A New Framework for Efficient Password-Based Authenticated Key Exchange

Adam Groce* Jonathan Katz*

Abstract

Protocols for password-based authenticated key exchange (PAKE) allow two users who share only a short, low-entropy password to agree on a cryptographically strong session key. The challenge in designing such protocols is that they must be immune to *off-line dictionary attacks* in which an eavesdropping adversary exhaustively enumerates the dictionary of likely passwords in an attempt to match a password to the set of observed transcripts.

To date, few general frameworks for constructing PAKE protocols in the standard model are known. Here, we abstract and generalize a protocol by Jiang and Gong to give a new methodology for realizing PAKE without random oracles, in the common reference string model. In addition to giving a new approach to the problem, the resulting construction offers several advantages over prior work. We also describe an extension of our protocol that is secure within the universal composability (UC) framework and, when instantiated using El Gamal encryption, is more efficient than a previous protocol of Canetti et al.

1 Introduction

Protocols for password-based authenticated key exchange (PAKE) enable two parties who share a short, low-entropy password to agree on a cryptographically strong session key. The difficulty in this setting is to design protocols preventing *off-line dictionary attacks* whereby an eavesdropping adversary exhaustively enumerates passwords, attempting to match the correct password to observed protocol executions. Roughly, a PAKE protocol is "secure" if off-line attacks are of no use and the best attack is an *on-line* dictionary attack whereby the adversary actively impersonate the honest user with each possible password. This is the best that can be hoped for in the password-only setting; more importantly, on-line attacks can be detected and defended against.

PAKE protocols are fascinating from a theoretical perspective, as they can be viewed as a means of "bootstrapping" a common cryptographic key from the (essentially) minimal setup assumption of a short, shared secret. PAKE protocols are also important in practice, since passwords are perhaps the most common and widely-used means of authentication.

There is, by now, a substantial body of research focused on the design of PAKE protocols. Early work [16] (see also [17]) considered a "hybrid" model where users share public keys in addition to a password; we are concerned here with the more challenging "password-only" setting. Bellovin and Merritt [7] initiated research in this direction, and presented a PAKE protocol with heuristic arguments for its security. It was not until several years later that formal models for PAKE were

*Dept. of Computer Science, University of Maryland. Email: {agroce, jkatz}@cs.umd.edu. Research supported by NSF grant #0627306.

developed [4, 8, 15], and provably secure PAKE protocols were shown in the random oracle/ideal cipher models [4, 8, 23].

To date, there are only a few general approaches for constructing PAKE protocols in the *standard model* (i.e., without random oracles). Goldreich and Lindell [15] constructed the first such protocol in the "plain model" where there is no additional setup. Unfortunately, their protocol is inefficient, and furthermore does not tolerate concurrent executions by the same party. Nguyen and Vadhan [24] show some simplifications and efficiency improvements to the Goldreich-Lindell protocol, but at the expense of achieving a qualitatively weaker notion of security. The results of Barak et al. [3] also imply a protocol for password-based key exchange, albeit in the common reference string model. Unfortunately, these protocols are all inefficient in terms of communication, computation, and round complexity, and yield nothing close to a practical instantiation.

Katz, Ostrovsky, and Yung (KOY) [20] demonstrated the first *efficient* PAKE protocol with a proof of security in the standard model. Their protocol was later abstracted by Gennaro and Lindell (GL) [14], who gave a general framework that encompasses the original KOY protocol as a special case. These protocols are secure even under concurrent executions by the same party, but require a *common reference string* (CRS). While this may be less appealing than the "plain model," reliance on a CRS does not appear to be a serious drawback in practice for the deployment of PAKE, where common parameters can be hard-coded into an implementation of the protocol.

Surprisingly, the KOY/GL framework remains the only *general* framework for constructing *efficient* PAKE protocols in *the standard model*, and almost all subsequent work on efficient PAKE in the standard model [14, 10, 19, 13, 2, 21] can be viewed as extending and building on the KOY/GL framework. The one exception is a paper by Jiang and Gong [18] that shows an efficient PAKE protocol in the standard model (assuming a common reference string) based on the decisional Diffie-Hellman assumption. Our work is to theirs as the work of Gennaro-Lindell [14] is to that of Katz-Ostrovsky-Yung [20]; namely, we present a (new) framework for PAKE that is obtained by suitably abstracting and generalizing the Jiang-Gong protocol. In so doing, we gain the same benefits as in the previous case: i.e., we get a simple-to-describe, generic protocol with a clean and intuitive proof of security, and derive (as corollaries to our work) new variants of the Jiang-Gong protocol based on different cryptographic assumptions.

Compared to PAKE protocols built using the KOY/GL framework we obtain several advantages:

Weaker assumptions. From a foundational point of view, our new framework relies on potentially *weaker* assumptions than the KOY/GL framework. Specifically, we require (1) a CCA-secure encryption scheme, and (2) a *CPA*-secure encryption scheme with an associated smooth projective hash function [12]. In contrast, the KOY/GL framework requires[1] a *CCA*-secure encryption scheme with an associated smooth projective hash function, something not known to follow from the previous assumptions.[2]

In particular, our results imply a more efficient — not to mention simpler — construction of PAKE from lattice-based assumptions as compared to the recent work of Katz and Vaikuntanathan [21]. (Most of the complexity in [21] arises from the construction of a lattice-based CCA-secure encryption scheme with an associated smooth projective hash function.)

Better efficiency. The above directly translates into better efficiency for protocols constructed

[1] Technically speaking, it requires a non-malleable, non-interactive *commitment scheme* with an associated smooth projective hash function, but all known constructions of this primitive are in fact CCA-secure encryption schemes.

[2] Cramer and Shoup [12] show that a CPA-secure encryption scheme Π with a smooth projective hash function implies a CCA-secure scheme Π', but there is no guarantee that Π' will *itself* admit a smooth projective hash function.

using the new framework, since the CCA-secure encryption scheme we use need not admit a smooth projective hash function. (E.g., restricting our attention to the decisional Diffie-Hellman assumption, our framework can use the Kurosawa-Desmedt [22] scheme instead of Cramer-Shoup encryption [12]. Significant efficiency improvements would also be obtained when basing the protocol on lattice assumptions, as discussed above.) The new framework also avoids using digital signatures (though Gennaro [13] shows how this can be avoided when using the KOY/GL framework as well).

Mutual authentication. The framework yields PAKE protocols achieving (explicit) *mutual authentication* in three rounds. In contrast, the KOY protocol and its extensions require four rounds in order to achieve mutual authentication. (This advantage was already noted in [18].)

We also show how our framework can be extended to yield a protocol that securely realizes the PAKE functionality within the universal composability (UC) framework [9], with all the above advantages carrying over to this setting. To the best of our knowledge, the only prior *efficient* PAKE protocols in the UC framework are those of Canetti et al. [10] and Abdalla et al. [1] (in the random oracle model); instantiating our framework using El Gamal encryption gives a protocol more efficient than either of these. Of independent interest, we define for the first time a PAKE functionality *with (explicit) mutual authentication* and show that our protocol realizes this stronger functionality. See further discussion in Section 4.1.

1.1 Outline of the Paper

We review definitions for PAKE and smooth projective hashing in Sections 2.1 and 2.2, respectively; these are fairly standard and can be skipped by readers already familiar with these notions. In Section 3 we describe the new framework for PAKE and prove it secure with respect to the standard definition. We discuss the extension of our protocol to the UC framework in Section 4, beginning with a discussion of the PAKE functionality *with explicit mutual authentication* in Section 4.1. We believe the latter to be of independent interest.

2 Definitions

2.1 Password-Based Authenticated Key Exchange

We present the definition of Bellare, Pointcheval, and Rogaway [4], based on prior work of [5, 6]. The treatment here is lifted almost verbatim from [20], except that here we also define mutual authentication but otherwise keep the discussion brief. We denote the security parameter by n.

Participants, passwords, and initialization. Prior to any execution of the protocol there is an initialization phase during which public parameters are established. We assume a fixed set User of protocol participants (also called principals or users). For every distinct $U, U' \in \mathsf{User}$, we assume U and U' share a password $\pi_{U,U'}$. We make the simplifying assumption that each $\pi_{U,U'}$ is chosen independently and uniformly at random from the set $\{1, \ldots, D_n\}$ for some integer D_n that may depend on n. (Our proof of security extends to more general cases.)

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 can execute the protocol multiple times (possibly concurrently) with different partners; this is modeled by allowing each principal to have an unlimited number of *instances*

with which to execute the protocol. We denote instance i of user U as Π_U^i. Each 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 maintains local state that includes the following variables:

- sid_U^i, pid_U^i, and sk_U^i denote the *session id*, *partner id*, and *session key*, respectively. The session id is simply a way to keep track of different executions; we let sid_U^i be the (ordered) concatenation of all messages sent and received by Π_U^i. The partner id denotes the user with whom Π_U^i believes it is interacting; we require $\mathsf{pid}_U^i \neq U$.
- acc_U^i and term_U^i are flags denoting acceptance and termination, respectively.

The adversary's interaction with various instances is modeled via access to the following *oracles*:

- $\mathsf{Send}(U, i, M)$ — This sends message M to instance Π_U^i. This instance runs according to the protocol specification, updating state as appropriate. The output of Π_U^i (i.e., the message sent by the instance) is given to the adversary.
- $\mathsf{Execute}(U, i, U', j)$ — If Π_U^i and $\Pi_{U'}^j$ have not yet been used, this oracle executes the protocol between these instances and gives the resulting transcript to the adversary. This models passive eavesdropping of a protocol execution.
- $\mathsf{Reveal}(U, i)$ — This outputs the session key sk_U^i, modeling leakage of session keys due to, e.g., improper erasure of session keys after use, compromise of a host computer, or cryptanalysis.
- $\mathsf{Test}(U, i)$ — This oracle does not model any real-world capability of the adversary, but is instead used to define security. A random bit b is chosen; if $b = 1$ the adversary is given sk_U^i, and if $b = 0$ the adversary is given a session key chosen uniformly from the appropriate space.

Partnering. Let $U, U' \in \mathsf{User}$. Instances Π_U^i and $\Pi_{U'}^j$ are *partnered* if: (1) $\mathsf{sid}_U^i = \mathsf{sid}_{U'}^j \neq \text{NULL}$; and (2) $\mathsf{pid}_U^i = U'$ and $\mathsf{pid}_{U'}^j = U$.

Correctness. To be viable, a key-exchange protocol must satisfy the following notion of correctness: if Π_U^i and $\Pi_{U'}^j$ are partnered then $\mathsf{acc}_U^i = \mathsf{acc}_{U'}^j = \text{TRUE}$ and $\mathsf{sk}_U^i = \mathsf{sk}_{U'}^j$, i.e., they both accept and conclude with the same session key.

Advantage of the adversary. Informally, the adversary can succeed in two ways: (1) if it guesses the bit b used by the Test oracle (this implies secrecy of session keys), or (2) if it causes an instance to accept without there being a corresponding partner (this implies mutual authentication). Defining this formally requires dealing with several technicalities.

We first define *freshness*. Instance Π_U^i is *fresh* unless one of the following is true at the conclusion of the experiment: (1) the adversary queried $\mathsf{Reveal}(U, i)$; or (2) the adversary queried $\mathsf{Reveal}(U', j)$, where $\Pi_{U'}^j$ and Π_U^i are partnered.

We also define a notion of *semi-partnering*. Instances Π_U^i and $\Pi_{U'}^j$ are *semi-partners* if they are partners, or if the following holds: (1) the (non-NULL) session ids sid_U^i and $\mathsf{sid}_{U'}^j$ agree except possibly for the final message, and $\mathsf{pid}_U^i = U'$ and $\mathsf{pid}_{U'}^j = U$. This relaxed definition is needed to rule out the trivial attack where an adversary forwards all protocol messages except the final one.

An adversary $\mathcal{A}$ *succeeds* if either:

1. $\mathcal{A}$ makes a single query $\mathsf{Test}(U, i)$ to a fresh instance Π_U^i, and outputs a bit b' with $b' = b$ (recall that b is the bit chosen by the Test oracle).

2. At the end of the experiment, there is an instance Π_U^i that accepts but is not semi-partnered with any other instance. (I.e., this is a violation of mutual authentication.)

We denote the event that the adversary succeeds by Succ. The *advantage* of $\mathcal{A}$ in attacking protocol Π is $\mathsf{Adv}_{\mathcal{A},\Pi}(k) \stackrel{\text{def}}{=} 2 \cdot \Pr[\mathsf{Succ}] - 1$, where the probability is taken over the random coins used by the adversary and during the course of the experiment (including the initialization phase).

It remains to define a secure protocol. A probabilistic polynomial-time (PPT) adversary can always succeed with probability 1 by trying all passwords one-by-one; this is possible since the size of the password dictionary is small. Informally, a protocol is secure if this is the best an adversary can do. Formally, an instance Π_U^i represents an *on-line attack* if, at some point, the adversary queried $\mathsf{Send}(U, i, *)$. The number of on-line attacks represents a bound on the number of passwords the adversary could have tested in an on-line fashion.

Definition 1 Protocol Π is a secure PAKE protocol with explicit mutual authentication if, for all dictionary sizes $\{D_n\}$ and for all PPT adversaries $\mathcal{A}$ making at most $Q(n)$ on-line attacks, there exists a negligible function $\mathsf{negl}(\cdot)$ such that $\mathsf{Adv}_{\mathcal{A},\Pi}(n) \leq Q(n)/D_n + \mathsf{negl}(n)$. ♢

2.2 Smooth Projective Hashing

Smooth projective hash functions were introduced by Cramer and Shoup [11]; we follow (and adapt) the treatment of Gennaro and Lindell [14]. Rather than aiming for utmost generality, we tailor the definitions to our eventual application.

Fix a CPA-secure public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ and an efficiently recognizable message space $\mathcal{D}$ (that will correspond to the dictionary of passwords in our application to PAKE). We assume the encryption scheme defines a notion of *ciphertext validity* such that (1) validity of a ciphertext (with respect to pk) can be determined efficiently using pk, and (2) honestly generated ciphertexts are valid.

For the rest of the discussion, fix a key pair (pk, sk) as output by $\mathsf{Gen}(1^n)$ and let $\mathcal{C}$ denote the set of valid ciphertexts with respect to pk. Define sets $X, \{L_m\}_{m\in\mathcal{D}}$, and L as follows. First, set

$$X = \{(C, m) \mid C \in \mathcal{C};\ m \in \mathcal{D}\}.$$

For $m \in \mathcal{D}$ let $L_m = \{(C, m) \mid \mathsf{Dec}_{sk}(C) = m\} \subset X$; i.e., L_m is the set of ciphertext/message pairs. Define $L = \bigcup_{m\in\mathcal{D}} L_m$. Note that for any C there is at most one $m \in \mathcal{D}$ for which $(C, m) \in L$.

Smooth projective hash (SPH) functions. A *smooth projective hash (SPH) function* is a collection of keyed functions $\{H_k : X \to \{0,1\}^n\}_{k\in K}$, along with a *projection function* $\alpha : K \times \mathcal{C} \to S$, satisfying notions of *correctness* and *smoothness*:

Correctness: If $x = (C, m) \in L$ then the value of $H_k(x)$ is determined by $\alpha(k, C)$ and x (in a sense we will make precise below).

Smoothness: If $x \in X \setminus L$ then the value of $H_k(x)$ is statistically close to uniform given $\alpha(k, C)$ and x (assuming k is chosen uniformly in K).

Formally, an SPH function is defined by a sampling algorithm that, given pk, outputs $(K, \mathcal{H} = \{H_k : X \to \{0,1\}^n\}_{k\in K}, S, \alpha : K \times \mathcal{C} \to S)$ such that:

1. There are efficient algorithms for (1) sampling a uniform $k \in K$, (2) computing $H_k(x)$ for $k \in K$ and $x \in X$, and (3) computing $\alpha(k, C)$ for all $k \in K$ and $C \in \mathcal{C}$.

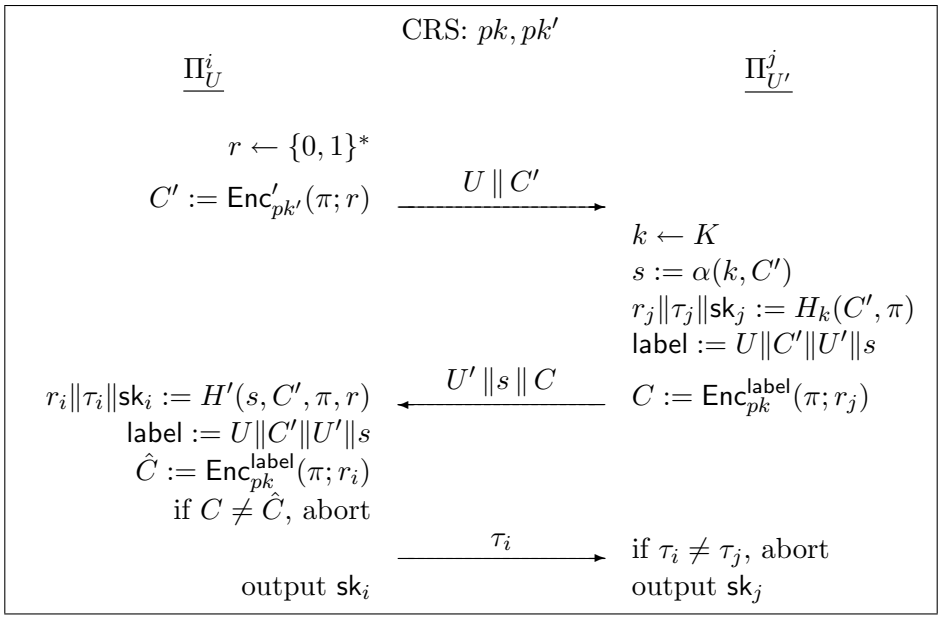

Figure 1: An honest execution of the protocol. The shared password $\pi_{U,U'}$ is denoted by π.

2. For $x = (C, m) \in L$, the value of $H_k(x)$ is determined by $\alpha(k, C)$. Specifically, there is an efficient algorithm H' that takes as input $s = \alpha(k, C)$ and $\bar{x} = (C, m, r)$ (where r is such that $C = \mathsf{Enc}_{pk}(m; r)$) and satisfies $H'(s, \bar{x}) = H_k(x)$.

3. For any $x = (C, m) \in X \setminus L$, the distributions

$$\big\{k \leftarrow K; s = \alpha(k, C) : \big(s, H_k(x)\big)\big\} \quad \text{and} \quad \{k \leftarrow K; s = \alpha(k, C); v \leftarrow \{0,1\}^n : (s, v)\}$$

have statistical difference negligible in n.

3 A New Framework for PAKE

We now describe our new framework for PAKE, obtained as a generalization and abstraction of the specific protocol by Jiang and Gong [18]. In our construction, we use the following primitives:

- A CPA-secure public-key encryption scheme $\Sigma' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ with an associated smooth projective hash function.
- A *labeled* [25] CCA-secure public-key encryption scheme $\Sigma = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Initialization. Our protocol relies on a common reference string (CRS) consisting of public keys pk, pk' for Σ and Σ', respectively, and parameters $(K,\ \mathcal{H} = \{H_k : X \to \{0,1\}^n\}_{k \in K},\ S,\ \alpha : K \times \mathcal{C} \to S)$ for an SPH function associated with pk'. As in all other work in the CRS model, no participants need to know the secret keys associated with the public keys in the CRS. Depending on the exact public-key encryption schemes used it is possible that pk, pk' can be generated from a common *random* string.

Protocol execution. A high-level depiction of the protocol is given in Figure 1. When a client instance Π_U^i wants to authenticate to the server instance $\Pi_{U'}^j$, the client first chooses a random tape r and then computes an encryption $C' := \mathsf{Enc}'_{pk'}(\pi; r)$ of the shared password π. The client then sends $U \| C'$ to the server.

Upon receiving the message $U \| C'$, the server proceeds as follows. It chooses a random hash key $k \leftarrow K$ and computes the projection key $s := \alpha(k, C')$. It then computes the hash $H_k(C', \pi)$ using the ciphertext C' it received in the first message and the password π that it shares with U. The result is parsed as a sequence of three bit-strings $r_j, \tau_j, \mathsf{sk}_j$, where τ_j and sk_j have length at least n, and r_j is sufficiently long to be used as the random tape for an encryption using Enc. The server then sets $\mathsf{label} := U \| C' \| U' \| s$ and generates an encryption $C := \mathsf{Enc}_{pk}^{\mathsf{label}}(\pi; r_j)$ of the shared password π, using the label label and the randomness r_j that it previously computed. Finally, P_j sends the message $U' \| s \| C$ back to the client.

Upon receiving $U' \| s \| C$, the client computes the hash using the projected key s and the randomness it used to generate the ciphertext C' in the first round; that is, P_i computes $r_i \| \tau_i \| \mathsf{sk}_i := H'(s, C', \pi, r)$. It sets $\mathsf{label} := U \| C' \| U' \| s$ and computes the ciphertext $\hat{C} := \mathsf{Enc}_{pk}^{\mathsf{label}}(\pi; r_i)$. If $C = \hat{C}$ the server has successfully authenticated to the client, and the client then accepts, sends τ_i to the server, and outputs the session key sk_i. If $C \neq \hat{C}$ then the client aborts.

When the server receives the client's final message τ_i, it checks that $\tau_i = \tau_j$ and aborts if that is not the case. Otherwise the client has successfully authenticated to the server, and the server accepts and outputs the session key sk_j.

Correctness is easily verified. If both parties are honest and there is no adversarial interference, then $H'(s, C', \pi, r) = H_k(C', \pi)$ and so it holds that $r_i = r_j$, $\tau_i = \tau_j$, and $\mathsf{sk}_i = \mathsf{sk}_j$. It follows that both parties will accept and output the same session key.

A concrete instantiation. By letting Σ' be the El Gamal encryption scheme (which is well-known to admit an SPH function), and Σ be the Cramer-Shoup encryption scheme (though more efficient alternatives are possible), we recover the Jiang-Gong protocol. Without any optimization, this is about 25% faster than the KOY protocol, and roughly 33% more communication efficient.

3.1 Proof of Security

This section is devoted to a proof of the following theorem:

Theorem 1 *If Σ' is CPA-secure public-key encryption scheme with associated smooth projective hash function, and Σ is a CCA-secure public-key encryption scheme, then the protocol in Figure 1 is a secure PAKE protocol with explicit mutual authentication.*

Proof Fix a PPT adversary $\mathcal{A}$ attacking the protocol. We use a hybrid argument to bound the advantage of $\mathcal{A}$. Let Γ_0 represent the initial experiment, in which $\mathcal{A}$ interacts with the real protocol as defined in the previous section. We define a sequence of experiments $\Gamma_1, \ldots$, and denote the advantage of adversary $\mathcal{A}$ in experiment Γ_i as:

$$\mathsf{Adv}_i(n) \stackrel{\text{def}}{=} 2 \cdot \Pr[\mathcal{A} \text{ succeeds in } \Gamma_i] - 1. \tag{1}$$

We bound the difference between the adversary's advantage in successive experiments, and then bound the adversary's advantage in the final experiment; this gives the desired bound on $\mathsf{Adv}_0(n)$, the adversary's advantage when attacking the real protocol.

Experiment Γ_1. In Γ_1 we modify the way $\mathsf{Execute}$ queries are handled. Namely, in response to a query $\mathsf{Execute}(U, i, U', j)$ we now compute $C' \leftarrow \mathsf{Enc}'_{pk'}(\pi_0)$, where π_0 represents some password not in the dictionary. The remainder of the transcript is computed the same way, and the (common) session key for instances Π^i_U and $\Pi^j_{U'}$ is set to be equal to the session key sk_j computed by the server (cf. Figure 1).

Lemma 1 $|\mathsf{Adv}_0(n) - \mathsf{Adv}_1(n)| \leq \mathsf{negl}(n)$.

Proof This follows in a straightforward way from the CPA-security of encryption scheme Σ'. Construct a PPT adversary $\mathcal{B}$ attacking Σ' as follows: given public key pk', the adversary $\mathcal{B}$ simulates the entire experiment for $\mathcal{A}$ including choosing random passwords for each pair of parties. In response to $\mathsf{Execute}(U, i, U', j)$ queries, $\mathcal{B}$ queries its own "challenge" oracle using as its pair of messages the real password $\pi_{U,U'}$ and the fake password π_0; when it receives in return a ciphertext C' it includes this in the transcript that it returns to $\mathcal{A}$. Note that $\mathcal{B}$ can compute correct sessions keys $\mathsf{sk}^i_U = \mathsf{sk}^j_{U'}$ since the actions of instance $\Pi^j_{U'}$ are simulated exactly as in the real protocol (and so, in particular, $\mathcal{B}$ can compute $\mathsf{sk}^j_{U'}$ exactly as an honest player in the real protocol would). At the end of the experiment, $\mathcal{B}$ outputs 1 iff $\mathcal{A}$ succeeds. The distinguishing advantage of $\mathcal{B}$ is exactly $|\mathsf{Adv}_0(n) - \mathsf{Adv}_1(n)|$, and CPA-security of Σ' yields the lemma. ■

Experiment Γ_2. Here we modify the response to a query $\mathsf{Execute}(U, i, U', j)$ as follows. The first message of the transcript is $U \| C'$, where C' is an encryption of π_0 as in Γ_1. Then $k \leftarrow K$ and $s := \alpha(k, C')$ are generated as before. Now, however, we simply choose $r_j \| \tau_j \| \mathsf{sk}_j$ as a random string of the appropriate length. The ciphertext C is computed as in the real protocol, and the message $U' \| s \| C$ is added to the transcript. The final message of the protocol is $\tau_i = \tau_j$, and the session keys $\mathsf{sk}^i_U, \mathsf{sk}^j_{U'}$ are set equal to sk_j (which, recall, was chosen at random).

Lemma 2 $|\mathsf{Adv}_2(n) - \mathsf{Adv}_1(n)| \leq \mathsf{negl}(n)$.

Proof This follows from the properties of the smooth projective hash function for Σ', since when answering $\mathsf{Execute}$ queries in Γ_1 the hash function $H_k(\cdot)$ is always applied to $(C', \pi) \notin L$, and so the output is statistically close to uniform even conditioned on s. Furthermore, in both Γ_1 and Γ_2 the values $r_i, \tau_i, \mathsf{sk}_i$ used by the client are equal to the values $r_j, \tau_j, \mathsf{sk}_j$ computed by the server. ■

Experiment Γ_3. In experiment Γ_3 we again change how $\mathsf{Execute}$ queries are handled. Namely, we compute the ciphertext C sent in the second round as $C \leftarrow \mathsf{Enc}^{\mathsf{label}}_{pk}(\pi_0)$. (We also remove the check performed by the client, and always have the client accept and output the same session key as the server.)

Lemma 3 $|\mathsf{Adv}_3(n) - \mathsf{Adv}_2(n)| \leq \mathsf{negl}(n)$.

Proof The lemma holds based on the CCA-security of Σ. (In fact, all we rely on here is security of Σ against chosen-plaintext attacks.) The key observation is that in experiment Γ_2, the ciphertext C is encrypted using *truly random* coins r_j. Thus, we can construct a PPT adversary $\mathcal{B}$ attacking Σ as follows: given public key pk, adversary $\mathcal{B}$ simulates the entire experiment for $\mathcal{A}$. In response to $\mathsf{Execute}(U, i, U', j)$ queries, $\mathcal{B}$ queries its own "challenge" oracle using as its pair of messages the real password $\pi_{U,U'}$ and the fake password π_0; when it receives in return a ciphertext C it includes this in the second message of the transcript that it returns to $\mathcal{A}$. Session keys are chosen at random.

At the end of the experiment, $\mathcal{B}$ outputs 1 iff $\mathcal{A}$ succeeds. It is immediate that the distinguishing advantage of $\mathcal{B}$ is $|\mathsf{Adv}_3(n) - \mathsf{Adv}_2(n)|$. CPA-security of Σ' yields the lemma. ■

Note that Execute queries in Γ_3 generate random session keys and transcripts that are independent of the actual passwords of any of the parties.

Experiment Γ_4. In this experiment we will begin to modify the Send oracle. For notational convenience, we let $\mathsf{Send}_0(U, i, U')$ denote a "prompt" message that causes the client instance Π^i_U to initiate the protocol with server U'; let $\mathsf{Send}_1(U', j, U\|C')$ denote sending the first message of the protocol to server instance $\Pi^j_{U'}$; let $\mathsf{Send}_2(U, i, U'\|s\|C)$ denote sending the second message of the protocol to client instance Π^i_U; and let $\mathsf{Send}_3(U', j, \tau)$ denote sending the final message of the protocol to server instance $\Pi^j_{U'}$.

In Γ_4 we now record the secret keys sk, sk' when the public keys in the CRS are generated. Furthermore, in response to the query $\mathsf{Send}_2(U, i, U'\|s\|C)$ we proceed as follows:

- If $\mathsf{pid}^i_U \neq U'$ then Π^i_U aborts as it would in Γ_3. From here on, we assume this is not the case.
- Let $U\|C'$ denote the initial message sent by Π^i_U (i.e., $U\|C'$ is the message that was output in response to the query $\mathsf{Send}_0(U, i, U')$). Then:
  - If $U'\|s\|C$ was output by a previous query $\mathsf{Send}_1(U', \star, U\|C')$ then we say that the message $U'\|s\|C$ is *previously-used* and the experiment continues as in Γ_3.
  - If $U'\|s\|C$ is not previously-used, then we set $\mathsf{label} := U\|C'\|U'\|s$ and compute $\pi := \mathsf{Dec}^{\mathsf{label}}_{sk}(C)$. If $\pi = \pi_{U,U'}$ the adversary is declared successful and the experiment ends. Otherwise, Π^i_U rejects (and outputs no session key, nor sends the final message of the protocol).

Lemma 4 $\mathsf{Adv}_3(n) \leq \mathsf{Adv}_4(n)$.

Proof The only situation in which Γ_4 proceeds differently from Γ_3 occurs when $U'\|s\|C$ is not previously-used but decrypts to the correct password; in this case the adversary is immediately declared successful, so its advantage can only increase. ■

Experiment Γ_5. In experiment Γ_5 we modify the way Send_0 and Send_2 queries are handled. In response to a query $\mathsf{Send}_0(U, i, U')$ we now compute $C' \leftarrow \mathsf{Enc}'_{pk'}(\pi_0)$, where (as before) π_0 denotes a dummy password that is not in the dictionary. When responding to a query $\mathsf{Send}_2(U, i, U'\|s\|C)$, we proceed as follows:

- If $\mathsf{pid}^i_U \neq U'$ we reject as always. From here on, we simply assume this does not occur.
- If $U'\|s\|C$ is previously-used (as defined in experiment Γ_4), then it was output in response to some previous query $\mathsf{Send}_1(U', j, U\|C')$; let $r_j, \tau_j, \mathsf{sk}_j$ be the internal variables used by the server instance $\Pi^j_{U'}$. Then to respond to the current Send_2 query we set $\tau_i := \tau_j$ (and send τ_i as the final message of the protocol), and set the session key for instance Π^i_U to $\mathsf{sk}^i_U := \mathsf{sk}_j$.
- If $U'\|s\|C$ is not previously-used, we respond as in Γ_4: namely, we set $\mathsf{label} := U\|C'\|U'\|s$ and compute $\pi := \mathsf{Dec}^{\mathsf{label}}_{sk}(C)$. If $\pi = \pi_{U,U'}$, the adversary is declared successful and the experiment ends. Otherwise, Π^i_U rejects (and outputs no session key, nor sends the final protocol message).

Lemma 5 $|\mathsf{Adv}_5(n) - \mathsf{Adv}_4(n)| \leq \mathsf{negl}(n)$.

Proof First consider an intermediate experiment Γ'_4, where the Send_2 oracle is modified as described above, but Send_0 still computes C' exactly as in Γ_4. This is simply a syntactic rewriting of Γ_4, and so the adversary's advantage remains unchanged.

We next show that the adversary's advantage can change by only a negligible amount in moving from Γ'_4 to Γ_5. This follows from the CPA-security of Σ'. Namely, we construct an adversary $\mathcal{B}$ who, given public key pk, simulates the entire experiment for $\mathcal{A}$. This includes generation of the CRS, which $\mathcal{B}$ does by generating $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ on its own and letting the CRS be (pk, pk'). In response to Send_0 queries, $\mathcal{B}$ queries its own "challenge" oracle using as its pair of messages the real password $\pi_{U,U'}$ and the dummy password π_0; when it receives in return a ciphertext C' it outputs the message $U \| C'$ to $\mathcal{A}$. Note that $\mathcal{B}$ can still respond to Send_2 queries since knowledge of the randomness used to generate C' is no longer used (in either Γ'_4 or Γ_5). At the end of the experiment, $\mathcal{B}$ determines whether $\mathcal{A}$ succeeds and outputs 1 iff this is the case. The distinguishing advantage of $\mathcal{B}$ is exactly $|\mathsf{Adv}_5(n) - \mathsf{Adv}'_4(n)|$. CPA-security of Σ' yields the lemma. ■

Experiment Γ_6**.** In experiment Γ_6 we introduce a simple modification to the way Send_1 oracle calls are handled. When the adversary queries $\mathsf{Send}_1(U', j, U \| C')$, we now compute $\pi := \mathsf{Dec}'_{sk'}(C')$ (using the secret key sk' that was stored at the time the CRS was generated) and check if $\pi = \pi_{U,U'}$. If so, we declare the adversary successful and end the experiment. Otherwise, the experiment continues as before. All this does is introduce a new way for the adversary to succeed, and so $\mathsf{Adv}_5(n) \leq \mathsf{Adv}_6(n)$.

It may at first appear odd that we allow the adversary to succeed in this way, since Σ' may be completely malleable. Recall, however, that in Γ_5/Γ_6 all ciphertexts C' output in response to Send_0 queries are encryptions of dummy passwords; thus, the condition introduced here will not occur "trivially".

Experiment Γ_7**.** In experiment Γ_7 we again modify the behavior of the Send_1 oracle. In response to a query $\mathsf{Send}_1(U', j, U \| C')$ we check whether $\mathsf{Dec}'_{sk'}(C')$ is equal to $\pi_{U,U'}$ as in experiment Γ_6. If so, the adversary is declared to succeed as before. If not, however, we now choose r_j, τ_j, and sk_j uniformly at random (rather than computing these values as the output of $H_k(C', \pi)$), and then continue as before. In particular, if there is a subsequent Send_3 query using the correct value of τ_j then the server instance $\Pi^j_{U'}$ accepts and outputs the session key $\mathsf{sk}^j_{U'} := \mathsf{sk}_j$.

Lemma 6 $|\mathsf{Adv}_7(n) - \mathsf{Adv}_6(n)| \leq \mathsf{negl}(n)$.

Proof This follows from the properties of the smooth projective hash function for Σ'. Consider a query $\mathsf{Send}_1(U', j, U \| C')$ where $\mathsf{Dec}'_{sk'}(C') \neq \pi_{U,U'}$. In Γ_6, we compute $r_j \| \tau_j \| \mathsf{sk}_j := H_k(C', \pi_{U,U'})$, whereas in Γ_7 we choose r_j, τ_j, and sk_j uniformly at random. Since $(C', \pi_{U,U'}) \notin L$, however, these are statistically close since the adversary only sees the projected key $s := \alpha(k, C')$. ■

The key observation about experiment Γ_7 is that every oracle-generated second-round message contains a ciphertext C that is an encryption of the correct password *using truly random coins.*

Experiment Γ_8**.** For the final experiment, we again modify the response to Send_1 queries; specifically, the ciphertext C is now computed as $C \leftarrow \mathsf{Enc}^{\mathsf{label}}_{pk}(\pi_0)$.

Lemma 7 $|\mathsf{Adv}_8(n) - \mathsf{Adv}_7(n)| \leq \mathsf{negl}(n)$.

Proof The proof relies on the CCA-security of Σ. Construct a PPT adversary $\mathcal{B}$ attacking Σ as follows: given public key pk, adversary $\mathcal{B}$ simulates the entire experiment for $\mathcal{A}$. In response to Send_1 queries, $\mathcal{B}$ queries its own "challenge" oracle using as its pair of messages $\pi_{U,U'}$ and π_0; when it receives in return a ciphertext C, it includes this ciphertext in the message that it outputs to $\mathcal{A}$. To fully simulate the experiment, $\mathcal{B}$ also has to check whether $\mathcal{A}$ succeeds in the course of making a Send_1 or Send_2 query. The former case is easy to handle, since $\mathcal{B}$ knows the secret key sk' corresponding to the public key pk' and can therefore decrypt the necessary ciphertexts on its own. In the latter case $\mathcal{B}$ will have to use its decryption oracle to determine whether $\mathcal{A}$ succeeds or not. It can be verified, however, that $\mathcal{B}$ never has to request decryption of a label/ciphertext pair that it received from its own challenge oracle (this follows from the way we defined "previously-used"). At the end of the experiment, $\mathcal{B}$ outputs 1 iff $\mathcal{A}$ succeeds. The distinguishing advantage of $\mathcal{B}$ is exactly $|\mathsf{Adv}_8(n) - \mathsf{Adv}_7(n)|$. CCA-security of Σ yields the lemma. ■

Bounding the advantage in Γ_8. Consider the different ways for the adversary to succeed in Γ_8:

1. $\mathsf{Send}_1(U', j, U|C')$ is queried, where $\mathsf{Dec}_{sk'}(C') = \pi_{U,U'}$.
2. $\mathsf{Send}_2(U, i, U'\|s\|C)$ is queried, where $U'\|s\|C$ is not previously-used and $\mathsf{Dec}_{sk}^{\mathsf{label}}(C) = \pi_{U,U'}$ for label computed as discussed in experiment Γ_4.
3. The adversary successfully guesses the bit used by the Test oracle.
4. $\mathsf{Send}_3(U', j, \tau)$ is queried, where $\tau = \tau_j$ but τ was not output by any instance partnered with $\Pi_{U'}^j$.

Case 4 occurs with only negligible probability, since τ_j is a uniform n-bit string that is independent of the adversary's view if τ_j was not output by any instance partnered with $\Pi_{U'}^j$. Let $\mathsf{PwdGuess}$ be the event that case 1 or 2 occurs. Since the adversary's view is independent of all passwords until one of these cases occurs, we have $\Pr[\mathsf{PwdGuess}] \leq Q(n)/D_n$. Conditioned on $\mathsf{PwdGuess}$ not occurring, the adversary can succeed only in case 3. But then all session keys defined throughout the experiment are chosen uniformly and independently at random (except for the fact that partnered instances are given identical session keys), and so the probability of success in this case is exactly 1/2. Ignoring case 4 (which we have already argued occurs with only negligible probability), then, we have

$$\begin{aligned}
\Pr[\mathsf{Success}] &= \Pr[\mathsf{Success} \wedge \mathsf{PwdGuess}] + \Pr[\mathsf{Success} \wedge \overline{\mathsf{PwdGuess}}] \\
&\leq \Pr[\mathsf{PwdGuess}] + \Pr[\mathsf{Success} \mid \overline{\mathsf{PwdGuess}}] \cdot (1 - \Pr[\mathsf{PwdGuess}]) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \Pr[\mathsf{PwdGuess}] \\
&\leq \frac{1}{2} + \frac{Q(n)}{2 \cdot D_n},
\end{aligned}$$

and so $\mathsf{Adv}_8(n) \leq Q(n)/D_n$. Lemmas 1–7 imply that $\mathsf{Adv}_0(n) \leq Q(n)/D_n + \mathsf{negl}(n)$ as desired. ■

4 PAKE in the UC Framework

In the UC framework [9], a cryptographic task is specified via an appropriate ideal-world functionality; a secure protocol is defined as one that adequately "mimics" this ideal functionality. More

formally, *protocol* Π *realizes a functionality* $\mathcal{F}$ if for any adversary $\mathcal{A}$ attacking Π in the real world there exists an adversary (or simulator) $\mathcal{S}$ attacking an execution in the ideal world where the parties interact only with $\mathcal{F}$, such that no environment $\mathcal{Z}$ can distinguish between the real-world and ideal-world executions. (We refer to [9] for extensive background, or to [10, Section 5.1] for a condensed discussion specific to the context of PAKE.)

Working in the UC framework offers several advantages. Key-exchange protocols proven secure in the UC framework satisfy strong *composability* properties: in particular, (1) they are guaranteed to remain secure even when run concurrently with any other set of protocols in the network; and (2) session keys generated by any such key-exchange protocol may be securely used by any application calling the protocol as a sub-routine. In addition to the above, Canetti et al. [10] observe several advantages of working in the UC framework that are specific to PAKE. For one, a definition of PAKE in the UC framework automatically handles *arbitrary password distributions* including dependencies between passwords chosen by different parties. The definition also guarantees security in case two honest parties run the protocol with different passwords (e.g., due to mistyping); prior definitions say nothing in that event. Note also that, as proved in [10], the definition of PAKE in the UC framework is at least as strong as what is ensured by Definition 1. We refer the reader to [10] for further discussion.

Canetti et al. [10] observe that PAKE protocols proven secure with respect to Definition 1 do not necessarily realize PAKE in the UC framework. A key issue that arises is that when proving security of a protocol according to Definition 1, the "experiment" may end if the adversary makes a correct password guess. (Indeed, this is exactly what occurs in our proof in the preceding section, cf. Experiment Γ_4 and others.) On the other hand, security in the UC framework requires that *the simulation continue* even in the event a correct password guess occurs.

Organization of this section. We describe our formalization of the PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ in Section 4.1. While we use the definition given in [10] as our starting point, we strengthen the functionality so that it also guarantees *mutual authentication*. (Although mutual authentication is discussed briefly in [10], the suggestion given there for handling the issue does not suffice.) We believe our treatment of mutual authentication is of independent interest.

In Section 4.2 we modify the protocol from Section 3 so as to obtain a protocol that securely realizes (the multi-session extension of) $\mathcal{F}_{\mathsf{pwKE}}$ in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model. (The $\mathcal{F}_{\mathsf{crs}}$-hybrid model provides a way of using a CRS in the UC framework. As shown in [10], PAKE is impossible to realize in the UC framework without some setup assumption.) We prove security of the protocol in Section 4.3.

4.1 Defining the Functionality

Functionality $\mathcal{F}_{\mathsf{pwKE}}$ is given in Figure 2. (Our proof will actually show that our protocol securely realizes the *multi-session extension* $\hat{\mathcal{F}}_{\mathsf{pwKE}}$ of $\mathcal{F}_{\mathsf{pwKE}}$; roughly, this means that multiple executions of the protocol can rely on the same CRS, as would obviously be the case in the real world. We refer to [10] for further details.)

The high-level structure of functionality $\mathcal{F}_{\mathsf{pwKE}}$ follows the approach used in [10], and we briefly describe it here. (Once again, we refer to [10] for more details.) A key feature is that the passwords are provided to the parties by the environment $\mathcal{Z}$. (This, in particular, is what allows the definition to capture arbitrary distributions on passwords.) The parties send their respective passwords to $\mathcal{F}_{\mathsf{pwKE}}$ to initialize a new session; upon initialization, a session is declared "fresh". The ideal-

Functionality $\mathcal{F}_{\mathsf{pwKE}}$

Upon receiving a query $(\mathsf{NewSession}, \mathsf{sid}, P_i, P_j, \pi, \mathsf{role})$ **from party** P_i**:**

Send $(\mathsf{NewSession}, \mathsf{sid}, P_i, P_j, \mathsf{role})$ to $\mathcal{S}$. If this is the first NewSession query, or if this is the second NewSession query and there is a stored session $(P_j, P_i, \pi', \mathsf{role}')$ with $\mathsf{role}' \neq \mathsf{role}$, then store $(P_i, P_j, \pi, \mathsf{role})$ and label this session fresh.

Upon receiving a query $(\mathsf{TestPwd}, \mathsf{sid}, P_i, \pi')$ **from the adversary** $\mathcal{S}$**:**

If there is a stored session of the form $(P_i, P_j, \pi, \mathsf{role})$ that is fresh, then do: If $\pi = \pi'$, label the session compromised and reply to $\mathcal{S}$ with "correct guess". If $\pi \neq \pi'$, label the session interrupted and reply to $\mathcal{S}$ with "wrong guess".

Upon receiving a query $(\mathsf{GetReady}, \mathsf{sid}, P_i)$ **from** $\mathcal{S}$**:**

If there is a stored session of the form $(P_i, P_j, \pi, \mathsf{client})$ that is fresh, then relabel it ready.

Upon receiving a query $(\mathsf{NewKey}, \mathsf{sid}, P_i, \mathsf{sk})$ **from** $\mathcal{S}$**, do:**

If there is a stored session $(P_i, P_j, \pi, \mathsf{role})$ that is not marked completed, then do:

- If the session is compromised, or either P_i or P_j are corrupted, send sk to P_i.
- If the session is interrupted, send $\perp$ to P_i.

If $\mathsf{role} = \mathsf{client}$ (and neither of the above rules were applied) then:

- If there is a stored session $(P_j, P_i, \pi', \mathsf{server}, \mathsf{sk}')$ with $\pi' = \pi$, then send sk' to P_i.

If $\mathsf{role} = \mathsf{server}$ (and none of the above rules were applied) then:

- If there is a stored session $(P_j, P_i, \pi, \mathsf{client})$ labeled ready, then choose $\mathsf{sk}' \leftarrow \{0,1\}^n$, send sk' to P_i, and store $(P_i, P_j, \pi, \mathsf{server}, \mathsf{sk}')$.

If none of the above rules apply, send $\perp$ to P_i. In any case, mark the session $(P_i, P_j, \pi, \mathsf{role})$ as completed.

Figure 2: The PAKE functionality $\mathcal{F}_{\mathsf{pwKE}}$ (with mutual authentication).

world adversary $\mathcal{S}$ can make a TestPwd query to any fresh instance; this models the adversary's ability to carry out on-line password guessing attacks. If the adversary makes a TestPwd query and is correct, the relevant session is marked "compromised" and the adversary can freely choose the session key for that session. (This models the fact that, in this case, the session key is completely known to the adversary.) If the guess is incorrect, the session is marked "interrupted". If the adversary does not make any TestPwd query, then random (but identical) session keys are sent to the two parties involved in the session, assuming the parties use the same password.

Mutual authentication was not required in [10]. As a consequence, in their formulation of $\mathcal{F}_{\mathsf{pwKE}}$ random and independent session keys are sent to the two parties involved in some session if the parties use *different* passwords, as well as for sessions marked interrupted. Here, in contrast, we capture (explicit) mutual authentication by introducing a "ready" state for the client, and then ensuring that (1) a server outputs $\perp$ unless there is a (partnered) client in the ready state, and (2) a client outputs $\perp$ unless there is a (partnered) server that has already output a session key (in which case the client outputs the same key). Moreover, once a client is in the ready state, the adversary can no longer make a TestPwd query to that instance of the client.

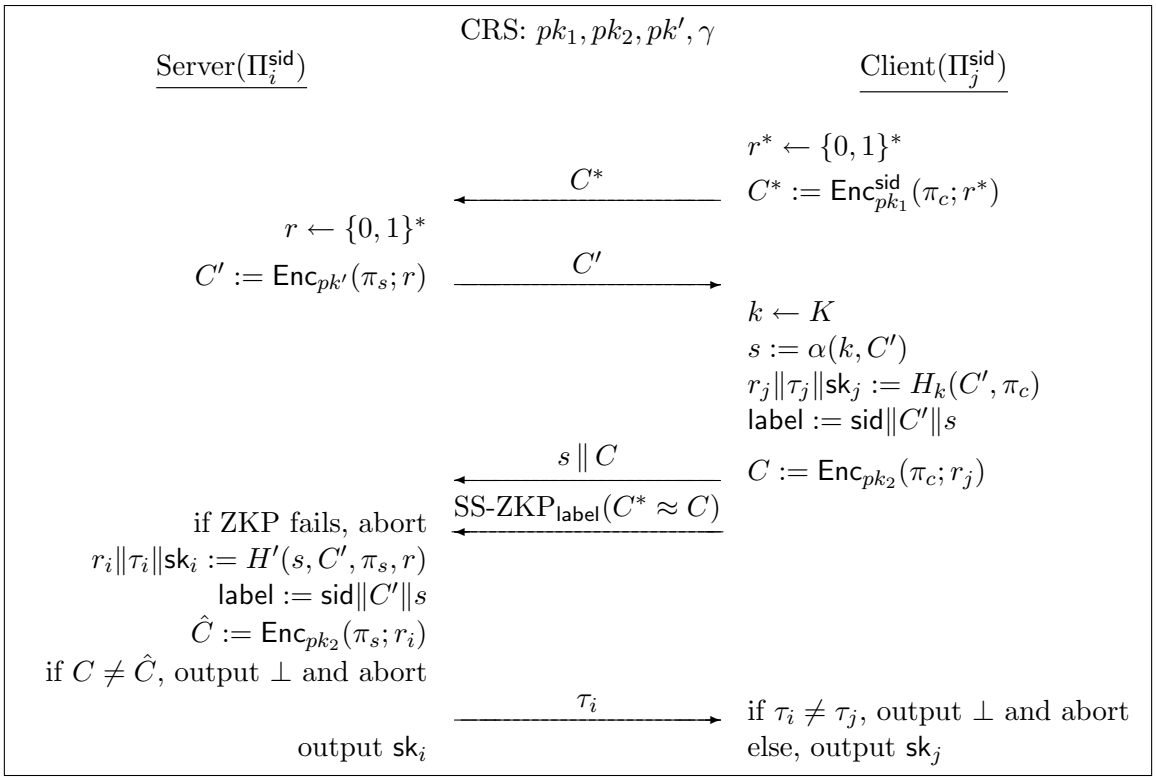

Figure 3: An honest execution of the protocol.

4.2 The Protocol

We modify the protocol from Section 3 in a way analogous to what was done in [10]. Specifically, we add an initial flow that contains an encryption of the password, and also add a *simulation-sound* zero-knowledge proof (SS-ZKP), depending on parameters γ included in the CRS, that the password encrypted in the third round is identical to the password that was encrypted in the first round (this is denoted as "$C^* \approx C$" in Figure 3). More formally, the SS-ZKP using label label proves that (C, C^*) is well-formed in that there exist r^*, r_j, π such that

$$C^* = \mathsf{Enc}_{pk_1}(\pi; r^*) \text{ and } C = \mathsf{Enc}_{pk_2}(\pi; r_j).$$

(Simulation-soundness guarantees that an adversary cannot give a false proof for any new label/statement pair if the statement is invalid.) This change not only allows us to prove security in the UC framework, but allows us to do so without the need for Σ to be CCA-secure. For this reason, we now dispense with the use of labeled encryption, and use the same encryption scheme throughout the protocol. We also make some smaller changes due to the specifics of the UC framework; in particular, we rely on the fact that the parties begin with matching, unique session ids and are aware of each others' identities before starting the protocol. Although not written explicitly, we also assume that if a party ever receives an ill-formed message then it immediately aborts with output $\perp$.

4.3 Proof of Security

4.3.1 Description of the Simulator

To prove that the protocol securely realizes $\mathcal{F}_{\mathsf{pwKE}}$ we must show how to transform any real-world adversary $\mathcal{A}$ to an ideal-world adversary (simulator) $\mathcal{S}$ such that no polynomial-time environment $\mathcal{Z}$ can distinguish between the real- and ideal-world executions. We describe the behavior of $\mathcal{S}$ here, and prove that $\mathcal{S}$ provides a good simulation in the next section.

$\mathcal{S}$ begins by running the key-generation algorithms for Σ, and the simulator for the zero-knowledge proof system, to obtain pk_1, pk_2, pk', and γ along with their respective secret keys. $\mathcal{S}$ uses these as the CRS for $\mathcal{A}$, which it runs as a subroutine. It also chooses a value π_0 that is in the domain of Enc but is assumed for simplicity to be outside the space of possible passwords that $\mathcal{Z}$ can provide to the parties. (A more cumbersome option is to choose π_0 at random.)

When $\mathcal{S}$ receives ($\mathsf{NewSession}$, sid, P_i, P_j, role) from $\mathcal{F}$, then $\mathcal{S}$ begins simulating the protocol on behalf of any uncorrupted parties involved. To do so, $\mathcal{S}$ begins running the protocol as specified except that it uses π_0 for the password, and uses the zero-knowledge simulator to generate proofs on behalf of an uncorrupted client. Execution of $\mathcal{S}$ then proceeds as follows.

Simulating a client instance. $\mathcal{S}$ simulates an uncorrupted client instance as discussed above until one of the following events occurs:

- If the client instance Π_j^{sid} receives a message C' in the second round of the protocol that was not output by $\mathcal{S}$'s simulation of the matching server instance, then $\mathcal{S}$ decrypts C' to obtain the underlying password π and sends $(\mathsf{TestPwd}, \mathsf{sid}, P_j, \pi)$ to $\mathcal{F}$. (If C' is not a valid ciphertext, $\mathcal{S}$ uses $\pi = \perp$ which we assume is treated as an incorrect password guess by $\mathcal{F}$.) There are then two sub-cases:
  - If the password guess is correct, $\mathcal{S}$ continues to simulate Π_j^{sid} but now uses the true password π. If the client instance later concludes by computing a session key sk_j (possibly $\mathsf{sk}_j = \perp$), then $\mathcal{S}$ sends $(\mathsf{NewKey}, \mathsf{sid}, P_j, \mathsf{sk}_j)$ to $\mathcal{F}$.
  - If the password guess is incorrect, $\mathcal{S}$ chooses $r_j \| \tau_j \| sk_j$ at random and continues to use π_0 as before. If the client instance later concludes by computing a session key sk_j (possibly[3] $\mathsf{sk}_j = \perp$), then $\mathcal{S}$ sends $(\mathsf{NewKey}, \mathsf{sid}, P_j, \perp)$ to $\mathcal{F}$. (Note that in this case the given instance is labeled $\mathsf{interrupted}$, anyway.)
- If the above did not occur, then after completing the (simulation of the) zero-knowledge proof on behalf of the client $\mathcal{S}$ sends $(\mathsf{ready}, \mathsf{sid}, P_j)$ to $\mathcal{F}$. If the client instance concludes by computing a session key sk_j (possibly $\mathsf{sk}_j = \perp$), then $\mathcal{S}$ sends $(\mathsf{NewKey}, \mathsf{sid}, P_j, \perp)$ to $\mathcal{F}$.

Simulating a server instance. $\mathcal{S}$ simulates an uncorrupted server instance as discussed above until one of the following events occurs:

- If the server instance Π_i^{sid} ever receives a message C^* that was not output by $\mathcal{S}$'s simulation of the matching client instance, then $\mathcal{S}$ decrypts C^* to obtain the underlying password π and sends $(\mathsf{TestPwd}, \mathsf{sid}, P_i, \pi)$ to $\mathcal{F}$. (If C^* is not a valid ciphertext, $\mathcal{S}$ uses $\pi = \perp$ which we assume is treated as an incorrect password guess by $\mathcal{F}$.) There are then two sub-cases:

[3] In fact, with overwhelming probability $\mathsf{sk}_j = \perp$.

– If the password guess is correct, $\mathcal{S}$ continues to simulate Π_i^{sid} as before, but now using the true password π. If the server instance concludes by computing a session key sk_i (possibly $\mathsf{sk}_i = \perp$), then $\mathcal{S}$ sends $(\mathsf{NewKey}, \mathsf{sid}, P_i, \mathsf{sk}_i)$ to $\mathcal{F}$.

– If the password guess is incorrect, $\mathcal{S}$ continues to use π_0 but aborts the simulation automatically after the zero-knowledge proof from the client. $\mathcal{S}$ then sends $(\mathsf{NewKey}, \mathsf{sid}, P_i, \perp)$ to $\mathcal{F}$.

- If the above did not occur, then if the server instance ever concludes by outputting a session key sk_i (possibly $\mathsf{sk}_i = \perp$), the simulator $\mathcal{S}$ sends $(\mathsf{NewKey}, \mathsf{sid}, P_i, \mathsf{sk}_i)$ to $\mathcal{F}$.

4.3.2 Proof of Indistinguishability

We now prove that the actions of $\mathcal{S}$ in the ideal world are indistinguishable from a real-world execution with adversary $\mathcal{A}$. To do so, we consider a sequence of experiments beginning with an experiment Γ_0 that corresponds to the real-world execution with $\mathcal{A}$ and concluding with an experiment that corresponds to the ideal-world execution with $\mathcal{S}$. Each pair of neighboring experiments is shown to be indistinguishable from the point of view of any polynomial-time environment $\mathcal{Z}$; by transitivity, this proves that the real and ideal worlds are indistinguishable.

We describe our intermediate experiments with reference to an entity $\mathcal{S}'$ that just provides us with a convenient way to encapsulate certain parts of the experiment. $\mathcal{S}'$ will also (internally) assign labels to various instances — always initially assigning instances a fresh label — for "book-keeping" purposes. In the final experiment (that, recall, is supposed to correspond to an execution in the ideal world), the role of $\mathcal{S}'$ will be taken on by the simulator $\mathcal{S}$ that we defined previously, and the internal book-keeping will be done by the ideal functionality itself.

In the following descriptions, we use the term "honestly forwarded" to refer to messages received by an uncorrupted instance that were output by an uncorrupted partnered instance (namely, the instance with matching sid and the opposite role) and then forwarded unchanged by the adversary.

Experiment Γ_0. This corresponds exactly to a real-world execution of the protocol in the presence of $\mathcal{A}$. In a bit more detail, $\mathcal{S}'$ generates the CRS honestly and then interacts with the environment $\mathcal{Z}$ while running $\mathcal{A}$ as a sub-routine. Messages from $\mathcal{Z}$ to $\mathcal{A}$ are forwarded, and vice versa. Furthermore, $\mathcal{S}'$ also receives inputs from $\mathcal{Z}$ on behalf of any uncorrupted parties, and runs the protocol honestly on their behalf. As mentioned above, $\mathcal{S}'$ (internally) assigns the label fresh to any uncorrupted instance when it is first initialized. It also labels a client instance ready after that instance's zero-knowledge proof is completed. $\mathcal{S}'$ labels an instance $\mathsf{completed}$ once it terminates the protocol (whether with output a legitimate session key or the failure symbol $\perp$).

Experiment Γ_1. In this experiment, $\mathcal{S}'$ generates the string γ in the CRS using the simulator algorithm for the SS-ZKP. Furthermore, when executing an honest client instance $\mathcal{S}'$ always simulates the zero-knowledge proof. The following is immediate.

Lemma 8 *Experiments Γ_0 and Γ_1 are computationally indistinguishable.*

Experiment Γ_2. The only change introduced here is that now, whenever $\mathcal{S}'$ executes an honest client instance, it computes the ciphertext C^* sent in the initial message as an encryption of π_0 (with respect to public key pk_1), rather than as an encryption of the password provided to this instance by $\mathcal{Z}$.

Lemma 9 *Experiments* Γ_1 *and* Γ_2 *are computationally indistinguishable.*

Proof The only part of the protocol that depends on C^* is the zero-knowledge proof, which is now simulated by $\mathcal{S}'$. We note also that $\mathcal{S}'$ never uses the secret key sk_1 associated with pk_1 throughout either experiment. The lemma thus follows readily from the CPA-security of Σ. ■

Experiment Γ_3**.** We modify the previous experiment as follows. If an uncorrupted server instance receives an initial message C^* that is not honestly forwarded, $\mathcal{S}'$ decrypts C^* (using the secret key sk_1) to obtain a password π^*. It then tests this password against the password π being used by this server instance (i.e., as given to $\mathcal{S}'$ by $\mathcal{Z}$). If $\pi^* = \pi$ then $\mathcal{S}'$ labels the server instance compromised, while if $\pi^* \neq \pi$ then $\mathcal{S}'$ labels the server instance interrupted. For any server instance labeled compromised, $\mathcal{S}'$ aborts execution of this instance at the conclusion of the zero-knowledge proof given to it. (I.e., $\mathcal{S}'$ aborts automatically even if the zero-knowledge proof succeeds.)

Lemma 10 *Experiments* Γ_2 *and* Γ_3 *are computationally indistinguishable.*

Proof The changes in the internal labeling are not observable by $\mathcal{Z}$. The only observable change occurs in case $\mathcal{S}'$ aborts a server instance labeled compromised in Γ_3 when this instance would not have aborted in Γ_2. But this occurs with only negligible probability. To see this, let $s\|C$ be the value sent to some server instance in the third message and consider two sub-cases:

- Case 1: C is an encryption of π. Since C^* was *not* an encryption of π, and was not forwarded from the relevant partner instance, simulation soundness of the zero-knowledge proof system implies that (in either Γ_2 or Γ_3) the adversary gives a convincing proof that C^* and C encrypt the same value with only negligible probability. Thus, an abort would have occurred with overwhelming probability in Γ_2.
- Case 2: C is not an encryption of π. In this case an abort would always occur in Γ_2 (assuming Σ has perfect correctness).

This concludes the proof. ■

Experiment Γ_4**.** We modify the preceding experiment in the following way. If an uncorrupted server instance receives an initial message C^* that is not honestly forwarded, $\mathcal{S}'$ decrypts C^* (as in Γ_3) to obtain the underlying password π^*. If $\pi^* \neq \pi$ (where π is the password being used by the instance in question) then $\mathcal{S}'$ labels the server instance interrupted (as in Γ_3) and computes C' as an encryption of π_0 with respect to the public key pk'. (If $\pi^* = \pi$, then $\mathcal{S}'$ computes C' as an encryption of π exactly as in Γ_3.)

Note that, because any session marked interrupted aborts after completing verification of the zero-knowledge proof, the randomness used to generate the ciphertext C' in any such session is never subsequently used. Moreover, the secret key sk' corresponding to pk' is not used in either Γ_3 or Γ_4. As such, a proof of the following is immediate:

Lemma 11 *Experiments* Γ_3 *and* Γ_4 *are computationally indistinguishable.*

Experiment Γ_5**.** We introduce the following modifications to the previous experiment. Consider an uncorrupted server instance Π_S and the corresponding (uncorrupted) client instance Π_C. (If no corresponding uncorrupted client instance exists, the following discussion is moot.) If Π_S ever

receives an honestly forwarded C^* (i.e., C^* was sent by Π_C), but either the second message C' received by Π_C was not honestly forwarded, or the third message $s\|C$ received by Π_S was not honestly forwarded, then $\mathcal{S}'$ aborts Π_S following the zero-knowledge proof.

Lemma 12 *Experiments* Γ_4 *and* Γ_5 *are computationally indistinguishable.*

Proof We claim that in any case where an abort is introduced in Γ_5, an abort would have occurred in Γ_4 except with negligible probability. This follows from the observation that C^* in the case considered here is an encryption of π_0 (since it was output by an uncorrupted client instance), and then a similar argument as in the proof of Lemma 10. ■

Experiment Γ_6**.** We modify the previous experiment as follows. In an uncorrupted server instance where C^*, C', and $s\|C$ are all honestly forwarded, but the passwords being used by the server instance and the partnered client instance do not match, then $\mathcal{S}'$ aborts the server instance at the conclusion of the zero-knowledge proof. If the passwords *do* match, then $\mathcal{S}'$ sets the values of $r_i\|\tau_i\|\mathsf{sk}_i$ equal to the values already computed by the partnered client instance. This is a syntactic change only, and has no effect on the output of $\mathcal{Z}$.

Experiment Γ_7**.** We modify the experiment as follows. When an uncorrupted server instance receives an honestly forwarded initial message C^*, the second message C' is not computed as an encryption of π_0 (with respect to the public key pk'), rather than an encryption of π as before. Since the randomness used to compute C' is not used subsequently by $\mathcal{S}'$ (given the modifications made in Γ_5 and Γ_6), nor is the secret key sk', it follows from the CPA-security of Σ that:

Lemma 13 *Experiments* Γ_6 *and* Γ_7 *are computationally indistinguishable.*

Experiment Γ_8**.** We now begin modifying the treatment of client instances. Now, when executing an uncorrupted client instance where both C^* and C' are honestly forwarded, $\mathcal{S}'$ chooses $r_j\|\tau_j\|\mathsf{sk}_j$ uniformly at random (and then runs the rest of the protocol as before). The following is immediate from the properties of the smooth projective hash function and the fact that an honestly forwarded C' is an encryption of (an invalid password) π_0:

Lemma 14 *Experiments* Γ_7 *and* Γ_8 *are statistically indistinguishable.*

Experiment Γ_9**.** We introduce the following modifications. In an uncorrupted client instance where the second message C' is not honestly forwarded, decrypt C' to obtain a password π'. If π' is equal to the password π being used by the client instance in question, then $\mathcal{S}'$ labels this client instance compromised; if $\pi' \neq \pi$ then $\mathcal{S}'$ labels it interrupted. Moreover, in the latter case $\mathcal{S}'$ chooses $r_j\|\tau_j\|\mathsf{sk}_j$ uniformly at random (and then runs the rest of the protocol as before). Once again, the following lemma is immediate from the properties of the smooth projective hash function:

Lemma 15 *Experiments* Γ_8 *and* Γ_9 *are statistically indistinguishable.*

Experiment Γ_{10}**.** Now, in an uncorrupted client instance where C^* is honestly forwarded (to the partnered server instance) but the received message C' is not honestly forwarded (from the partnered server instance), $\mathcal{S}'$ decrypts C' to obtain a password π' and labels the client instance as in Γ_9. As in the previous experiment, if $\pi' \neq \pi$ then $\mathcal{S}'$ chooses $r_j\|\tau_j\|\mathsf{sk}_j$ uniformly at random

(and then runs the rest of the protocol as before). Exactly as previously, we have that Γ_9 and Γ_{10} are statistically close.

Experiment Γ_{11}. We modify the preceding experiment as follows. In an uncorrupted client instance that is not labeled compromised when sending the third message of the protocol, the ciphertext C is computed as an encryption of π_0 (with respect to the public key pk_2) rather than an encryption of the given password. Note that, for any such instance in Γ_{10}, it is the case that the randomness used to generate C is chosen uniformly at random, and is not used subsequently. Moreover, $\mathcal{S}'$ never uses the secret key sk_2 in either experiment. These observations, along with the CPA-security of Σ, immediately imply the following lemma.

Lemma 16 *Experiments Γ_{10} and Γ_{11} are computationally indistinguishable.*

Experiment Γ_{12}. In this experiment, we modify the execution of a client instance as follows. If a client instance is not labeled compromised, and the final message τ_i is not honestly forwarded, then the client instance aborts (with output $\perp$).

Lemma 17 *Experiments Γ_{11} and Γ_{12} are statistically indistinguishable.*

Proof If the client instance in question is interrupted, or is fresh but the partnered server instance has not yet output the final message τ_i, then the value τ_j local to the client instance is uniformly distributed from the point of view of $\mathcal{A}$ and so the client instance would accept only with negligible probability. On the other hand, if the client instance is fresh and the partnered server instance has already output $\tau_i = \tau_j$, then adversarial modification of this value will certainly lead to abort. ■

One can now (tediously) verify that the actions of $\mathcal{S}'$ in Γ_{12} are simulated perfectly by our simulator $\mathcal{S}$ (described in the previous section) in its interaction with the ideal functionality $\mathcal{F}$. Specifically, although $\mathcal{S}$ will not be given the passwords used by the various instances, $\mathcal{S}$ only ever needs to perform tests of equality on these passwords which it can do using TestPwd queries to $\mathcal{F}$. The labels maintained in $\mathcal{S}'$ will correspond correctly with the labels maintained by the ideal functionality, and all tests of equality by $\mathcal{S}'$ are done when the label of an instance is one for which a TestPwd query is allowed by $\mathcal{F}$. In any situation where one of the parties is compromised or its partner is corrupted, $\mathcal{S}$ will simply use the NewKey command to ensure that the relevant party will output the key that was computed in that session. If a session is interrupted then $\mathcal{S}$ uses the NewKey command to force output of $\perp$ just as would be the case in Γ_{12}. In the case of successful completion of the protocol, $\mathcal{S}$ will also use the NewKey command; this will give a different output than that computed by $\mathcal{S}'$, but since both are uniform random values the difference is purely syntactic. This concludes the proof of security for our protocol.

References

[1] M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In T. Malkin, editor, *Cryptographers' Track — RSA 2008*, LNCS, pages 335–351. Springer, Apr. 2008.

[2] M. Abdalla, C. Chevalier, and D. Pointcheval. Smooth projective hashing for conditionally extractable commitments. In *Advances in Cryptology — Crypto 2009*, volume 5677 of *LNCS*, pages 671–689. Springer, 2009.

[3] B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure computation without authentication. In *Advances in Cryptology — Crypto 2005*, volume 3621 of *LNCS*, pages 361–377. Springer, 2005.

[4] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Advances in Cryptology — Eurocrypt 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, 2000.

[5] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology — Crypto '93*, volume 773 of *LNCS*, pages 232–249. Springer, 1994.

[6] M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In *27th Annual ACM Symposium on Theory of Computing (STOC)*, pages 57–66. ACM Press, 1995.

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

[8] V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Advances in Cryptology — Eurocrypt 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, 2000.

[9] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145. IEEE, 2001.

[10] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In *Advances in Cryptology — Eurocrypt 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, 2005.

[11] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *Advances in Cryptology — Eurocrypt 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, 2002.

[12] R. Cramer and V. Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[13] R. Gennaro. Faster and shorter password-authenticated key exchange. In *5th Theory of Cryptography Conference — TCC 2008*, volume 4948 of *LNCS*, pages 589–606. Springer, 2008.

[14] R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. *ACM Trans. Information and System Security*, 9(2):181–234, 2006.

[15] O. Goldreich and Y. Lindell. Session-key generation using human passwords only. *Journal of Cryptology*, 19(3):241–340, 2006.

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

[17] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. *ACM Trans. Information and System Security*, 2(3):230–268, 1999.

[18] S. Jiang and G. Gong. Password based key exchange with mutual authentication. In *11th Annual International Workshop on Selected Areas in Cryptography (SAC)*, volume 3357 of *LNCS*, pages 267–279. Springer, 2004.

[19] J. Katz, P. D. MacKenzie, G. Taban, and V. D. Gligor. Two-server password-only authenticated key exchange. In *3rd International Conference on Applied Cryptography and Network Security (ACNS)*, volume 3531 of *LNCS*, pages 1–16. Springer, 2005.

[20] J. Katz, R. Ostrovsky, and M. Yung. Efficient and secure authenticated key exchange using weak passwords. *Journal of the ACM*, 57(1):78–116, 2009.

[21] J. Katz and V. Vaikuntanathan. Password-based authenticated key exchange based on lattices. In *Advances in Cryptology — Asiacrypt 2009*, volume 5912 of *LNCS*, pages 636–652. Springer, 2009.

[22] K. Kurosawa and Y. Desmedt. A new paradigm of hybrid encryption scheme. In *Advances in Cryptology — Crypto 2004*, volume 3152 of *LNCS*, pages 426–442. Springer, 2004.

[23] P. D. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. In *Advances in Cryptology — Asiacrypt 2000*, volume 1976 of *LNCS*, pages 599–613. Springer, 2000.

[24] M.-H. Nguyen and S. Vadhan. Simpler session-key generation from short random passwords. *Journal of Cryptology*, 21(1):52–96, 2008.

[25] V. Shoup. A proposal for an ISO standard for public key encryption. Cryptology ePrint Archive, Report 2001/112, 2001. `http://eprint.iacr.org/`.