Two-Server Password-Only Authenticated Key Exchange

Jonathan Katz[1⋆], Philip MacKenzie[2], Gelareh Taban[3], and Virgil Gligor[3]

[1] Dept. of Computer Science, University of Maryland
jkatz@cs.umd.edu
[2] DoCoMo USA Labs, USA
philmac@docomolabs-usa.com
[3] Dept. of Electrical and Computer Engineering, University of Maryland
{gelareh,gligor}@eng.umd.edu

Abstract. Typical protocols for password-based authentication assume a single server which stores all the information (e.g., the password) necessary to authenticate a user. Unfortunately, an inherent limitation of this approach (assuming low-entropy passwords are used) is that the user's password is exposed if this server is ever compromised. To address this issue, a number of schemes have been proposed in which a user's password information is shared among multiple servers, and these servers cooperate in a threshold manner when the user wants to authenticate.

We show here a two-server protocol for this task assuming public parameters available to everyone in the system (as well as the adversary). Ours is the first provably-secure two-server protocol for the important *password-only* setting (in which the user need remember only a password, and not the servers' public keys), and is the first two-server protocol (in any setting) with a proof of security in the standard model.

1 Introduction

A well-known fact in the context of designing authentication systems is that human users/clients typically choose "weak", low-entropy passwords from a relatively small space of possibilities. Unfortunately, protocols designed and proven secure for the case when clients use *cryptographic* (i.e., high-entropy) secrets are generally insecure when passwords (i.e., low-entropy secrets) are used instead; this is so because these protocols are typically not resistant to *off-line dictionary attacks* in which an eavesdropping adversary derives information about the password from observed transcripts of login sessions. In recent years, much attention has focused on designing password-based authenticated key-exchange protocols resistant to such attacks. (We remark that *on-line dictionary attacks* — in which an adversary simply attempts to log-in repeatedly, trying each possible password — cannot be prevented by cryptographic means but can be dealt with using other methods outside the scope of this work.)

⋆ Supported by NSF CAREER award 0447075 and Trusted Computing grant 0310751.

Means of protecting against off-line dictionary attacks in a single-server setting were first suggested by Gong, et al. [23] in a "hybrid", PKI-based model in which users are assumed to know the server's public key in addition to a password. Bellovin and Merritt [5] were the first to suggest protocols for what we term *password-only authenticated key exchange* (PAKE), where the clients are required to "store" (i.e., remember) *only* a short password and no additional information. These initial works (and others [6, 25, 30, 38]) were relatively informal and did not provide definitions or proofs of security. More recently, formal definitions and provably-secure protocols for the "hybrid" model have been given [24, 7], followed soon thereafter by formal models for the password-only setting [1, 8, 22] and associated protocols with proofs of security in the random oracle/ideal cipher models[1] [1, 8, 31] and in the standard model [28, 22, 20, 27]. (The protocols of [28, 20, 27] assume some public information which is available to all parties. Note, however, that since this information can be hard-coded into implementations of the protocol, clients do not need to memorize or store any high-entropy, cryptographic information as they are required to do in the PKI-based setting.)

Although the above protocols protect against off-line dictionary attacks, they do nothing to mitigate the concern that an adversary might obtain users' passwords via *server compromise*. Such attacks represent a serious threat since they are potentially cost-effective (in that an adversary might be able to obtain thousands of users' passwords by corrupting a single, poorly-protected server) and users frequently utilize the same password at multiple sites. Unfortunately, it is easy to show the *impossibility* of protecting against server compromise when a single server holds the necessary information to authenticate a user (assuming the only secret information held by the user is a low-entropy password). To protect against server compromise, Ford and Kaliski [19] thus proposed a *threshold* protocol in the PKI-based model, in which the authentication functionality is distributed across n servers who must all cooperate to authenticate a user. Their protocol remains secure (and, in particular, an adversary learns nothing about users' passwords other than what it learns from its on-line password guesses) as long as $n-1$ or fewer servers are compromised. Jablon [26] subsequently suggested a protocol with similar functionality in the password-only setting. Neither of these works, however, include rigorous definitions or proofs of security.

A number of provably-secure protocols for threshold password-based authentication have recently appeared. We summarize what is known:

- MacKenzie, et al. [35] showed a protocol in the "hybrid", PKI-based setting which requires only t (out of n) servers to cooperate in order to authenticate a user, for any values of t, n (of course, security is only obtained as long as

[1] In the random oracle model [2], parties are assumed to have "black-box" access to a random function. (The ideal cipher model assumes that parties have "black-box" access to a random keyed permutation, an even stronger assumption.) In practice, the random oracle is instantiated with a cryptographic hash function. It is known [10], however, that protocols secure in the random oracle model may not be secure for *any* choice of hash function.

$t-1$ or fewer servers are compromised). They prove security for their protocol in the random oracle model.

- Di Raimondo and Gennaro [16] proposed a protocol in the password-only setting with a proof of security in the standard model. (A second protocol given in their paper, which we will not discuss further, achieves the weaker functionality in which the same session key is computed by all servers.) However, their protocol requires less than $1/3$ of the servers to be compromised (i.e., they require $n > 3t$) and thus does not give a solution for the two-server case.[2] We remark that, in general, threshold cryptosystems for the two-party case do not follow immediately from threshold solutions for the case of general n; see, e.g., the work of [21, 33, 34, 32, 17] in this regard.
- Brainard, et al. [9] have developed a two-server protocol (called "Nightingale" and being shipped by RSA Security), a variant of which has been proven secure in the random oracle model [37]. These protocols assume the PKI-based setting: as stated in the papers, the protocols assume a "secure channel" between the client and the server(s) which would in practice be implemented using public-key techniques such as SSL.

1.1 Our Contributions

We show here a two-server protocol for password-only authenticated key exchange, with proof of security in the standard model. Ours is the first *provably-secure* two-server protocol in the password-only setting, and is the first two-server protocol (in any setting) with a proof of security in the standard model.

Our protocol extends and builds upon the (non-threshold) protocol of Katz-Ostrovsky-Yung [28] and a more efficient variant of this protocol — called, for brevity, KOY* — described by Canetti, *et al.* [11] (we also introduce an additional modification that makes the protocol even more efficient). In Section 3 we describe a "basic" two-server protocol which is secure against a passive (i.e., "honest-but-curious") adversary who has access to the entire state of one of the servers throughout its execution of the protocol, but cannot cause this server to deviate from its prescribed behavior. We believe this protocol is interesting in its own right (when the assumption on adversarial behavior is warranted), and anyway the basic protocol and its proof of security serve as a useful prelude to our second result. In Section 4 we show how to modify the basic protocol so as to achieve security against an *active* adversary who may arbitrarily deviate from the protocol. The changes we make consist of having the servers (efficiently) prove to each other that their computations were performed according to the protocol. Here, we make use of techniques developed by MacKenzie [32] for performing these proofs efficiently even in a concurrent setting.

The protocols we construct are relatively efficient. Each party in the basic two-server protocol performs roughly twice the amount of work as in the KOY*

[2] The approach in their paper does not extend to the case $t \geq n/3$. The authors mention (without details) that "[i]t is possible to improve the fault-tolerance to $t < n/2\ldots$", but even this would not imply a two-server solution.

protocol. For the protocol secure against active adversaries, the work of the client stays the same but the work of the servers increases by a factor of roughly 2–4.

2 Definitions and Preliminaries

We assume the reader is familiar with the model of [1] (building on [3, 4]) for password-based key exchange in the single-server case. Here, we generalize their model and present formal definitions for the case of two-server protocols. While the model presented here is largely equivalent to the model proposed by MacKenzie, et al. [35] (with the main difference that we do not assume a PKI), we can simplify matters a bit since we focus on the two-server setting exclusively. For convenience we describe the model for the case of a "passive" adversary first and then discuss briefly the modifications needed in the case of "active" adversaries. (As discussed below, in both the "passive" and "active" cases the adversary is free to interfere with all communication between the client and the servers. These cases only differ in the power of the adversary to control the actions of the corrupted servers: specifically, a "passive" adversary is unable to control the actions of corrupted servers, whereas a "active" adversary can.)

We first present a general overview of the system as we imagine it. For simplicity only, we assume that every client C in the system shares its password pw with exactly two servers A and B. In this case we say that servers A and B are *associated with* C. (Note that a single server may be associated with multiple clients.) In addition to holding password shares, these servers may also be provisioned with arbitrary other information which is *not* stored by C. Any such information is assumed to be provisioned by some incorruptible, central mechanism (a system administrator, say) at the outset of the protocol. Note that this does *not* represent an additional assumption or restriction in practice: the servers *must* minimally be provisioned with correct password shares anyway, and there is no reason why additional information cannot be provided to the servers at that time (in particular, the servers' password shares are already high-entropy values, and the servers have no restriction — as the client does — on the amount of information they can store). An (honest) execution of a password-based key-exchange protocol between client C and associated servers A and B should result in the client holding independent session keys $\mathsf{sk}_{C,A}, \mathsf{sk}_{C,B}$, and servers A and B holding $\mathsf{sk}_{A,C}$ and $\mathsf{sk}_{B,C}$, respectively, with $\mathsf{sk}_{C,A} = \mathsf{sk}_{A,C}$ and $\mathsf{sk}_{C,B} = \mathsf{sk}_{B,C}$.

2.1 Passive Adversaries

We first describe the adversarial model under consideration. We assume an adversary who corrupts some servers at the outset of the protocol, such that for any client C at most one of the servers associated with C is corrupted. In the case of a passive adversary, a corrupted server continues to operate according to the protocol, but the adversary may monitor its internal state. Following [35, 16], we make the (largely conceptual) assumption that there exists a single *gateway* which is the only entity that *directly* communicates with the clients. This gateway essentially acts as a "router", splitting messages from the clients to each of the two associated servers and aggregating messages from these servers to the client; we also allow the gateway to perform some simple, publicly-computable operations. Introduction of this gateway is not strictly necessary; however, it enables a simplification of the formal model and also provides a straightforward way to quantify the number of on-line attacks made by an adversary.

During the course of the protocol, then, there may potentially be three types of communication: between the clients and the gateway, between the servers and the gateway, and between the servers themselves. We assume the adversary has the ability to eavesdrop on all of these. We further assume that the client-gateway communication is under the full control of the adversary, and thus the adversary can send messages of its choice to either of these entities, or may tamper with, delay, refuse to deliver, etc. any messages sent between clients and the gateway. On the other hand, in the case of a passive adversary we assume that the server-gateway and server-server communication is determined entirely by the protocol itself (i.e., a corrupted server follows the protocol exactly as specified).[3] In addition, the adversary is assumed to see the entire internal state of any corrupted servers throughout their execution of the protocol. With the above in mind, we proceed to the formal definitions.

Participants, passwords, and initialization. We assume a fixed set of protocol participants (also called principals) each of which is either a client $C \in \mathsf{Client}$ or a server $S \in \mathsf{Server}$, where Client and Server are disjoint. Each $C \in \mathsf{Client}$ is assumed to have a password pw_C chosen uniformly and independently from a "dictionary" of size N.[4] (In fact, we further simplify matters and assume that passwords are chosen from the set $\{1, \ldots, N\}$.) As noted earlier, we make the simplifying assumption that each client shares his password with exactly two other servers (and no more). If client C shares his password with the distinct servers A, B, then A (resp., B) holds a *password share* $pw_{C,A}$ (resp., $pw_{C,B}$); the mechanism for generating these shares depends on the protocol itself. We also allow each server to hold information in addition to these password shares. For example, as described in footnote 3, two servers associated with a particular client may be provisioned with a shared, symmetric key. As we have already discussed, the initialization phase during which this information is provisioned is assumed to be carried out by some trusted authority. Any information stored by a corrupted server is available to the adversary.

[3] For the case of a passive adversary, the assumption that the adversary cannot tamper with the server-server communication is easy to realize via standard use of message authentication codes or signatures (as the servers can store long-term keys for this purpose). The assumption that the adversary cannot tamper with the server-gateway communication is essentially for convenience/definitional purposes only, as the adversary can anyway tamper with client-gateway communication (and the gateway processes messages in a well-defined and predictable way).

[4] As in other work, though, our proof of security may be adapted to handle arbitrary dictionaries and arbitrary distributions over these dictionaries.

In general, additional information can be generated during the initialization phase. For example, in the "hybrid" password/PKI model [24, 7] public/secret key pairs are generated for each server and the secret key is given as input to the appropriate server, while the public key is provided to the appropriate client(s). For the protocol presented here, we require only the weaker requirement of a single set of public parameters which is provided to all parties.

Execution of the protocol. In the real world, a protocol determines how principals behave in response to input from their environment. In the formal model, these inputs are provided by the adversary. Each principal is assumed to be able to execute the protocol multiple times (possibly concurrently) with different partners; this is modeled by allowing each principal to have an unlimited number of *instances* [4, 1] with which to execute the protocol. We denote instance i of principal U as Π_U^i. A given instance may be used only once. The adversary is given oracle access to these different instances; furthermore, each instance maintains (local) state which is updated during the course of the experiment. In particular, each instance Π_U^i has associated with it the following variables, initialized as NULL or FALSE (as appropriate) during the initialization phase:

- sid_U^i, pid_U^i, and sk_U^i are variables containing the *session id*, *partner id*, and *session key(s)* for an instance, respectively. A client's partner id will be a set of two servers; a server's partner id will be a single client (viewed as a set for notational convenience). For C a client, sk_C^i consists of a pair $\mathsf{sk}_{C,A}^i, \mathsf{sk}_{C,B}^i$, where these are the keys shared with servers A and B, respectively. A server instance Π_S^i with partner C has only a single session key $\mathsf{sk}_{S,C}^i$.
- term_U^i and acc_U^i are boolean variables denoting whether a given instance has terminated or accepted, respectively. state_U^i records any state necessary for execution of the protocol by Π_U^i.

As highlighted earlier, the adversary is assumed to have complete control over all communication between the client and the gateway. This is modeled via access to *oracles* which are essentially as in [1] and are described formally in the full version of this paper. Briefly, these include various Send oracles modeling "active" attacks in which the adversary tampers with communication between the client and the servers; an $\mathsf{Execute}$ oracle modeling passive eavesdropping attacks; a Reveal oracle which models possible leakage of session keys; and a Test oracle used to define security.

Sessions ids, partnering, correctness, and freshness. Session ids in our protocol are defined in a natural way, which then allows us to define notions of partnering, correctness, and freshness. Due to lack of space the details appear in the full version.

Advantage of the adversary. Informally, the adversary succeeds if it can guess the bit b used by the Test oracle on a "fresh" instance associated with a non-corrupted participant. Formally, we say an adversary A *succeeds* if it makes a single query $\mathsf{Test}(U, U', i)$ regarding a fresh key $\mathsf{sk}_{U,U'}^i$, and outputs a single bit b' with $b' = b$ (recall that b is the bit chosen by the Test oracle). We denote this event by Succ. Note that restricting the adversary to making its Test query regarding a fresh key is necessary for a meaningful definition of security. The advantage of adversary A in attacking protocol P is then given by:

$$\mathsf{Adv}_{A,P}(k) \stackrel{\text{def}}{=} 2 \cdot \Pr[\mathsf{Succ}] - 1,$$

where the probability is taken over the random coins used by the adversary as well as the random coins used during the course of the experiment.

It remains to define a secure protocol, as a PPT adversary can always succeed by trying all passwords one-by-one in an on-line impersonation attack. Informally, a protocol is secure if this is the best an adversary can do. Formally, we define in the full version what it means for an instance to represent an *on-line attack* (which, in particular, will not include instances used in $\mathsf{Execute}$ queries). The number of on-line attacks bounds the number of passwords the adversary could have tried in an on-line fashion, motivating the following definition:

Definition 1. *Protocol P is a* secure two-server protocol for password-only authenticated key-exchange *if there exists a constant c such that, for all dictionary sizes N and for all* PPT *adversaries A making at most $Q(k)$ on-line attacks and corrupting at most one server associated with each client, there exists a negligible function $\varepsilon(\cdot)$ such that $\mathsf{Adv}_{A,P}(k) \leq c \cdot Q(k)/N + \varepsilon(k)$.*

Of course, we would optimally like to achieve $c = 1$; however, as in previous definitions and protocols [38, 1, 22] we will allow $c \neq 1$ as well. The proof of security for our protocol shows that we achieve $c = 2$, indicating that the adversary can (essentially) do no better than guess *two* passwords during each on-line attack.

Explicit mutual authentication. The above definition captures the requirement of *implicit* authentication only (and the protocol we present here achieves only implicit authentication). Using standard techniques, however, it is easy to add explicit authentication to any protocol achieving implicit authentication.

2.2 Active Adversaries

The key difference in the active case is that the adversary may now cause any corrupted servers to deviate in an arbitrary way from the actions prescribed by the protocol. Thus, if a server is corrupted the adversary controls all messages sent from this server to the gateway as well as messages sent from this server to any other server. As in the passive case, however, we continue to assume that communication between the gateway and any non-corrupted servers (as well as communication between two non-corrupted servers) is *not* under adversarial control. See footnote 3 for the rationale behind these conventions.

3 A Protocol Secure Against Passive Adversaries

3.1 Description of the Protocol

We assume the reader is familiar with the decisional Diffie-Hellman (DDH) assumption [15], strong one-time signature schemes, and the Cramer-Shoup encryption scheme [14] with labels. A high-level depiction of the protocol is given in Figures 1–3, and a more detailed description, as well as some informal discussion about the protocol, follows.

Initialization. During the initialization phase, we assume the generation of public parameters (i.e., a common reference string) which are then made available to all parties. For a given security parameter k, the public parameters will contain a group $\mathbb{G}$ (written multiplicatively) having prime order q with $|q| = k$; we assume the hardness of the DDH problem in $\mathbb{G}$. Additionally, the parameters include random generators $g_1, g_2, g_3, h, c, d \in \mathbb{G}^{\times}$ and a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ chosen at random from a collision-resistant hash family.

As part of the initialization, each server S is provisioned with an El Gamal [18] public-/secret-key pair (pk_S, sk_S), where $pk_S = g_1^{sk_S}$. If A and B are associated with the same client C, then A (resp., B) is given pk_B (resp., pk_A). We stress that, in contrast to the PKI-based model, the client is not assumed or required to know the public keys of any of the servers.

Passwords and password shares are provisioned in the following way: a password pw_C is chosen randomly for each client C and we assume that this password can be mapped in a one-to-one fashion to $\mathbb{Z}_q$. If A and B are the servers associated with a client C, then password shares $pw_{A,C}, pw_{B,C} \in \mathbb{Z}_q$ are chosen uniformly at random subject to $pw_{A,C} + pw_{B,C} = pw_C \bmod q$, with $pw_{A,C}$ given to server A and $pw_{B,C}$ given to server B. In addition, both A and B are given $\mathsf{Com}_{A,C}, \mathsf{Com}'_{A,C}$, $\mathsf{Com}_{B,C}$, and $\mathsf{Com}'_{B,C}$, where:

$$\begin{aligned}
\mathsf{Com}_{A,C} &\stackrel{\text{def}}{=} \mathsf{ElG}_{g_3}(g_1^{pw_{A,C}}) = \left(g_1^{r_a},\ g_3^{r_a} g_1^{pw_{A,C}}\right) \\
\mathsf{Com}'_{A,C} &\stackrel{\text{def}}{=} \mathsf{ElG}_{pk_A}(g_1^{pw_{A,C}}) \\
\mathsf{Com}_{B,C} &\stackrel{\text{def}}{=} \mathsf{ElG}_{g_3}(g_1^{pw_{B,C}}) = \left(g_1^{r_b},\ g_3^{r_b} g_1^{pw_{B,C}}\right) \\
\mathsf{Com}'_{B,C} &\stackrel{\text{def}}{=} \mathsf{ElG}_{pk_B}(g_1^{pw_{B,C}}).
\end{aligned}$$

Note that different public keys are used. Server A (resp., server B) is additionally given the randomness r_a (resp., r_b) used to construct $\mathsf{Com}_{A,C}$ (resp., $\mathsf{Com}_{B,C}$).

Protocol execution. At a high level one can view our protocol as two executions of the KOY* protocol, one between the client and server A (using server B to assist with the authentication), and one between the client and server B (using server A to assist with the authentication). Note that the assistance of the other server is necessary since the password information is split between the two servers. For efficiency, the signature and verification for the two executions are combined, and shares of the El Gamal encryption of the password sent by the servers (i.e., (F, G) in Figure 1) are fixed and stored by the servers.

When a client with password pw_C wants to initiate an execution of the protocol, this client computes Cramer-Shoup "encryptions" of pw_C for each of the two servers. In more detail (cf. Figure 1), the client begins by running a key-generation algorithm for a one-time signature scheme, yielding VK and SK. The client next chooses random $r_1 \in \mathbb{Z}_q$ and computes $A_a = g_1^{r_1}$, $B_a = g_2^{r_1}$, and $C_a = h^{r_1} \cdot pw_C$. The client then computes $\alpha_a = H(\mathit{Client}|\mathsf{VK}|A_a|B_a|C_a)$ and

Public: $\mathbb{G}$; $g_1, g_2, g_3, h, c, d \in \mathbb{G}^{\times}$; $H : \{0,1\}^* \to \mathbb{Z}_q$

Client

$$
\begin{array}{rc}
(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^k) & \\
r_1, r_2 \leftarrow \mathbb{Z}_q & \\
A_a := g_1^{r_1}, \quad B_a := g_2^{r_1}, \quad C_a := h^{r_1} \cdot g_1^{pw_C} & \\
\alpha_a := H(\mathit{Client}\,|\mathsf{VK}|A_a|B_a|C_a), \quad D_a := (cd^{\alpha_a})^{r_1} & \\
A_b := g_1^{r_2}, \quad B_b := g_2^{r_2}, \quad C_b := h^{r_2} \cdot g_1^{pw_C} & \\
\alpha_b := H(\mathit{Client}\,|\mathsf{VK}|A_b|B_b|C_b), \quad D_b := (cd^{\alpha_b})^{r_2} & \xrightarrow{\left\langle \mathit{Client}, \mathsf{VK}, \begin{array}{c} A_a, B_a, C_a, D_a \\ A_b, B_b, C_b, D_b \end{array} \right\rangle} \\
 & \xleftarrow{\langle E_a, E_b, F, G \rangle} \\
x_1, y_1, x_2, y_2 \leftarrow \mathbb{Z}_q & \\
K_a := g_1^{x_1} g_3^{y_1} & \\
K_b := g_1^{x_2} g_3^{y_2} & \\
\sigma \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{msg}_1|\mathsf{msg}_2|K_a|K_b) & \\
\mathsf{sk}_{C,A} := E_a^{r_1} F^{x_1} (G/g_1^{pw_C})^{y_1} & \xrightarrow{\langle K_a, K_b, \sigma \rangle} \\
\mathsf{sk}_{C,B} := E_b^{r_2} F^{x_2} (G/g_1^{pw_C})^{y_2} &
\end{array}
$$

Fig. 1. An execution of the protocol from the client's point of view.

sets $D = (cd^{\alpha_a})^{r_1}$. This exact procedure is then carried out again, using an independent random value $r_2 \in \mathbb{Z}_q$. The client sends

$$\mathsf{msg}_1 \stackrel{\text{def}}{=} \langle \mathit{Client}, \mathsf{VK}, A_a, B_a, C_a, D_a, A_b, B_b, C_b, D_b \rangle$$

to the gateway as the first message of the protocol. Note that this corresponds to two independent "encryptions" of pw_C using the label $\mathit{Client}\,|\mathsf{VK}$.

When the gateway receives msg_1 from a client, the gateway simply forwards this message to the appropriate servers. The servers act symmetrically, so for simplicity we simply describe the actions of server A (cf. Figure 2). Upon receiving msg_1, server A sends "shares" of (1) two values of the form $g_1^x g_2^y h^z (cd^{\alpha})^w$ (for $\alpha \in \{\alpha_a, \alpha_b\}$), one for server A and one for server B, and (2) an El Gamal encryption of pw_C. In more detail, server A chooses random $x_a, y_a, z_a, w_a \in \mathbb{Z}_q$ and computes $E_{a,1} = g_1^{x_a} g_2^{y_a} h^{z_a} (cd^{\alpha_a})^{w_a}$. It also chooses random $x'_a, y'_a, z'_a, w'_a \in \mathbb{Z}_q$ and computes $E_{b,1} = g_1^{x'_a} g_2^{y'_a} h^{z'_a} (cd^{\alpha_b})^{w'_a}$. Finally, it sets (F_a, G_a) equal to $\mathsf{Com}_{A,C}$ (which, recall, is an El Gamal encryption of $g_1^{pw_{A,C}}$ using "public key" g_3 and randomness r_a). The message $\langle E_{a,1}, E_{b,1}, F_a, G_a \rangle$ is sent to the gateway.

The gateway combines the values it receives from the servers by multiplying them component-wise. This results in a message $\mathsf{msg}_2 = \langle E_a, E_b, F, G \rangle$ which

Public: $\mathbb{G}$; $g_1, g_2, g_3, h, c, d \in \mathbb{G}^{\times}$; $H : \{0,1\}^* \to \mathbb{Z}_q$

Gateway

$$\boxed{\text{Receive } \mathsf{msg}_1 = \left\langle \mathit{Client}, \mathsf{VK}, \begin{matrix} A_a, B_a, C_a, D_a \\ A_b, B_b, C_b, D_b \end{matrix} \right\rangle}$$

Server A — **Server** B

$\xleftarrow{\mathsf{msg}_1}$ $\xrightarrow{\mathsf{msg}_1}$

Server A	Server B
Compute α_a, α_b	Compute α_a, α_b
$\begin{matrix} x_a, y_a, z_a, w_a \\ x'_a, y'_a, z'_a, w'_a \end{matrix} \longleftarrow \mathbb{Z}_q$	$\begin{matrix} x_b, y_b, z_b, w_b \\ x'_b, y'_b, z'_b, w'_b \end{matrix} \longleftarrow \mathbb{Z}_q$
$E_{a,1} := g_1^{x_a} g_2^{y_a} h^{z_a} (cd^{\alpha_a})^{w_a}$	$E_{b,2} := g_1^{x_b} g_2^{y_b} h^{z_b} (cd^{\alpha_b})^{w_b}$
$E_{b,1} := g_1^{x'_a} g_2^{y'_a} h^{z'_a} (cd^{\alpha_b})^{w'_a}$	$E_{a,2} := g_1^{x'_b} g_2^{y'_b} h^{z'_b} (cd^{\alpha_a})^{w'_b}$
$(F_a, G_a) := \mathsf{Com}_{A,C}$	$(F_b, G_b) := \mathsf{Com}_{B,C}$
$\xrightarrow{\langle E_{a,1}, E_{b,1}, F_a, G_a \rangle}$	$\xleftarrow{\langle E_{b,2}, E_{a,2}, F_b, G_b \rangle}$

$$E_a := E_{a,1} \cdot E_{a,2}$$
$$E_b := E_{b,1} \cdot E_{b,2}$$
$$F := F_a \cdot F_b$$
$$G := G_a \cdot G_b$$
$$\boxed{\text{Send } \mathsf{msg}_2 = \langle E_a, E_b, F, G \rangle}$$
$$\boxed{\text{Receive } \langle K_a, K_b, \sigma \rangle}$$

if $(\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1 | \mathsf{msg}_2 | K_a | K_b, \sigma) = 0)$ terminate

Server A	Server B
$\xleftarrow{\langle K_a, K_b \rangle}$	$\xrightarrow{\langle K_a, K_b \rangle}$
perform $\mathsf{Compute}$ to obtain X_a	perform $\mathsf{Compute}$ to obtain X_b
$\mathsf{sk}_{A,C} := A_a^{x_a} B_a^{y_a} C_a^{z_a} D_a^{w_a} K_a^{r_a} X_a$	$\mathsf{sk}_{B,C} := A_b^{x_b} B_b^{y_b} C_b^{z_b} D_b^{w_b} K_b^{r_b} X_b$

Fig. 2. Execution of the protocol from the servers' points of view (client-gateway messages are boxed). The $\mathsf{Compute}$ protocol is depicted in Figure 3.

is sent to the client, and for which (1) neither server knows the representation of E_a with respect to $g_1, g_2, h, (cd^{\alpha_a})$ (and similarly for E_b with respect to $g_1, g_2, h, (cd^{\alpha_b})$), and (2) the values (F, G) form an El Gamal encryption of the client's password pw_C (with respect to public key g_3).

Upon receiving msg_2, the client proceeds as follows (cf. Figure 1): it chooses random values $x_1, y_1, x_2, y_2 \in \mathbb{Z}_q$ and computes $K_a = g_1^{x_1} g_3^{y_1}$ and $K_b = g_1^{x_2} g_3^{y_2}$. It then computes a signature σ on the "message" $\mathsf{msg}_1 | \mathsf{msg}_2 | K_a | K_b$ using the secret key SK that it had previously generated. It sends the message $\langle K_a, K_b, \sigma \rangle$

Public: $\mathbb{G};\ g_1, g_2, g_3, h, c, d \in \mathbb{G}^\times;\ H : \{0,1\}^* \to \mathbb{Z}_q$

Server A (sk_A, pk_B)		**Server** B (sk_B, pk_A)
$M_1 \leftarrow \mathsf{ElG}_{pk_A}(g_1^{-z_a})$	$\xrightarrow{M_1}$	$M'_1 \leftarrow \mathsf{ElG}_{pk_B}(g_1^{-z_b})$
	$\xleftarrow{M'_1}$	
$T_A := A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a} K_b^{r_a}$		$T_B := A_a^{x'_b} B_a^{y'_b} C_a^{z'_b} D_a^{w'_b} K_a^{r_b}$
$M_2 \leftarrow (M'_1)^{pw_{A,C}} \times \left(\mathsf{Com}'_{B,C}\right)^{-z'_a}$ $\times \mathsf{ElG}_{pk_B}(g_1^{-z'_a pw_{A,C}} \cdot T_A)$		$M'_2 \leftarrow (M_1)^{pw_{B,C}} \times \left(\mathsf{Com}'_{A,C}\right)^{-z'_b}$ $\times \mathsf{ElG}_{pk_A}(g_1^{-z'_b pw_{B,C}} \cdot T_B)$
	$\xrightarrow{M_2}$	
$X_a := g_1^{-z_a \cdot pw_{A,C}} \cdot \frac{M'_2[2]}{(M'_2[1])^{sk_A}}$	$\xleftarrow{M'_2}$	$X_b := g_1^{-z_b \cdot pw_{B,C}} \cdot \frac{M_2[2]}{(M_2[1])^{sk_B}}$

Fig. 3. The $\mathsf{Compute}$ protocol. See text for a description of the notation used.

to the gateway and computes session keys

$$\mathsf{sk}_{C,A} := E_a^{r_1} F^{x_1} (G/g_1^{pw_C})^{y_1}$$
$$\mathsf{sk}_{C,B} := E_b^{r_2} F^{x_2} (G/g_1^{pw_C})^{y_2}.$$

Upon receiving $\langle K_a, K_b, \sigma\rangle$ from the client, the gateway first verifies that σ is a valid signature on the "message" $\mathsf{msg}_1|\mathsf{msg}_2|K_a|K_b$ with respect to the key VK.[5] If verification fails, the gateway terminates this instance of this protocol and send a special message to the servers indicating this fact. Otherwise, it sends K_a, K_b to each of the servers. The servers then execute the $\mathsf{Compute}$ protocol (cf. Figure 3 and described next) in order to compute their session keys.

Before describing the $\mathsf{Compute}$ protocol, we introduce some notation for manipulation of El Gamal ciphertexts. Given a message $m \in \mathbb{G}$ and a public key (i.e., group element) pk, we let $M \leftarrow \mathsf{ElG}_{pk}(m)$ denote the act of choosing a random $r \in \mathbb{Z}_q$ and setting $M = (g_1^r, pk^r m)$. We let $M[1]$ refer to the first component of this ciphertext, and let $M[2]$ refer to the second. Note that if sk is the corresponding secret key (i.e., $pk = g_1^{sk}$), then we have $m = \frac{M[2]}{M[1]^{sk}}$.

If M, M' are two El Gamal ciphertexts (encrypted with respect to the same public key pk), then we let $M \times M'$ denote $(M[1] \cdot M'[1],\ M[2] \cdot M'[2])$. Note that if M is an encryption of m and M' is an encryption of m', then $M \times M'$ is an encryption of $m \cdot m'$. For $x \in \mathbb{Z}_q$, we let M^x denote the ciphertext $(M[1]^x, M[2]^x)$. Here, the resulting ciphertext is an encryption of m^x.

With this in mind, we now describe the $\mathsf{Compute}$ protocol. Since the protocol is symmetric, we simply describe it from the point of view of server A. This server

[5] We remark that this can just as easily be done by the servers; allowing the gateway to perform this check, however, simplifies things slightly.

sets M_1 to be an El Gamal encryption (with respect to pk_A) of $g_1^{-z_a}$. It then sends M_1 to server B, who computes

$$M_2' \leftarrow M_1^{pw_{B,C}} \times \left(\mathsf{Com}'_{A,C}\right)^{-z_b'} \times \mathsf{ElG}_{pk_A}(g_1^{-z_b' pw_{B,C}} A_a^{x_b'} B_a^{y_b'} C_a^{z_b'} D_a^{w_b'} K_a^{r_b})$$

and sends this value back to server A (recall that r_b is the randomness used to construct $\mathsf{Com}_{B,C}$). Finally, server A decrypts M_2' and multiplies the result by $g_1^{-z_a \cdot pw_{A,C}}$ to obtain X_a. Note that

$$X_a = g_1^{-(z_a+z_b')\cdot pw_C} \cdot \left(A_a^{x_b'} B_a^{y_b'} C_a^{z_b'} D_a^{w_b'} K_a^{r_b}\right), \tag{1}$$

using the fact that $pw_C = pw_{A,C} + pw_{B,C} \bmod q$.

Although omitted in the above description, we assume that the client and the gateway always verify that incoming messages are well-formed, and in particular that all appropriate components of the various messages indeed lie in $\mathbb{G}$ (we assume that membership in $\mathbb{G}$ can be efficiently verified).

Correctness. One can easily verify correctness of the above protocol. Due to space limitations, we omit the detailed computations.

Security against passive adversaries. A proof of the following theorem appears in the full version of this paper.

Theorem 1. *Assuming (1) the DDH problem is hard for $\mathbb{G}$; (2)* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is a secure one-time signature scheme; and (3) H is collision-resistant, the protocol of Figures 1–3 is a secure two-server protocol for password-only authenticated key exchange in the presence of a passive adversary (in particular, it satisfies Definition 1 with $c = 2$).*

4 Handling Active Adversaries

Here, we describe the necessary changes to the protocol in order to handle active adversaries. At a high level, these changes can be summarized as follows:

Proofs of correctness. We require servers to give *proofs of correctness* for their actions during the $\mathsf{Compute}$ protocol. We stress that we use only the fact that these are *proofs* (and not *proofs of knowledge*) and therefore we do not require any rewinding in our proof of security. This is crucial, as it enables us to handle concurrent executions of the protocol. Nevertheless, as part of the proofs of correctness we will have the servers encrypt certain values with respect to (additional) per-server public keys provisioned during protocol initialization. This will, in fact, enable extraction of certain values from the adversary during the security proof.

Commitments to password shares. The protocol as described in the previous section already assumes that each server is provisioned with appropriate El Gamal encryptions of the password share of the other server. We will use these shares (along with the proofs of correctness discussed earlier) to "force" a corrupted server to use the appropriate password share in its computations.

Simulating proofs for non-corrupted servers. During the course of the proof of security it will be necessary for non-corrupted servers to deviate from the prescribed actions of the protocol, yet these servers must give "valid" proofs of correctness to possibly corrupted servers. We cannot rely on "standard" use of zero-knowledge proofs in our setting, since (1) this would require rewinding which we explicitly want to avoid, and (2) potential malleability issues arise due to the fact that a corrupted server may be giving its proof of correctness at the same time a non-corrupted server is giving such a proof (this is so even if we force sequential executions of the proofs of correctness within any particular instance, since multiple instances may be simultaneously active). To enable simulatability we rely on techniques of MacKenzie [32] described in greater detail below.

4.1 Detailed Description of Changes to the Protocol

We first discuss the necessary modifications to the initialization phase (this is all in addition to the provisioned values already discussed in Section 3.1): (1) each server S is given a random triple $\mathsf{triple}_S = (U_{S,1}, U_{S,2}, U_{S,3})$ of elements chosen uniformly at random from $\mathbb{G}$. Furthermore, (2) each server S is given $\mathsf{triple}_{S'}$ for any server S' for which there is a client with associated servers S, S'.

We will next describe the necessary changes to the protocol itself. In what follows, we will use witness-indistinguishable Σ-protocols (with negligible soundness error[6]) [12] of various predicates and it will be useful to develop some notation. If Ψ represents a predicate (defined over some public values), we let $\Sigma[\Psi]$ denote a Σ-protocol for this predicate. If Ψ_1, Ψ_2 are two predicates, then we let $\Sigma[\Psi_1 \vee \Psi_2]$ denote a Σ-protocol for the "or" of these predicates. We remark that given Σ-protocols for Ψ_1 and Ψ_2, it is easy to combine these so as to obtain a Σ-protocol for $\Psi_1 \vee \Psi_2$ [13]. We define the predicate DDH_S, for any server S with $\mathsf{triple}_S = (U_1, U_2, U_3)$, as follows:

$$\mathsf{DDH}_S(U_1, U_2, U_3) \stackrel{\text{def}}{=} [\exists x, y \text{ s.t. } U_1 = g_1^x \wedge U_2 = g_1^y \wedge U_3 = g_1^{xy}]\,;$$

i.e., DDH_S denotes the predicate asserting that triple_S is a Diffie-Hellman triple.

Given the above, the protocol is modified in the following ways:

Initial server message. To construct the second message of the protocol (i.e., msg_2), the servers first construct $E_{a,1}, E_{b,1}, E_{b,2}$, and $E_{a,2}$ as in Figure 2. The servers also choose random $\mathsf{nonce}_A, \mathsf{nonce}_B \in \{0,1\}^k$. Server A sends to the gateway the message $\langle \mathsf{nonce}_A, E_{a,1}, E_{b,1}, \mathsf{Com}_{A,C}, \mathsf{Com}_{B,C}\rangle$. Similarly, server B sends to the gateway the message $\langle \mathsf{nonce}_B, E_{b,2}, E_{a,2}, \mathsf{Com}_{A,C}, \mathsf{Com}_{B,C}\rangle$. The gateway verifies that $\mathsf{Com}_{A,C}$ and $\mathsf{Com}_{B,C}$ (as sent by the two servers) are identical: if so, it constructs the outgoing message as in Figure 2 but also including

[6] From now on, "Σ-protocol" means a witness-indistinguishable Σ-protocol with negligible soundness error.

$\mathsf{nonce}_A, \mathsf{nonce}_B$; if not, the gateway sends a special abort message to each of the servers. (The client will sign all of msg_2 — including the nonces — exactly as in Figure 1, and this signature will be verified by the gateway as in Figure 2.)

The Compute protocol. After the gateway receives msg_3 and verifies the signature as in Figure 2, it forwards the values $E_{a,1}, E_{b,1}$ (resp., $E_{b,2}, E_{a,2}$) to server B (resp., server A) in addition to forwarding (K_a, K_b) as before. The $\mathsf{Compute}$ protocol is then modified as follows (we describe the changes from the point of view of server A, but they are applied symmetrically to server B): In the first phase, in addition to computing $M_1 \leftarrow \mathsf{ElG}_{pk_A}(g_1^{-z_a})$, server A computes

$$v_{x_a,y_a,z_a,w_a} := A_a^{x_a} B_a^{y_a} C_a^{z_a} D_a^{w_a} \quad \text{and} \quad V_{x_a,y_a,z_a,w_a} \leftarrow \mathsf{ElG}_{pk_A}(v_{x_a,y_a,z_a,w_a})$$

and sends these values to B. Define the predicate Ψ_1 as follows:

$$\Psi_1 \stackrel{\text{def}}{=} \left[\exists x_a, y_a, z_a, w_a, r, \tilde{r} \text{ s.t. } : \begin{array}{r} E_{a,1} = g_1^{x_a} g_2^{y_a} h^{z_a} (cd^{\alpha_a})^{w_a} \\ M_1 = \left(g_1^r,\ pk_A^r \cdot g_1^{-z_a}\right) \\ V_{x_a,y_a,z_a,w_a} = \left(g_1^{\tilde{r}},\ pk_A^{\tilde{r}} \cdot A_a^{x_a} B_a^{y_a} C_a^{z_a} D_a^{w_a}\right) \end{array} \right].$$

Server A then acts as a prover in the protocol $\Sigma[\Psi_1 \vee \mathsf{DDH}_A]$. Meanwhile, A acts as a verifier in the symmetric Σ-protocol being given (possibly concurrently) by server B. If B's proof fails, then A aborts immediately.

In the second phase of the $\mathsf{Compute}$ protocol, in addition to computing M_2 as in Figure 3, server A also computes

$$\begin{array}{cc} v_{z'_a} := g_1^{z'_a} & v_{x'_a,y'_a,z'_a,w'_a} := A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a} \\ V_{z'_a} \leftarrow \mathsf{ElG}_{pk_A}(v_{z'_a}) & V_{x'_a,y'_a,z'_a,w'_a} \leftarrow \mathsf{ElG}_{pk_A}(v_{x'_a,y'_a,z'_a,w'_a}) \end{array}$$

and sends these values to B. Define the predicate Ψ_2 as follows:

$$\Psi_2 \stackrel{\text{def}}{=}$$

$$\left[\begin{array}{l} \exists x'_a, y'_a, z'_a, w'_a, \\ r_a, pw_{A,C}, r, \tilde{r}, \hat{r} \text{ s.t.} \end{array} : \begin{array}{r} E_{b,1} = g_1^{x'_a} g_2^{y'_a} h^{z'_a} (cd^{\alpha_b})^{w'_a} \\ V_{z'_a} = \left(g_1^r,\ pk_A^r \cdot g_1^{z'_a}\right) \\ V_{x'_a,y'_a,z'_a,w'_a} = \left(g_1^{\tilde{r}},\ pk_A^{\tilde{r}} \cdot A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a}\right) \\ M_2 = (M'_1)^{pw_{A,C}} \times (\mathsf{Com}'_{B,C})^{-z'_a} \\ \times \left(g_1^{\hat{r}},\ pk_B^{\hat{r}} \cdot g_1^{-z'_a \cdot pw_{A,C}} A_b^{x'_a} B_b^{y'_a} C_b^{z'_a} D_b^{w'_a} K_b^{r_a}\right) \\ \mathsf{Com}_{A,C} = \left(g_1^{r_a},\ g_3^{r_a} g_1^{pw_{A,C}}\right) \end{array} \right].$$

Server A then acts as a prover in the protocol $\Sigma[\Psi_2 \vee \mathsf{DDH}_A]$. Meanwhile, A acts as a verifier in the symmetric Σ-protocol being given (possibly concurrently) by server B. If B's proof fails, then A aborts without computing a session key.

Relatively efficient Σ-protocols for the above predicates can be constructed using standard techniques, and we omit further details.

Security against active adversaries. A proof of the following theorem appears in the full version of this paper.

Theorem 2. *With the modifications described above and under the same assumptions as in Theorem 1, we obtain a secure two-server protocol for password-only authenticated key exchange in the presence of an active adversary.*

References

1. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. *Adv. in Cryptology — Eurocrypt 2000*, LNCS vol. 1807, Springer-Verlag, pp. 139–155, 2000.
2. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *Proc. 1st ACM Conference on Computer and Communications Security*, ACM, pp. 62–73, 1993.
3. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *Adv. in Cryptology — Crypto 1993*, LNCS vol. 773, Springer-Verlag, pp. 232–249, 1994.
4. M. Bellare and P. Rogaway. Provably-Secure Session Key Distribution: the Three Party Case. *27th ACM Symposium on Theory of Computing (STOC)*, ACM, pp. 57–66, 1995.
5. S.M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. *IEEE Symposium on Research in Security and Privacy*, IEEE, pp. 72–84, 1992.
6. S.M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise. *1st ACM Conf. on Computer and Comm. Security*, ACM, pp. 244–250, 1993.
7. M. Boyarsky. Public-Key Cryptography and Password Protocols: The Multi-User Case. *7th Ann. Conf. on Computer and Comm. Security*, ACM, pp. 63–72, 1999.
8. V. Boyko, P. MacKenzie, and S. Patel. Provably-Secure Password-Authenticated Key Exchange Using Diffie-Hellman. *Adv. in Cryptology — Eurocrypt 2000*, LNCS vol. 1807, Springer-Verlag, pp. 156–171, 2000.
9. J. Brainard, A. Juels, B. Kaliski, and M. Szydlo. Nightingale: A New Two-Server Approach for Authentication with Short Secrets. *12th USENIX Security Symp.*, pp. 201–213, 2003.
10. R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. *J. ACM* 51(4): 557–594, 2004.
11. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally-Composable Password Authenticated Key Exchange. Eurocrypt 2005, to appear.
12. R. Cramer. Modular Design of Secure Yet Practical Cryptographic Protocols. PhD Thesis, CWI and University of Amsterdam, 1996.
13. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. *Adv. in Cryptology — Crypto 1994*, LNCS vol. 839, Springer-Verlag, pp. 174–187, 1994.
14. R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Chosen Ciphertext Attack. *Adv. in Cryptology — Crypto 1998*, LNCS vol. 1462, Springer-Verlag, pp. 13–25, 1998.
15. W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory* 22(6): 644–654, 1976.
16. M. Di Raimondo and R. Gennaro. Provably Secure Threshold Password-Authenticated Key Exchange. *Adv. in Cryptology — Eurocrypt 2003*, LNCS vol. 2656, Springer-Verlag, pp. 507–523, 2003.
17. Y. Dodis, M. Krohn, D. Mazieres, and A. Nicolosi. Proactive Two-Party Signatures for User Authentication. NDSS 2003.
18. T. El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory* 31: 469–472, 1985.
19. W. Ford and B.S. Kaliski. Server-Assisted Generation of a Strong Secret from a Password. *Proc. 5th IEEE Intl. Workshop on Enterprise Security*, 2000.

20. R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. *Adv. in Cryptology — Eurocrypt 2003*, LNCS vol. 2656, Springer-Verlag, pp. 524–543, 2003.
21. N. Gilboa. Two-Party RSA Key Generation. *Adv. in Cryptology — Crypto 1999*, LNCS vol. 1666, Springer-Verlag, pp. 116–129, 1999.
22. O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. *Adv. in Cryptology — Crypto 2001*, LNCS vol. 2139, Springer-Verlag, pp. 408–432, 2001.
23. L. Gong, T.M.A. Lomas, R.M. Needham, and J.H. Saltzer. Protecting Poorly-Chosen Secrets from Guessing Attacks. *IEEE J. on Selected Areas in Communications* 11(5): 648–656, 1993.
24. S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. *ACM Trans. Information and System Security* 2(3): 230–268, 1999.
25. D. Jablon. Strong Password-Only Authenticated Key Exchange. *ACM Computer Communications Review* 26(5): 5–20, 1996.
26. D. Jablon. Password Authentication Using Multiple Servers. *RSA Cryptographers' Track 2001*, LNCS vol. 2020, Springer-Verlag, pp. 344–360, 2001.
27. S. Jiang and G. Gong. Password Based Key Exchange With Mutual Authentication. *Workshop on Selected Areas of Cryptography (SAC)*, 2004.
28. J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. *Adv. in Cryptology — Eurocrypt 2001*, LNCS vol. 2045, Springer-Verlag, pp. 475–494, 2001.
29. T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham. Reducing Risks from Poorly-Chosen Keys. *ACM Operating Systems Review* 23(5): 14–18, 1989.
30. S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. *Proc. of the Security Protocols Workshop*, LNCS 1361, Springer-Verlag, pp. 79–90, 1997.
31. P. MacKenzie, S. Patel, and R. Swaminathan. Password-Authenticated Key Exchange Based on RSA. *Adv. in Cryptology — Asiacrypt 2000*, LNCS 1976, Springer-Verlag, pp. 599–613, 2000.
32. P. MacKenzie. An Efficient Two-Party Public-Key Cryptosystem Secure against Adaptive Chosen-Ciphertext Attack. *Public Key Cryptography (PKC) 2003*, LNCS vol. 2567, Springer-Verlag, pp. 47–61, 2003.
33. P. MacKenzie and M. Reiter. Networked Cryptographic Devices Resilient to Capture. *IEEE Security and Privacy*, 2001.
34. P. MacKenzie and M. Reiter. Two-Party Generation of DSA Signatures. *Adv. in Cryptology — Crypto 2001*, LNCS vol. 2139, Springer-Verlag, pp. 137–154, 2001.
35. P. MacKenzie, T. Shrimpton, and M. Jakobsson. Threshold Password-Authenticated Key Exchange. *Adv. in Cryptology — Crypto 2002*, LNCS vol. 2442, Springer-Verlag, pp. 385–400, 2002.
36. V. Shoup. A Proposal for an ISO Standard for Public-Key Encryption, version 2.1. Draft, 2001. Available at `http://eprint.iacr.org/2001/112`.
37. M. Szydlo and B. Kaliski. Proofs for Two-Server Password Authentication. *RSA Cryptographers' Track 2005*, LNCS vol. 3376, Springer-Verlag, pp. 227–244, 2005.
38. T. Wu. The Secure Remote Password Protocol. *Proc. Internet Society Symp. on Network and Distributed System Security*, pp. 97–111, 1998.