\rangle$, where $\sigma = DH(\alpha, \mu)$ and $m = \alpha \cdot (\gamma')^{-1}$. The associated triple of this event is $(k, h_5, h_6)$.

- $\mathsf{testpw}(S, j, C, \pi)$: $\mathcal{A}$ makes an $H_{4,5,6}(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ query, and previously made a SERVER ACTION 1 query to a server instance $\Pi_j^S$ with input $\langle C, m\rangle$ and output $\langle \mu, k, \cdot, \cdot\rangle$, and an $H_{1,2}(\langle C, S, \pi\rangle)$ query returning $((\gamma')^{-1}, \cdot)$, where $\sigma = DH(\alpha, \mu)$, $m = \alpha \cdot (\gamma'_1)^{-1}$. The associated triple of this event is the triple $(k, sk_S^j, a')$ of values generated by $\Pi_j^S$.

- $\mathsf{testpw}(C, i, S, j, \pi)$: both a $\mathsf{testpw}(S, i, C, \pi)$ event occurs, where $\Pi_i^C$ is paired with $\Pi_j^S$.

- $\mathsf{testexecpw}(C, i, S, j, \pi)$: $\mathcal{A}$ makes an $H_{4,5,6}(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query, and previously made an $\mathsf{Execute}$ $(C,\ i,\ S,\ j)$ query that generates $m$ and $\mu$, and an $H_1(\langle C, S, \pi \rangle)$ query returning $(\gamma')^{-1}$, where $\sigma = DH(\alpha, \mu)$, and $m = \alpha \cdot (\gamma')^{-1}$. The associated triple of this event is the triple $(k, sk_S^j, a')$ of values generated by $\Pi_j^S$.
- $\mathsf{testpwoffline}(C, S, \pi)$: a $\mathsf{Corrupt}$ $(S, C)$ query has been made, and an $H_{1,2}(\langle C, S, \pi \rangle)$ query has been made (either before or after the $\mathsf{Corrupt}$ $(S, C)$ query).
- $\mathsf{correctpw}$: for some $C$ and $S$, before any $\mathsf{Corrupt}$ $(C, S)$ query, either a $\mathsf{testpw}(C, i, S, \pi_{C,S})$ event occurs for some $i$, a $\mathsf{testpw}(S, j, C, \pi_{C,S})$ event occurs for some $j$. or a $\mathsf{testpwoffline}(C, S, \pi_{C,S})$ event occurs.
- $\mathsf{correctpwexec}$: a $\mathsf{testexecpw}(C, i, S, j, \pi_{C,S})$ event occurs for some $C$, $i$, $S$, and $j$.
- $\mathsf{doublepwserver}$: before a $\mathsf{Corrupt}$ $(C, S)$ or $\mathsf{Corrupt}$ $(S, C)$ query, both a $\mathsf{testpw}(S, j, C, \pi)$ event and a $\mathsf{testpw}(S, j, C, \pi')$ event occur, where $\pi \neq \pi'$.
- $\mathsf{pairedguess}$: a $\mathsf{testpw}(C, i, S, j, \pi)$ event occurs, or an $\mathsf{impersonateserver}(C, i, S)$ event occurs where $\Pi_i^C$ is paired with $\Pi_j^S$, for some $C$, $i$, $S$, $j$, and $\pi$.
- $\mathsf{forgesig}(S, j, C)$: before a $\mathsf{Corrupt}$ $(C, S)$ query and before a $\mathsf{correctpw}$ event, a Server Action 2 query to $\Pi_j^S$ is made with input $s$, where $\Pi_j^S$ is unpaired and for $W$ the public key stored in $\pi_S[C]$, and $sid$ computed in the associated Server Action 1 query, $\mathsf{Verify}_W(sid, s) = 1$.

**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 $H_{4,5,6}(\cdot)$ query (made by the adversary), Let $E_2$ be the event that a $\mu$ value generated in a Server Action 1 or $\mathsf{Execute}$ query is equal to a $\mu$ value generated in a previous Server Action 1 or $\mathsf{Execute}$ query, a $\mu$ value sent as input in a previous Client Action 1 query, or a $\mu$ value in a previous $H_{4,5,6}(\cdot)$ 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 5.2** 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 $\mathsf{Send}$ and $\mathsf{Execute}$ 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 $\mathsf{Send}$ and $\mathsf{Execute}$ queries.

We assume each server $S$ initializes the second and fourth element of $\pi_S[C] = (\cdot, W, \cdot, V'')$ where $V'' \xleftarrow{R} \{0,1\}^\kappa$, and $(W, V) \leftarrow \mathsf{Gen}(1^\kappa)$. ($S$ also saves $V$.)

The queries in $P_2$ are changed as follows:

- In an $\mathsf{Execute}$ $(C,\ i,\ S,\ j)$ query, $m \leftarrow g^{\tau[i,C]}$, where $\tau[i,C] \xleftarrow{R} \mathbb{Z}_q$, $\mu \leftarrow g^{\tau[j,S]}$, where $\tau[j,S] \xleftarrow{R} \mathbb{Z}_q$, $k \xleftarrow{R} \{0,1\}^\kappa$, $a' \xleftarrow{R} \{0,1\}^{\kappa'}$, $s \leftarrow \mathsf{Sig}_V(sid)$, and $sk_C^i \leftarrow sk_S^j \xleftarrow{R} \{0,1\}^\kappa$.
- In a Client Action 0 query to instance $\Pi_i^C$, $m \leftarrow g^{\tau[i,C]}$, where $\tau[i,C] \xleftarrow{R} \mathbb{Z}_q$.
- In a Server Action 1 query to instance $\Pi_j^S$, $\mu \leftarrow g^{\tau[j,S]}$, where $\tau[j,S] \xleftarrow{R} \mathbb{Z}_q$, and $sk_S^j, k \xleftarrow{R} \{0,1\}^\kappa$ and $a' \xleftarrow{R} \{0,1\}^{\kappa'}$.
- In a Client Action 1 query to instance $\Pi_i^C$, do the following.
  - If this query causes a $\mathsf{testpw}(C, i, S, \pi_{C,S})$ event to occur, then set $sk_C^i$ to the second element of the associated triple of the $\mathsf{testpw}(C, i, S, \pi_{C,S})$ event, and run the remainder of the protocol for Client Action 1 (starting with the computation of $a'$).
  - If this query causes an $\mathsf{impersonateserver}(C, i, S)$ event to occur, then set $sk_C^i$ to the second element of the associated triple of the event, set $V'$ as in the protocol, set $V \leftarrow V' \oplus \langle V_S \rangle \oplus V'_S$ (where $V_S$ and $V'_S$ are the $V$ and $V'$ values from $\pi_S[C]$), and continue with the protocol.
  - Otherwise, if $\Pi_i^C$ is fully paired with a server instance $\Pi_j^S$, $sk_C^i \leftarrow sk_S^j$, and set $s \leftarrow \mathsf{Sig}_V(sid)$.
  - Otherwise, $\Pi_i^C$ aborts.
- In an $H_{1,2}(\langle C, S, \pi_{C,S} \rangle)$ query, if this query causes a $\mathsf{testpwoffline}(C, S, \pi_{C,S})$ event (i.e., there has been a $\mathsf{Corrupt}$ $(S, C)$ query), set the output pair $(h_1, h_2)$ using $\pi_S[C] = \{\gamma', W, V', V''\}$ and associated secret signature key $V$ by setting $h_1 \leftarrow (\gamma')^{-1}$ and $h_2 \leftarrow V' \oplus V$.
- In an $H_3(\langle V \rangle)$ query (for $V$ associated with a $W$ in a $\pi_S[C]$ record), output the associated $V''$ value stored in that record.
- In an $H_{4,5,6}(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query, if this $H_{4,5,6}(\cdot)$ query causes a $\mathsf{testpw}(S, j, C, \pi_{C,S})$, or $\mathsf{testexecpw}(C, i, S, j, \pi_{C,S})$ event to occur, then output the associated triple of that event.
- In a $\mathsf{Corrupt}$ $(S, C)$ query, if this query causes a $\mathsf{testpwoffline}(C, S, \pi_{C,S})$ event, set the first element of $\pi_S[C]$ to $(H_1(\langle C, S, \pi_{C,S} \rangle))^{-1}$ and set the third element of $\pi_S[C]$ to $H_2(\langle C, S, \pi_{C,S} \rangle) \oplus V$, for the $V$ value associated with $\pi_S[C]$. Otherwise set the first element to $g^{\psi[C,S,\pi_{C,S}]}$, where $\psi[C, S, \pi_{C,S}] \xleftarrow{R} \mathbb{Z}_q$ and the third element to $V' \xleftarrow{R} \{0,1\}^{\kappa'}$.

**Claim 5.3** For any adversary $\mathcal{A}$,

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

**Proof:** $P_2$ is consistent with $P_1$ as long as the adversary is not able to guess the output of a hash query before it is made.[6] This probability is $\frac{O(n_{\mathrm{ro}})}{q} + \frac{O(n_{\mathrm{se}})}{2^\kappa}$, and the claim follows by noting $q \leq 2^\kappa$.

Note the calculation of $V$ when an $\mathsf{impersonateserver}(C, i, S)$ event occurs in a Client Action 1 query. Normally $\Pi_i^C$ would calculate $V \leftarrow V' \oplus H_2(C, S, \pi_{C,S})$. However, it may be that $H_2(C, S, \pi_{C,S})$ is not queried yet. But we know that $V_S' = H_2(C, S, \pi_{C,S}) \oplus V_S$ where $V_S$ and $V_S'$ are the $V$ and $V'$ values associated with $\pi_S[C]$, so $\Pi_i^C$ uses $V_S' \oplus V_S$ to calculate $V$. ▌

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

**Claim 5.4** 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 $\epsilon$ 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 $\rho_{i,C}, \rho'_{j,S} \xleftarrow{R} \mathbb{Z}_q$.
2. When $\mathcal{A}$ finishes, for every $H_{4,5,6}(\langle C, S, m, \mu, \sigma, \gamma' \rangle)$ query, where $m$ and $\mu$ were generated in an $\mathsf{Execute}\,(C,\ i,\ S,\ j)$ query, and an $H_1(\langle C, S, \pi_{C,S} \rangle)$ query returned $(\gamma')^{-1}$, add

$$\sigma X^{-\rho'_{j,S}} Y^{\psi[C,S,\pi_{C,S}]-\rho_{i,C}} g^{-\rho_{i,C}\rho'_{j,S}} g^{\psi[C,S,\pi_C]\rho'_{j,S}}$$

   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 $\epsilon$. 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 at most $n_{\mathrm{ro}}$, and its advantage is $\epsilon$. 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 $\mathsf{Adv}_{G_q}^{\mathrm{CDH}}(D) \leq \mathsf{Adv}_{G_q}^{\mathrm{CDH}}(t', n_{\mathrm{ro}})$. ▌

[6] The one situation where this is not quite as obvious is when a client instance $\Pi_i^C$ aborts when it is not fully paired, since it may be that it is paired. But if it is paired and not fully paired, then either the $a$ or $V''$ value sent by the paired server instance $\Pi_j^S$ is changed. But there is no $\mathsf{testpw}(C, i, S, \pi_{C,S})$ event and no $\mathsf{impersonateserver}(C, i, S)$ event, so the relevant $H_{4,5,6}(\cdot)$ query was not made. Therefore, the $V$ value computed by $\Pi_i^C$ would be completely random, and the probability of $H_3(\langle V \rangle) = V''$ would be negligible.

**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 $\Pi_i^C$, if a $\mathsf{testpw}(C, i, S, \pi_{C,S})$ event occurs and no $\mathsf{Corrupt}\ (C, S)$ query has been made, halt and say the adversary automatically succeeds.
2. In an $H_{4,5,6}(\cdot)$ query, if a $\mathsf{testpw}(S, j, C, \pi_{C,S})$ event occurs and no $\mathsf{Corrupt}\ (C, S)$ query has been made, halt and say the adversary automatically succeeds.
3. If a $\mathsf{testpwoffline}(C, S, \pi_{C,S})$ event occurs (this must be tested in $\mathsf{Corrupt}\ (S, C)$ queries and $H_{1,2}(\cdot)$ queries) and no $\mathsf{Corrupt}\ (C, S)$ query to a client has been made, halt and say the adversary automatically succeeds.

**Claim 5.5** For any adversary $\mathcal{A}$,

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

**Proof:** Obvious. ▮

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

**Claim 5.6** 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

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

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

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 $d \in \{1, \ldots, n_{\text{se}}\}$ and simulates $P_4$ for $\mathcal{A}$ with these changes:

1. In the $d$th 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 $\Pi_j^{S'}$ that receives the input $(C', m)$ that was output from the Client Action 0 query to $\Pi_{i'}^{C'}$, set $\mu \leftarrow Y g^{\rho'_{j,S'}}$, where $\rho'_{j,S} \xleftarrow{R} \mathbb{Z}_q$.

3. In a Client Action 1 query to $\Pi_{i'}^{C'}$, if $\Pi_{i'}^{C'}$ is unpaired, $D$ outputs an empty list and halts. Otherwise, if $\Pi_{i'}^{C'}$ is not fully paired, it aborts. (Note that we are not able to check for a $\mathsf{testpw}(C', i', S', \pi_{C,S})$ event nor for an $\mathsf{impersonateserver}(C', i', S')$ event.)

4. When $\mathcal{A}$ finishes, if $\Pi_{i'}^{C'}$ is paired with $\Pi_{j}^{S'}$, then for every $H_{4,5,6}(\langle C', S', m, \mu, \sigma, \gamma' \rangle)$ query, where $m$ and $\mu$ were generated by $\Pi_{i'}^{C'}$ and $\Pi_{j}^{S'}$, respectively, and either there was a $\mathsf{Corrupt}\,(S', C')$ query and $\gamma'$ was the first element of $\pi_{S'}[C']$, or an $H_{1,2}(C, S, \pi)$ query returned $(\gamma')^{-1}$, add

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

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

This simulation is perfectly indistinguishable from $P_4$, unless $\Pi_{i'}^{C'}$ is paired after the Client Action 1 query and there was either a $\mathsf{testpw}(C', i', S', \pi_{C,S})$ event or an $\mathsf{impersonateserver}(C', i', S')$ event (in which case, $D$ adds the correct $\mathsf{DH}(X, Y)$ to the list). In particular, if $\Pi_{i'}^{C'}$ is paired after the Client Action 1 query, then either it is fully paired, in which case it produces a correct signature, or it is not fully paired, in which case, either there was a $\mathsf{testpw}(C', i', S', \pi_{C,S})$ event or an $\mathsf{impersonateserver}(C', i', S')$ event (in which case $D$ adds the correct $\mathsf{DH}(X, Y)$ to the list), or not (in which case, the instance would abort) (see $P_2$). Note that if $D$ ends the simulation in a Client Action 1 query to $\Pi_{i'}^{C'}$, the $\mathsf{pairedguess}$ would not have occurred for that instance. Note that the probability of a $\mathsf{pairedguess}$ event occurring for $\Pi_{i'}^{C'}$ is at least $\frac{\epsilon}{n_{\rm se}}$, and in this case, $D$ adds the correct $\mathsf{DH}(X, Y)$ to the list.

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

**Protocol** $P_6$**.** Let $P_6$ be a protocol that is identical to $P_5$ except that in a Server Action 2 query, if this query causes a $\mathsf{forgesig}(S, j, C)$ event to occur, the server instance aborts.

**Claim 5.7** 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

$$\mathsf{Adv}_{P_5}^{\rm ake}(\mathcal{A}) \le \mathsf{Adv}_{P_4}^{\rm ake}(\mathcal{A}) + 2 n_{\rm se} \cdot \mathsf{Succ}_{\mathcal{S},\kappa}^{\rm eu\text{-}cma}(t', n_{\rm se}) + \frac{O(n_{\rm ro})}{q}.$$

**Proof:** Let $E$ be the event that for some $C$, $S$, and $i$ or $j$, either a $\mathsf{makesig}(C, i, S)$ event occurs, a $\mathsf{forgesig}(S, j, C)$ event occurs, or an $H_3(\langle V \rangle)$ query is made for a $V$ associated with $\pi_S[C]$ for which a $\mathsf{Corrupt}\,(C, S)$ has not occurred. Obviously, if $E$ does not occur, then $P_6$ and $P_5$ are indistinguishable. Let $\epsilon$ be the probability that $E$ occurs when $\mathcal{A}$ is running against protocol $P_5$. Then $\Pr(\mathsf{Succ}_{P_6}^{\rm ake}(\mathcal{A})) \le \Pr(\mathsf{Succ}_{P_5}^{\rm ake}(\mathcal{A})) + \epsilon$, and thus by Fact 3.1, $\mathsf{Adv}_{P_6}^{\rm ake}(\mathcal{A}) \le \mathsf{Adv}_{P_5}^{\rm ake}(\mathcal{A}) + 2\epsilon$.

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. Let $C, S$ be the $d$th client/server pair from any $\mathsf{Execute}$, $\mathsf{Corrupt}$, or Server Action 1 queries. Set the value $W$ in $\pi_S[C]$ to $pk$.
2. In a Client Action 1 query to a fully paired $\Pi_i^C$, use the signature oracle for $pk$ to perform the signature computation.
3. In a Client Action 1 query with input $\langle \mu, k, a, V'' \rangle$ that causes an $\mathsf{impersonateserver}(C, i, S)$ event (and thus the client instance is not paired with any server instance, see $P_5$) if $V''$ is the fourth element of $\pi_S[C]$ then if $a = V' \oplus a'$ (for $a'$ the third element from the associated triple of the event and $V'$ from $\pi_S[C]$) compute $s$ using the signature oracle for $pk$, and otherwise abort. If $V''$ is not the fourth element of $\pi_S[C]$, then for every $V^*$ such that $H_3(\langle V^* \rangle) = V''$, compute $V^{**} \leftarrow a \oplus a' \oplus \langle V^* \rangle \oplus V'$ (for $V'$ from $\pi_S[C]$) and $s = \mathsf{Sig}_{V^{**}}(sid)$, and check if $\mathsf{Verify}_W(sid, s) = 1$. (The intuition here is that if $V^*$ would have been computed by the client instance, then this would fix $H_2(C, S, \pi_{C,S}) = V^* \oplus V'$, and thus we could compute the value $V^{**}$ that would be the secret key corresponding to $W$.) If so, $F$ outputs $(sid, s)$ and halts. Otherwise $F$ has $\Pi_i^C$ abort.
4. If an $H_3(\langle V \rangle)$ query is made, check if $\mathsf{Verify}_W(r, s) = 1$ for a random $r$, and $s = \mathsf{Sig}_V(r)$. If so, $F$ outputs $(r, s)$ and halts.
5. In a Server Action 2 query to an instance $\Pi_j^S$ (note this could be any instance of $S$) if a $\mathsf{forgesig}(S, j, C)$ event occurs, $F$ outputs $(sid, s)$ and halts (for values $sid$ and $s$ from $\Pi_j^S$).
6. If $\mathcal{A}$ makes a $\mathsf{Corrupt}$ $(C, S)$ query, $F$ halts and fails.
7. If $\mathcal{A}$ finishes, $F$ halts and fails.

Note that the simulation is indistinguishable from $P_5$ until $F$ outputs a forged signature for the $d$th client/server pair considered by the adversary. Since it always produces a forged signature when a $\mathsf{forgesig}(S, j, C)$ event occurs for the $d$th client/server pair, $F$ will produce a forged signature with probability at least $\frac{\epsilon}{n_{\text{se}}}$. Note that the $sid$ output would not have been asked to the signature oracle since $\mathsf{impersonateserver}(C, i, S)$ can never occur for a client instance that is paired with a server instance (see $P_5$).

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

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

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

**Protocol $P_7$.** Let $P_7$ be a protocol that is identical to $P_6$ 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 $\mathsf{pairedguess}$ or $\mathsf{correctpw}$.

**Claim 5.8** For any adversary $\mathcal{A}$ running in time $t$, there is a $t' = O(t + (n_{\rm ro} + n_{\rm se} + n_{\rm ex})t_{\sf exp})$ such that

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

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

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 $P_6$ for $\mathcal{A}$ with these changes:

1. In an $H_{1,2}(\langle C, S, \pi\rangle)$ query, output $X^{\psi[C,S,\pi]}g^{\psi'[C,S,\pi]}$ as the first element of the output pair, where $\psi[C, S, \pi] \stackrel{R}{\leftarrow} \{0, 1\}$ and $\psi'[C, S, \pi] \stackrel{R}{\leftarrow} \mathbb{Z}_q$.
2. In a Server Action 1 query to a server instance $\Pi_j^S$ with input $\langle C, m\rangle$ where ACCEPTABLE$(m)$ is true, set $\mu \leftarrow Yg^{\rho'_{j,S}}$.
3. Tests for $\mathsf{correctpw}$ (from $P_4$) and $\mathsf{pairedguess}$ (from $P_5$) are not made. In particular, a Client Action 1 query to a client instance that is not fully paired causes the instance to abort, and $H_{4,5,6}(\cdot)$ queries do not cause any backpatching.
4. When $\mathcal{A}$ finishes or makes a $\mathsf{Corrupt}$ query, for every pair of queries $H_{4,5,6}(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$ and $H_{4,5,6}(\langle C, S, m, \mu, \hat{\sigma}, \hat{\gamma}'\rangle)$, where there was a Server Action 1 query to a server instance $\Pi_j^S$ with input $\langle C, m\rangle$ and output $\langle \mu, k, a, V''\rangle$, an $H_{1,2}(\langle C, S, \pi\rangle)$ query that returned $(\gamma')^{-1}$, an $H_{1,2}(\langle C, S, \hat{\pi}\rangle)$ query that returned $(\hat{\gamma}')^{-1}$, and $\psi[C, S, \pi] \neq \psi[C, S, \hat{\pi}]$, add

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

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

This simulation is perfectly indistinguishable from $P_6$ until a $\mathsf{doublepwserver}$ event, a $\mathsf{pairedguess}$ event, or a $\mathsf{correctpw}$ event occurs. If a $\mathsf{doublepwserver}$ event occurs, then with probability $\frac{1}{2}$ it occurs for two passwords $\pi$ and $\hat{\pi}$ with $\psi[C, S, \pi] \neq \psi[C, S, \hat{\pi}]$, and in this case $D$ adds the correct $\mathsf{DH}(X, Y)$ to the list. If a $\mathsf{correctpw}$ event or a $\mathsf{pairedguess}$ event occurs, then the $\mathsf{doublepwserver}$ event would never have occurred in $P_6$, since $P_6$ 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 $P_6$, but this does not change the fact that a $\mathsf{doublepwserver}$ event will occur with probability at least $\epsilon$ 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 $t' = O(t + ((n_{\rm ro})^2 + n_{\rm se} + n_{\rm ex})t_{\sf exp})$. The claim follows from the fact that $\mathsf{Adv}_{G_q}^{\rm CDH}(D) \leq \mathsf{Adv}_{G_q}^{\rm CDH}(t', (n_{\rm ro})^2)$. ▌

**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 $\mathsf{testpw}(C, S, \pi)$ and returns TRUE if $\pi = \pi_{C,S}$, and FALSE otherwise. It also accepts $\mathsf{Corrupt}\ (C, S)$ queries and returns $\pi_{C,S}$, and accepts $\mathsf{Corrupt}\ (S, C)$ queries and returns $\pi_S[C]$. When a $\mathsf{Corrupt}$ query is received in the protocol, it is answered using the same $\mathsf{Corrupt}$ 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 $\Pi_i^C$ and password $\pi$, or the first $\mathsf{testpw}(S, j, C, \pi)$ event occurs for an instance $\Pi_j^S$ and password $\pi$, or a $\mathsf{testpwoffline}(C, S, \pi)$ event occurs, a $\mathsf{testpw}(C, S, \pi)$ query is made to the password oracle to see if $\pi = \pi_{C,S}$.

**Claim 5.9** For any adversary $\mathcal{A}$,

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

**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

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

First, there are at most $n_{\text{se}}$ queries to the password oracle before a $\mathsf{Corrupt}$ query, and if a $\mathsf{Corrupt}$ query to a server has occurred, at most $n_{\text{ro}}$ queries to the password oracle in total. (Note that any online password guess also corresponds to an offline password guess after a $\mathsf{Corrupt}$ query to a server.) Passwords are chosen uniformly from a dictionary of size $N$, so $\Pr(\mathsf{correctpw}) \leq \frac{n_{\text{se}}(1-b_{\text{co}})+n_{\text{ro}}b_{\text{co}}}{N}$.

Now we compute $\Pr(\mathsf{Succ}^{\text{ake}}_{P_8}(\mathcal{A})|\neg\mathsf{correctpw})$. If $\mathsf{correctpw}$ does not occur, then $\mathcal{A}$ succeeds by making a $\mathsf{Test}$ query to a fresh instance $\Pi_i^U$ and guessing the bit used in that $\mathsf{Test}$ 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 $\mathsf{Reveal}$ queries. Recall that since $\Pi_i^U$ is fresh, there could be no $\mathsf{Reveal}\ (U,\ i)$ query, and if $\Pi_j^{U'}$ is partnered with $\Pi_i^U$, no $\mathsf{Reveal}\ (U',\ j)$ query. Second note that since *sid* includes $m$ and $\mu$ 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 $\mathsf{Reveal}$ queries is independent of $sk_U^i$.

Second we examine $H_{4,5,6}(\cdot)$ queries. Note that in $P_6$, until a $\mathsf{correctpw}$ event or a $\mathsf{Corrupt}\ (C, S)$ or $\mathsf{Corrupt}\ (S, C)$ query, no unpaired instance $\Pi_i^C$ with partner-id $S$ will terminate, and until a $\mathsf{correctpw}$ event or a $\mathsf{Corrupt}\ (C, S)$ query, no unpaired instance $\Pi_j^S$ with partner-id $C$ will terminate, and thus an instance may only be fresh and receive a $\mathsf{Test}$ query if it is paired. However, an $H_{4,5,6}(\cdot)$ query will never reveal $sk_U^i$ if $\Pi_i^U$ is paired (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

$$\Pr(\mathsf{Succ}^{\text{ake}}_{P_8}(\mathcal{A})) \quad \leq \quad \Pr(\mathsf{correctpw}) + \Pr(\mathsf{Succ}^{\text{ake}}_{P_8}(\mathcal{A})|\neg\mathsf{correctpw})(1 - \Pr(\mathsf{correctpw}))$$

$$\begin{aligned} &\leq \Pr(\mathsf{correctpw}) + \frac{1}{2}(1 - \Pr(\mathsf{correctpw})) \\ &= \frac{1}{2} + \frac{\Pr(\mathsf{correctpw})}{2}. \end{aligned}$$

Therefore $\mathsf{Adv}^{\mathrm{ake}}_{P_8}(\mathcal{A}) \leq \Pr(\mathsf{correctpw})$. The theorem follows from this, the bound on $\Pr(\mathsf{correctpw})$ from above, and Claims 5.2 through 5.9. ▌

## 6 Conclusion

We have proven that the PAK-Z+ protocol is a secure augmented password-authenticated key exchange protocol in the random oracle model. This is the only such protocol *proven* secure that is being considered for the IEEE P1363.2 standard.

## 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 *WET-ICE'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] P. MacKenzie. The PAK Suite. DIMACS Tech Report 2002-46.

[35] 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.

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

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

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

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

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

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

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

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

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

Figure 2: 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 $sk = H_4(\langle C, S, m, \mu, \sigma, \gamma'\rangle)$.