# Universally Composable Password-Based Key Exchange

Ran Canetti* Shai Halevi* Jonathan Katz† Yehuda Lindell‡
Philip MacKenzie§

June 17, 2005

**Abstract**

We propose and realize a definition of security for password-based key exchange within the framework of universal composability (UC), thus providing security guarantees under arbitrary composition with other protocols. In addition, our definition captures some aspects of the problem that were not adequately addressed by most prior notions. For instance, our definition does not assume any underlying probability distribution on passwords, nor does it assume independence between passwords chosen by different parties. We also formulate a definition of password-based secure channels, and show how to realize such channels given any password-based key exchange protocol.

The password-based key exchange protocol shown here is in the common reference string model and relies on standard number-theoretic assumptions. The components of our protocol can be instantiated to give a relatively efficient solution which is conceivably usable in practice. We also show that it is impossible to satisfy our definition in the "plain" model (e.g., without a common reference string).

**Keywords:** key exchange, password-based protocols, universal composability.

*IBM T.J. Watson Research Center, Hawthorne, NY, USA. canetti@watson.ibm.com, shaih@alum.mit.edu.

†Dept. of Computer Science, University of Maryland, College Park, MD, USA. jkatz@cs.umd.edu. Supported by NSF CAREER award #0447075 and Trusted Computing grant #0310751.

‡Dept. of Computer Science, Bar-Ilan University, Israel. lindell@cs.biu.ac.il. Some of this work was carried out while the author was at IBM T.J. Watson.

§DoCoMo USA Labs, USA. philmac@docomolabs-usa.com. This work was carried out while the author was at Bell Labs, Lucent Technologies

# Contents

## 1 Introduction

Protocols for password-based key exchange have received much attention in recent years. In short, the problem is how to enable authenticated generation of a "high-quality" secret key between two parties whose only *a priori* shared information consists of a low-entropy password. In this setting, an attacker has a noticeable chance of impersonating one of the parties simply by guessing the correct password and running the prescribed authentication protocol. Such an attack is called an *on-line dictionary attack*, since one of the parties must be "on-line" and ready to participate in the protocol as the attacker exhaustively enumerates the dictionary in this way. Since this attack is unavoidable when low-entropy passwords are used, work in this area focuses on preventing *off-line dictionary attacks* in which an adversary tries to determine the correct password without the involvement of the honest parties based on information obtained during previous executions of the protocol. Roughly, a secure password-based protocol guarantees that an exhaustive on-line attack is the "best" possible strategy for an attacker. That is, the attacker must interact with a legitimate player in order to verify each password guess, and the interaction leaks no information other than whether or not the attacker's guess is correct. Besides their practical importance, password-based protocols are also interesting from a purely theoretical point of view: they provide a rare case where bootstrapping "strong security" from "weak security" has to be modeled, obtained, and argued.

The problem of resistance to off-line password-guessing attacks was first raised by Gong, et al. [22] in the asymmetric "PKI model" (where, in addition to a password, the user has the public key of the server). Formal definitions and proofs of security in this setting were later given by Halevi and Krawczyk [23] (see also [6]). A more difficult setting for this problem is one where the parties share *only* a password (and in particular, neither party knows the other's public-key). This setting was first considered by Bellovin and Merritt [5], and their work was followed by much additional research developing protocols with heuristic justifications for their security (see [7] for a survey). Formal definitions for this setting, together with protocols analyzed in the random-oracle/ideal-cipher models, were given by Bellare, Pointcheval, and Rogaway [3] (who proposed an indistinguishability-based definition) and Boyko, MacKenzie, Patel, and Swaminathan [8, 26] (who proposed a simulation-based definition). Goldreich and Lindell [20] introduced a third security definition and also gave the first provably-secure solution to this problem in the standard model, based on general assumptions; their protocol was recently simplified (at the expense of achieving a weaker security guarantee) by Nguyen and Vadhan [28]. Another setting that has been considered for this problem is one where, in addition to shared low-entropy passwords, all parties share a common reference string. In this setting, a practical and provably-secure protocol was first developed by Katz, et al. [25] based on the decisional Diffie-Hellman assumption (in the standard model). This protocol was subsequently generalized and abstracted by Gennaro and Lindell [19] who, among other things, obtain protocols that rely on the quadratic residuosity and $N^{\text{th}}$-residuosity assumptions.

The many definitions that have already been introduced [3, 8, 20] indicate that finding a "good" definition of security for password-based authentication is difficult and remains a challenging problem. In fact, it is not clear that any of the above definitions adequately address all aspects of the problem. For example, none of the above definitions relate to the (realistic) setting where the password-based protocol is used as a component within a larger protocol. Rather, it is assumed that the entire network activity consists only of many executions of the password-based protocol itself. Since the problem at hand involves *non-negligible* probabilities of "success" by the adversary, providing security-preserving composition is even more delicate than usual. Some of the above definitions have not been proven sufficient for implementing any form of secure channels — a natural goal of key-exchange protocols (see [10] for motivation). Finally, existing (explicit) definitions assume that passwords are chosen from some *pre-determined, known distribution*, and (with the exception of [8]) assume also that passwords shared between different parties are *independent*. (We note, however, that it is claimed in [25] that their proof extends to the case of dependent passwords.) These assumptions about password selection rarely hold in practice.

### 1.1 A New Definition

In this work, we propose and realize a new definition of security for password-based key-exchange protocols within the universally composable (UC) security framework [9]. That is, we propose an ideal functionality for "password-based key exchange" that captures the security requirements of the problem. (Such an ideal functionality can be thought of as the code for a "centralized trusted service", were one actually available to the parties.) Working in the UC framework allows us to benefit from the universal composition theorem. Loosely speaking, the theorem states that a protocol secure in this framework remains secure even when run in an arbitrary network, where many different protocols (secure or not) may run concurrently. In addition to addressing composability, the definition in this work also addresses the other concerns mentioned above: in particular, security is preserved even in the case of arbitrary and unknown password distributions, and even if related passwords are used. The important feature here is that the probability of the adversary succeeding in its attack is negligibly close to the probability of its guessing the password outright, even when this guess is based on information about the password that the adversary obtains from the arbitrary network or from related passwords that it learned. Finally, we show how protocols satisfying our definition may be used to construct password-based secure channels. Such channels enable private and authenticated communication, which in most cases is the goal of running the protocol in the first place.

As one might expect, formulating an ideal functionality that captures all the requirements of password-based key exchange involves a number of non-trivial definitional choices. Our formulation builds on the known UC formulation of (standard) key-exchange [9, 10], where security is guaranteed except with negligible probability. Unlike standard key-exchange, however, some mechanism must be introduced that allows the adversary to "break" the protocol with noticeable probability by guessing the correct password. A natural way of doing this is to have the functionality choose the passwords for the parties; then, if the adversary correctly guesses the password (where this guess is made explicitly by the adversary to the functionality), the adversary is allowed to choose the session-key that the parties obtain. Although this formulation is quite intuitive, it is somewhat limited in that it assumes a pre-determined dictionary or distribution on passwords and that passwords are chosen independently from each other. This formulation also fails to model possible leakage of partial information about the password to the adversary (since only the functionality knows the password).

We therefore take a different approach and allow the calling protocol (or the environment) to provide the password to the parties as part of their input. While this formulation may seem somewhat counter-intuitive at first, we show that it results in a definition of security that is at least as strong as that given by the first formulation.[1] Furthermore, it does not make any assumptions as to how the password is chosen and it imposes no pre-determined probabilities of failure.

**Realizing the definition.** We construct a protocol that realizes our definition. The protocol is an extension of the protocols of KOY/GL [25, 19], and as such is in the common reference string

[1]The alternative formulation in which the functionality chooses the passwords may be obtained from the authors, along with a proof that it is no stronger than the definition presented here.

model and may be based on some standard number-theoretic assumptions. Our protocol adds to the protocols from [25, 19] a pre-flow and a simulation-sound zero-knowledge proof of consistency. All the building blocks used have efficient instantiations under the decisional Diffie-Hellman, quadratic residuosity, or $N^{\text{th}}$-residuosity assumptions. As a result, our protocol is reasonably efficient (it has 6 rounds and requires at most 30 modular exponentiations per party). Some of the efficiency improvements we use in our protocol are applicable also to the protocol of [25] (and seemingly [19]); see [24]. Applied there, these improvements yield the most efficient known password-based protocols meeting the definition of [3] without random oracles.

**On the necessity of set-up assumptions.** Our protocol is constructed in the common reference string model, and so requires a trusted setup phase. In fact, we show in Section 7 that our UC-based definition of password-based key-exchange cannot be securely realized by any protocol in the *plain model* (i.e., in a model with no trusted setup whatsoever). Beyond providing some justification for our use of a common reference string, this result stands in sharp contrast with the fact that standard UC-secure key exchange *can* be realized in the plain model [10]. It also shows that our definition is strictly stronger than the definitions used by [20, 28], which *can* be realized in the plain model. (We stress that in contrast to our definition, the definitions of [20, 28] do not guarantee security even under concurrent composition of the same protocol with the same password.)

**Password-based secure channels.** Perhaps the most important application of key-exchange protocols is for establishing secure communication sessions between pairs of parties. To advocate the adequacy of our proposed definition we formulate a UC notion of password-based secure channels, and show how to realize it given our notion of password-based key exchange. It is of course impossible to obtain standard secure channels using short passwords, since the adversary may guess the password with non-negligible probability. Consequently, our notion of password-based secure channels relaxes the standard notion in a way similar to which our notion of password-based key exchange relaxes the standard notion of key exchange. In Section 6 we show that the standard protocols for realizing secure channels based on standard key exchange (see, e.g., [10]), suffice also for realizing password-based secure channels from password-based key exchange.

# 2 Definition of Security

In this section we motivate and present our formulation of an ideal functionality for password-based key exchange in the UC framework. The starting point for our approach is the definition for universally composable "standard" key-exchange [10] (cf. Figure 1). Our aim is to define a functionality that achieves the same effect as standard key-exchange (where the parties have high-entropy keys), except that we also incorporate the inherent "security defect" due to the use of low-entropy passwords. Two ways of introducing this "security defect" come to mind:

1. One option is to consider the same functionality $\mathcal{F}_{\mathsf{KE}}$ as in Figure 1, but to relax the requirement of indistinguishability between the real and ideal worlds. For example, when passwords are assumed to be chosen uniformly from a dictionary $\mathcal{D}$, one would define a secure protocol as one whose real-world execution is distinguishable from an interaction with the ideal functionality with probability at most, say, $1/|\mathcal{D}|$ plus a negligible amount.

2. A second possibility is to incorporate the "defect" directly into the functionality by, e.g., allowing the adversary to make explicit password guesses and to "break" the protocol following

a successful guess. Here, the adversary "breaks" the protocol with noticeable probability even in the ideal world, and thus the standard notion of realizing an ideal functionality can be used.

Among previous works that used simulation-based definitions of security for password protocols, the first approach was taken by [20, 28], while the second was taken by [8]. (Other definitions are not simulation-based and so do not fit into either approach.) In this work we adopt the second option for two reasons. First, the second option allows us to use the UC composition theorem directly and thus guarantee security of password-based key-exchange protocols even when run in arbitrary protocol environments. Second, this approach easily extends to handle additional complexities such as multiple users with different distributions on their passwords, or dependencies among various passwords. These aspects seem hard (if not impossible) to handle using the first approach.

Before proceeding to our definition, we describe the standard key-exchange functionality of [10]. (We note that the formulation of $\mathcal{F}_{\mathsf{KE}}$ in Figure 1 is somewhat different from the one in [10]; however, the differences are inconsequential for the purpose of this work.) The main idea behind the $\mathcal{F}_{\mathsf{KE}}$ functionality is as follows: If both participating parties are not corrupted, then they receive the same uniformly-distributed session key, and the adversary learns nothing about the key except that it was generated. However, if one of the parties *is* corrupted, then the adversary is given the power to fully determine the session key. The rationale for this is that the aim of key exchange is to enable honest parties to generate a key that is *unknown* to an external adversary. If one of the participating parties is corrupted, then the adversary will learn the generated key (because it is one of the participants), and so the security requirement is meaningless. In such a case, there is nothing lost by allowing the adversary to determine the key. (We remark that the "role" variable in the NewSession message is included in order to let a party know if it is playing the role of the initiator or the responder in the protocol. This has no effect on the security, but is needed for correct executions.)

**Functionality $\mathcal{F}_{\mathsf{KE}}$**

$\mathcal{F}_{\mathsf{KE}}$ is parameterized by a security parameter $k$. It interacts with an adversary $S$ and a set of (dummy) parties via the following queries:

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

Send $(\mathsf{NewSession}, sid, P_i, P_j, \text{role})$ to $S$. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record $(P_j, P_i)$, then record $(P_i, P_j)$.

**Upon receiving a query** $(\mathsf{NewKey}, sid, P_i, sk)$ **from** $S$**, where** $|sk| = k$**:**

If there is a record $(P_i, P_j)$, and this is the first NewKey query for $P_i$, then:

- If either $P_i$ or $P_j$ is corrupted, then output $(sid, sk)$ to player $P_i$.
- If there is also a record $(P_j, P_i)$, and a key $sk'$ was sent to $P_j$, output $(sid, sk')$ to $P_i$.
- In any other case, pick a new random key $sk'$ of length $k$ and send $(sid, sk')$ to $P_i$.

Figure 1: The authenticated key-exchange functionality $\mathcal{F}_{\mathsf{KE}}$

Our definition of the password-based key-exchange functionality $\mathcal{F}_{\mathsf{pwKE}}$ is given in Figure 2. The definition is similar to that of $\mathcal{F}_{\mathsf{KE}}$, in that if one of the participating parties is corrupted the adversary is given the power to fully determine the resulting session key. However, this power is also given to the adversary in case it succeeds in guessing the parties' shared password. In the functionality $\mathcal{F}_{\mathsf{pwKE}}$, a session is marked compromised if the adversary makes a successful password guess, and the adversary can determine the session key for compromised sessions.

An additional property of our definition is that *failed* adversarial attempts at guessing a key are detected by the participating parties. Specifically, if the adversary makes an incorrect password guess in a given session, then the session is marked interrupted and the parties are provided independently-chosen session keys. (Giving the parties error messages in this case would correspond to requiring explicit mutual authentication; see additional discussion below.)

A session that is not compromised or interrupted (and is still in progress) is considered fresh. Such sessions (between honest parties) conclude with both parties receiving the same, uniformly distributed session-key. Once a party receives a key in a fresh session, that session is marked completed.

In the definition of $\mathcal{F}_{\mathsf{pwKE}}$, the password is chosen by the environment who then hands it to the parties as input.[2] Since we quantify over all (polynomial-time) environments, this implies that security is preserved for all efficient password distributions, as well as when arbitrarily-related passwords are used in different sessions. Furthermore, since passwords are provided by the "environment in which the protocol is run", security is preserved even when passwords are used for other, unintended purposes by that same environment, thereby possibly leaking information about the passwords. (When we say that security is preserved here, we mean that the probability that an adversary can break the password-based key-exchange protocol is the same as its probability of guessing a password outright, given the potential misuse mentioned above.) We also remark that our definition guarantees security even in the case where two honest players execute the protocol with different passwords. (In fact, this is quite a realistic scenario which occurs every time a user mistypes a password; previous definitions did not guarantee anything in such a case.)

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

The functionality $\mathcal{F}_{\mathsf{pwKE}}$ is parameterized by a security parameter $k$. It interacts with an adversary $S$ and a set of parties via the following queries:

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

Send $(\mathsf{NewSession}, sid, P_i, P_j, \mathrm{role})$ to $S$. In addition, if this is the first NewSession query, or if this is the second NewSession query and there is a record $(P_j, P_i, pw')$, then record $(P_i, P_j, pw)$ and mark this record fresh.

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

If there is a record of the form $(P_i, P_j, pw)$ which is fresh, then do: If $pw = pw'$, mark the record compromised and reply to $S$ with "correct guess". If $pw \neq pw'$, mark the record interrupted and reply with "wrong guess".

**Upon receiving a query** $(\mathsf{NewKey}, sid, P_i, sk)$ **from** $S$**, where** $|sk| = k$**:**

If there is a record of the form $(P_i, P_j, pw)$, and this is the first NewKey query for $P_i$, then:

- If this record is compromised, or either $P_i$ or $P_j$ is corrupted, then output $(sid, sk)$ to player $P_i$.
- If this record is fresh, and there is a record $(P_j, P_i, pw')$ with $pw' = pw$, and a key $sk'$ was sent to $P_j$, and $(P_j, P_i, pw)$ was fresh at the time, then output $(sid, sk')$ to $P_i$.
- In any other case, pick a new random key $sk'$ of length $k$ and send $(sid, sk')$ to $P_i$.

Either way, mark the record $(P_i, P_j, pw)$ as completed.

Figure 2: The password-based key-exchange functionality $\mathcal{F}_{\mathsf{pwKE}}$

As additional justification for our definition, we show in Section 6 that it suffices to construct password-based secure channels (arguably the most common application of such protocols). In addition, we show in Appendix A that a protocol securely realizing our functionality is secure also with respect to the definition of Bellare, et al. [3] (modulo unimportant differences regarding the

[2]This is in contrast to an alternative approach described in the Introduction where the functionality chooses the password according to some predetermined distribution, and this password is hidden even from the environment. As we have mentioned, security under our definition implies security under that alternative approach.

formalization of session identifiers[3]). We also show there that our definition implies the "expected" notion of security against a passive eavesdropper (even one who happens to know the password being used). These last two results can be viewed as "sanity checks" of our definition.

**Additional discussion.** The definition of $\mathcal{F}_{\mathsf{pwKE}}$ could be *strengthened* to require explicit mutual authentication by insisting that after a "wrong guess" of the password, the session would fail (instead of producing a random and independent key). Similarly, a session with mismatching passwords would also fail. We chose not to include these requirements because **(a)** we want to keep the exposition simple; **(b)** mutual authentication is not needed for secure channels; and **(c)** it is well known that any secure key-exchange protocol (including ours) can be augmented to provide mutual authentication by adding two "key confirmation" flows at the end (and refreshing the session key).

The definition could also be *weakened* by notifying the simulator whether or not the passwords match in the two NewSession queries. Roughly, the difference is that the current formulation requires that an eavesdropper be unable to detect whether the session succeeded (i.e., both parties got the same key) or failed (i.e., they got different keys). Although we are not aware of any application where this is needed, it makes the definition simpler to describe and our protocol anyway satisfies this requirement.

Finally we comment that the reason for marking a record as completed after delivering the key is to preclude protocols where the adversary has an opportunity for an on-line guessing attack even after the key was established. We thank Vladimir Kolesnikov for pointing this out to us.

# 3 Securely Realizing $\mathcal{F}_{\mathsf{pwKE}}$: a High Level Description

We now sketch our protocol for securely realizing the functionality $\mathcal{F}_{\mathsf{pwKE}}$ in the common reference string model. We remark that our protocol is proven secure in the model of *static corruptions* (where the adversary may corrupt some of the participants, but only prior to the beginning of a protocol execution) and *unauthenticated channels* (where the adversary has full control over the communication channels and can insert, modify and delete all messages sent by the honest parties). Although we consider only static corruptions, the "weak-corruption model" of [3] is implied by our definition and achieved by our protocol; see Appendix A. (In the weak-corruption model, the adversary may obtain passwords adaptively throughout the execution. This essentially models leakage of passwords, rather than adaptive corruption of parties.)

## 3.1 Preliminaries

The protocol uses a number of primitives: one-time signatures, CPA-secure and CCA-secure public-key encryption, simulation-sound zero-knowledge proofs, and smooth projective hashing. We provide only a brief description of the latter two primitives here. More details are provided later in Section 4.

**Simulation-sound zero-knowledge (SSZK) proofs.** Informally speaking, a zero-knowledge proof system is said to be (unbounded) *simulation-sound* if it has the property that an adversary cannot provide a convincing proof for a false statement, even if it has seen *simulated proofs*. Such

[3]In our formalization, a unique session identifier *sid* is assumed to be part of the input to the functionality while in [3] the *sid* is a function of the eventual transcript of a protocol execution. In the two-party setting, a session identifier as required by our formulation can be obtained by having the parties exchange random strings and then setting *sid* to be their concatenation.

simulated proofs may actually prove false statements, and so we require that the adversary can copy these proofs but do nothing more. More formally, the adversary is given oracle access to the zero-knowledge simulator and can request simulated proofs of any statement that it wishes (true or false). The adversary is then said to *succeed* if it generates a convincing proof of a false statement, and this proof was not received from the oracle. This concept was first introduced by Sahai [30] and De Santis, et al. [15] in the context of non-interactive zero-knowledge. For the case of interactive protocols, the notion was formally defined by Garay, et al. [18].[4] Efficient methods for transforming three-round honest-verifier zero-knowledge protocols (also called $\Sigma$-protocols [13]) into simulation-sound zero-knowledge protocols in the common reference string model have been shown in [18] and [27]. We note that, according to the definition of [18], simulation-sound zero knowledge protocols also achieve *concurrent* zero knowledge; i.e., the zero knowledge property holds for an unbounded number of asynchronous executions of an honest prover. Finally, we note that simulation-sound zero knowledge is a weaker requirement than universally-composable zero-knowledge, and more efficient constructions for it are known.

**Smooth projective hashing [14].** On a very high level, a projective hash family is a family of hash functions that can be computed using one of two keys: the (secret) hashing key can be used to compute the function on every point in its domain, whereas the (public) projected key can only be used to compute the function on a specified subset of the domain. Such a family is called "smooth" if the value of the function on a point outside the specified subset is uniformly distributed, even given the projected key. More formally (but still far from being exact), let $X$ be a set and let $L \subset X$. We say a hash function $H_{hk}$ mapping $X$ to some set is projective if there exists a projection function $\alpha(\cdot)$ that maps hash keys $hk$ into their projections $hp = \alpha(hk)$, such that for every $x \in L$ the value of $H_{hk}(x)$ is uniquely determined by $hp$ and $x$. (In contrast, for $x \notin L$ the value of $H_{hk}(x)$ need not be determined by $hp$ and $x$.) A smooth projective hash function has the additional property that for $x \notin L$, the projection $hp$ actually says *nothing* about the value of $H_{hk}(x)$. More specifically, given $x$ and $hp = \alpha(hk)$, the value $H_{hk}(x)$ is (statistically close to) a uniformly distributed element in the range of $H_{hk}$.

We have already mentioned that for $x \in L$ the projected key $hp$ fully determines the value $H_{hk}(x)$, but so far we have said nothing about whether or not this value can be efficiently computed. An important property of smooth projective hash functions is that if the subset $L$ is an $NP$-language, then for $x \in L$ it is possible to compute $H_{hk}(x)$ using the projected key $hp = \alpha(hk)$ and a witness of the fact that $x \in L$. Thus, for $x \in L$ there are two alternative ways of computing $H_{hk}(x)$:

1. Given the hashing key $hk$, compute $H_{hk}(x)$ directly.
2. Given the projected key $hp = \alpha(hk)$ and a witness $w$ that $x \in L$, compute the hash value $h_{hp}(x; w) = H_{hk}(x)$.

Following [19], the set $X$ that we consider in this work is the set $\{(c, m)\}$ of all ciphertext/plaintext pairs under a given public-key $pke$. Furthermore, the language $L$ is taken to be $\{(c, m) \mid c = E_{pke}(m)\}$; that is, $L$ is the set of all ciphertext/plaintext pairs $(c, m)$ where $c$ is an encryption of $m$ under the public-key $pke$. This language is indeed an $NP$ language, with the witness being the randomness that was used in the encryption of $m$.

We also comment that the semantic security of the encryption implies that $L$ is hard on the average (i.e., it is hard to distinguish a random element in $X$ from a random element in $L$). For

[4]When we say a proof is simulation sound, we will also mean that it is *uniquely applicable* [30]; that is, a proof is valid for at most one statement.

| $\underline{P_i \text{ (server)}}$ | | $\underline{P_j \text{ (client)}}$ |
|---|---|---|
| | **CRS:** public key $pke$ | |
| | | $(sk, vk) \leftarrow \text{sigKey}(\$)$ |
| | $\xleftarrow{\quad c_1,\ vk \quad}$ | $c_1 \leftarrow E_{pke}(pw; r_1)$ |
| $c_2 \leftarrow E_{pke}(pw; r_2)$ | | |
| $hk \leftarrow \mathcal{H}$ | | |
| $hp \leftarrow \alpha(hk)$ | $\xrightarrow{\quad c_2,\ hp \quad}$ | |
| | | $hk' \leftarrow \mathcal{H}$ |
| | | $hp' \leftarrow \alpha(hk')$ |
| | $\xleftarrow{\quad hp',\ \sigma \quad}$ | $\sigma \leftarrow \mathsf{Sign}_{sk}(c_2, hp, hp')$ |
| if $(\mathsf{Verify}_{vk}((c_2, hp, hp'), \sigma) = 1)$ | | |
| session-key $\leftarrow H_{hk}(c_1, pw)$ $+\, h_{hp'}(c_2, pw; r_2)$ | | session-key $\leftarrow h_{hp}(c_1, pw; r_1)$ $+H_{hk'}(c_2, pw)$ |

Figure 3: The core of the KOY/GL protocols.

such languages, it was proven in [19] that given a random $x \in L$ and $hp = \alpha(hk)$, the value $H_{hk}(x)$ is *computationally indistinguishable* from a random value in the range of $H_{hk}$. (This holds even though for any $x \in L$, the value $H_{hk}(x)$ is uniquely determined by $x$ and $hp$.)

In the description below we denote by $hk \leftarrow \mathcal{H}$ the selection of a random hashing key, and denote the projection of this key by $hp = \alpha(hk)$. We also denote the computation of the hash value using the hashing key $hk$ by $H_{hk}(x)$, and denote computation of the hash value using the projected key $hp$ and witness $w$ by $h_{hp}(x; w)$. (The statements $x$ below are actually pairs $(c, m)$ and the witness is the randomness $r$; thus, we write $H_{hk}(c, m)$ and $h_{hp}(c, m; r)$.)

### 3.2 The KOY/GL Protocol

The starting point of our protocol is the password-based key-exchange protocol of Katz, Ostrovsky, and Yung [25], as generalized and abstracted by Gennaro and Lindell [19]. The "core" of this protocol is sketched in Figure 3 (in this figure we suppress various details unimportant for the present high-level discussion). At a high level, the parties in the KOY/GL protocol exchange CCA-secure encryptions[5] of the password, encrypted with the public-key found in the common reference string, and then compute the session key by combining (smooth projective) hashes of the two ciphertext/password pairs. In order to do this, each party chooses a hashing key for a smooth projective hash function and sends the "projected key" to the other party.

Ignoring for the moment the signature keys from Figure 3, let $c_2, hk$ and $c_1, hk'$ be the ciphertexts and hashing keys generated by parties $P_i$ and $P_j$, respectively. Party $P_i$ can compute $H_{hk}(c_1, pw)$ since it knows the actual hashing key $hk$. Furthermore, since it generated the ciphertext $c_2$, it can compute $h_{hp'}(c_2, pw; r_2) = H_{hk'}(c_2, pw)$ using its knowledge of the randomness $r_2$ that was used to generate $c_2 = E_{pke}(pw; r_2)$. (Recall the two alternate ways of computing $H_{hk}(x)$ described above.) Symmetrically, $P_j$ computes the same session key using $hk'$, $hp$, and its knowledge of $r_1$.

[5]It is shown in [19] that non-malleable commitments can be used in place of CCA-secure encryption. However, for our extension of the protocol to the UC framework we will need to use encryption, so we describe it this way.

The basic idea behind the security of the protocol can be described as follows. Denote the shared password of a client and server by $pw$. If the client receives an encryption $c$ of the wrong password $pw'$, then (by the definition of smooth projective hashing) the hash she computes will be random and independent of all her communication. (This holds because the statement $(c, pw)$ is not in the language, so $H_{hk'}(c, pw)$ is close to uniform even given the projected key $hp'$.) A similar argument holds for the server. Thus, for an adversary to distinguish a session key from random, it must send one of the parties an encryption of the *correct* password $pw$.

The adversary can obtain an encryption of the correct password by copying a ciphertext from another execution of the protocol, but then it does not know the randomness that was used to generate this ciphertext. By the property discussed earlier (regarding hard-on-the-average languages), the value $H_{hk}(c, pw)$ is computationally indistinguishable from uniform, even given $hp$. Moreover, since the encryption scheme is CCA-secure, and thus non-malleable, the adversary cannot generate a new encryption of $pw$ with probability any better than it could achieve by simply guessing passwords from the dictionary and encrypting them.

Finally, the adversary may try to gain information by copying ciphertexts from a current session faithfully but not copying other values (such as the hash projected keys). This type of man-in-the-middle attack is prevented using the one-time signature. We conclude that the adversary succeeds in its attack if and only if it generates an encryption of the correct password. In other words, the adversary succeeds only with the same probability with which it can guess the secret password, as required.

### 3.3 Extending the Protocol to Realize $\mathcal{F}_{\mathsf{pwKE}}$

The protocol of Figure 3 serves as a good starting point, but it does not seem to achieve the security required by our definition. The main issue is that an ideal-model simulator must be able to extract the adversary's password guess.[6] At first glance, it may seem that this is not a problem because in the ideal model the simulator has control over the common reference string and so can include a public key $pke$ for which it knows the corresponding secret key $ske$. Then, when the adversary generates an encryption of the password $c = E_{pke}(pw)$, the simulator can decrypt using $ske$ and obtain the password guess $pw$. However, as we will now show, this seems not to suffice. In order to see where the difficulty arises, consider an ideal-model adversary/simulator $\mathcal{S}$ that has access to the functionality $\mathcal{F}_{\mathsf{pwKE}}$ and needs to simulate the KOY/GL protocol for a real-life adversary $\mathcal{A}$. Informally, simulating the server when the adversary impersonates a *client* can be carried out as follows: The simulator decrypts the ciphertext $c_1$ generated by $\mathcal{A}$ and recovers the adversary's "password guess" $pw$ (this decryption can be carried out because $\mathcal{S}$ chooses the common reference string so that it has the corresponding secret key). The simulator then sends $pw$ to $\mathcal{F}_{\mathsf{pwKE}}$ as its own guess. If the guess is incorrect then, as described above, the smoothness of the hash function causes the honest parties to output independent random keys (as occurs in the ideal model for the interrupted session). In contrast, if the guess is correct then the simulator has learned the correct password and can continue the remainder of the execution exactly as an honest party would when using that password.

However, consider what happens when the adversary impersonates a *server*. Here, the simulator must send some $c_1$ (presumably an encryption of some password $pw'$) *before* the adversary replies with $c_2$. As before, the simulator can decrypt $c_2$ to recover the adversary's password guess $pw$,

[6] This need to extract is not a mere technicality, but is rather quite central to our definition. In particular, this enables us to argue that the level of security achieved is equivalent to the probability of successfully guessing the password, even in the case that related and partially-leaked passwords are used.

| $P_i$ (client) | | $P_j$ (server) |
|---|---|---|
| | **CRS:** $pke$ | |
| $c_0 \leftarrow E_{pke}(pw; r_0)$ | $\xrightarrow{\quad c_0 \quad}$ | |
| | | $(sk, vk) \leftarrow \text{sigKey}(\$)$ |
| | $\xleftarrow{\quad c_1, vk \quad}$ | $c_1 \leftarrow E_{pke}(pw; r_1)$ |
| $c_2 \leftarrow E_{pke}(pw, r_2)$ | | |
| $hk \leftarrow \mathcal{H}$ | | |
| $hp \leftarrow \alpha(hk; c_1)$ | $\xrightarrow{\quad c_2, hp \quad}$ | |
| | $\xrightarrow{\text{ZKP}(c_0 \approx c_2)}$ | |
| | | $hk' \leftarrow \mathcal{H}$ |
| | | $hp' \leftarrow \alpha(hk'; c_2)$ |
| | $\xleftarrow{\quad hp', \sigma \quad}$ | $\sigma \leftarrow \mathsf{Sign}_{sk}(c_2, hp, hp')$ |
| if $(\mathsf{Verify}_{vk}((c_2, hp, hp'), \sigma) = 1)$ | | |
| session-key $\leftarrow H_{hk}(c_1, pw)$ | | session-key $\leftarrow h_{hp}(c_1, pw; r_1)$ |
| $+ h_{hp'}(c_2, pw; r_2)$ | | $+ H_{hk'}(c_2, pw)$ |

Figure 4: The core of the universally-composable protocol.

and submit this guess to $\mathcal{F}_{\mathsf{pwKE}}$. However, if it turns out that $pw$ is a *correct* guess, the simulator is stuck with a ciphertext $c_1$ that in all likelihood is an encryption of a different (i.e., incorrect) password. Without knowing the hashing key $hk$ that $\mathcal{A}$ holds, the simulator cannot predict the value $H_{hk}(c_1, pw)$ that the adversary will compute (since $(c_1, pw) \notin L$). Thus, the simulator seems to have no way of ensuring that the secret key computed by $\mathcal{A}$ is the same as the one the environment obtains from the functionality (via the client).[7]

To overcome this problem, we modify the protocol by having the server send a "pre-flow" $c_0$ which also contains an encryption of the password (i.e., $c_0 = E(pw; r_0)$). Then, when the server later sends $c_2$, it proves in zero knowledge that $c_0$ and $c_2$ are encryptions of the same value. We stress that the session key is still computed using only $c_1$ and $c_2$ and so the simulator need only ensure that consistency hold between these two ciphertexts (where by consistency, we mean that they are both an encryption of the same password). The modified protocol is sketched in Figure 4. (Note that we switch the "client" and "server" roles so that the client is still the one who sends the first message.)

We now describe the high-level simulation strategy for the modified protocol (and thus why this modification solves the above-described problem):

1. *Case 1 — the adversary $\mathcal{A}$ impersonates the client:* The simulator $\mathcal{S}$ obtains the ciphertext $c_0$, decrypts it to obtain $pw$, and sends $pw$ to $\mathcal{F}_{\mathsf{pwKE}}$ as the password guess. If this guess is correct, then $\mathcal{S}$ continues the simulation using the same $pw$ and consistency is achieved. Note that the zero-knowledge proof ensures that the ciphertext $c_2$ later sent by $\mathcal{A}$ is also an encryption of $pw$ (and thus, in this case, consistency between $c_0$ and $c_1$ implies consistency

[7] This problem does not arise in the proofs of KOY/GL, since the "simulator" there can just halt upon a correct password guess by the adversary. Our simulator, on the other hand, must continue to simulate both when the adversary fails *and* when it succeeds.

between $c_1$ and $c_2$, as desired).

2. *Case 2 — the adversary $\mathcal{A}$ impersonates the server:* In this case, the simulator $\mathcal{S}$ generates the pre-flow ciphertext $c_0$ as an encryption of some default value. Then, upon receiving $c_1$ from $\mathcal{A}$, the simulator $\mathcal{S}$ decrypts it to obtain $pw$ and sends $pw$ to $\mathcal{F}_{\mathsf{pwKE}}$ as the password guess. If the guess is correct, then $\mathcal{S}$ generates $c_2$ to also be an encryption of $pw$. Notice that $c_1$ and $c_2$ are now consistent in that they both encrypt the correct password $pw$. The only problem remaining in the simulation is that $\mathcal{S}$ is supposed to prove that $c_0$ and $c_2$ are consistent (which in this case they are not). It does this by using the *zero-knowledge simulator* for the proof of consistency. By the zero-knowledge property, this proof is indistinguishable from a real one (and this holds even though the statement in this case is false). We therefore conclude that in the case of a correct password guess consistency is achieved and, in addition, the adversary cannot distinguish its view in the simulation from its view in a real execution.

We stress that Figure 4 is only a sketch of the protocol and does not contain all the details. For example, the full protocol uses *labeled encryption* [31] in order to bind certain protocol information to the ciphertexts (such as the session-id and the verification key of the signature scheme) and in order to prevent other types of man-in-the-middle attacks. (Labels were used implicitly for the same purpose in [25, 19].) A detailed description of the protocol can be found in the next section, where we also provide a formal statement and proof of the following result:

**Theorem 1** (main theorem – informally stated): *Assume the existence of CCA-secure encryption schemes with smooth projective hash functions, and simulation-sound zero-knowledge proofs. Then there exists a protocol in the common reference string model that securely realizes the $\mathcal{F}_{\mathsf{pwKE}}$ functionality in the presence of static-corruption adversaries.*

We remark that all the building blocks of our protocol can be constructed under the DDH, quadratic residuosity, or $N^{\text{th}}$-residuosity assumptions, and so UC-secure password-based key exchange is possible under any of these assumptions. For the most efficient instantiation we would use encryption and smooth projective hash functions based on the DDH assumption and a simulation-sound zero-knowledge proof system [18, 27] based on the strong RSA assumption. Hence, the end result would rely on all of these assumptions for its security.

**Efficiency notes.** Considering the protocol as depicted in Figure 4, we emphasize that it suffices to use a (simulation sound) zero-knowledge proof of *membership*, rather than a proof of *knowledge*. This allows for improved efficiency; see [18]. Furthermore, it suffices to generate $c_0$ and $c_2$ using an encryption scheme that is only *CPA-secure*, rather than CCA-secure[8] (in this case, the public key used to encrypt $c_0$ and $c_2$ is generated independently of the public key used to encrypt $c_1$); this is how we present the scheme in the following section. Using a CPA-secure scheme provides efficiency improvements in the encryption itself, the projective hashing step, and the proof of consistency. (In particular, a highly efficient and simple construction of smooth projective hashing was demonstrated for the El Gamal encryption scheme — which is CPA-secure under the DDH assumption — in [19]. Furthermore, proving consistency of El Gamal encryptions is more efficient than proving consistency of, e.g., CCA-secure Cramer-Shoup encryptions.)

[8] In fact, the second encryption in the KOY/GL protocols can be generated using a CPA-secure scheme (e.g., El Gamal) as well; see [24]. This yields the most efficient known password-based key-exchange protocol in the standard model (i.e., without random oracles), albeit under a weaker definition than the one considered here.

# 4 Detailed Description of our Protocol

In this section we present the full description of our protocol. We begin by describing in some detail the primitives that we use in our protocol, followed by a detailed description of the protocol itself. Section 5 contains the proof of security.

## 4.1 Building Blocks

**CPA-secure and labeled CCA-secure public-key encryption.** We use both a semantically-secure encryption scheme [21] (which we call *CPA-secure*), and a *labeled* CCA-secure encryption scheme (see [31]). The latter is similar to the standard notion of CCA-secure encryption [29, 16, 2], with the added property that an arbitrary label can be bound to the ciphertext in a non-malleable way. In more detail, both the encryption and decryption procedures accept an additional input string called a *label*, and we only require correct decryption if the label provided to the decryption algorithm is the same as that used to encrypt. The standard CCA attack model is also modified by letting the adversary specify labels in its decryption queries, as well as a "target label" when it submits its two "target messages" to be encrypted. The crux of the definition is that the adversary is *allowed* to query the decryption oracle on any ciphertext/label pair that is not equal to the target ciphertext/label pair. A generic way of transforming any CCA-secure encryption scheme $E$ into a labeled CCA-secure encryption scheme $E'$ is to set $E'_{pke}(m; \mathit{label}) = E_{pke}(m \mid \mathit{label})$, and have the decryption routine verify that the labels match. However, more efficient solutions are possible in many cases.

We denote by $E_{pke}(m; \ell; r)$ the labeled encryption of message $m$ with respect to public key $pke$, using label $\ell$ and randomness $r$. Similarly we denote by $D_{ske}(c; \ell)$ the decryption of ciphertext $c$ with secret key $ske$ and label $\ell$. We denote by $\mathcal{E}_{pke}$ an efficiently recognizable superset of the set of all possible encryptions using key $pke$. In particular, in our definitions of smooth projective hashing below, a "ciphertext" with respect to a key $pke$ will refer to an element in $\mathcal{E}_{pke}$. The exact nature of this set depends on the specific encryption scheme being used (see, e.g., [19]). The notations for CPA-secure schemes are similar, except with the labels omitted.

**Smooth projective hashing.** Recall that a projective hashing family [14] is a family of hash functions that can be computed using one of two keys: the (secret) hashing key can be used to compute the function on every point in its domain, whereas the (public) projected key can only be used to compute the function on some specified subset of the domain. Such a family is called "smooth" if the value of the function on any single point outside the specified subset is independent of the projected key. For our purposes, we use two such families: one related to the CPA-secure encryption scheme, and the other related to the labeled CCA-secure scheme. Below we describe the notations we use for the family related to the labeled CCA-secure scheme. The notations for the other family are similar, except with the labels omitted.

The domain of the hash functions consists of triples of the form (message, label, ciphertext), with respect to a specific public key of the labeled CCA-secure scheme; the range of these functions will be some group $G$. We use here a variant of smooth projective hashing similar to that defined by Gennaro and Lindell [19]. Specifically, we consider hash families described by the following four components (all of which are assumed to be efficiently implementable):

$\mathcal{H}$ is a distribution ensemble of hashing keys, indexed by public keys of an encryption scheme. Namely, for an encryption public key $pke$, $\mathcal{H}(pke)$ is a distribution over hashing keys.

$\alpha$ is the projection function. It takes as input a hashing key, a label (for labeled CCA-secure encryption schemes), and a ciphertext, and outputs a projected key. A projection of a hash key $hk$ with respect to label $\ell$ and ciphertext $c$ is denoted $hp = \alpha(hk; \ell, c)$.

$H$ is the hashing algorithm. It takes a hashing key $hk$ and an element $(m, \ell, c)$ of the domain, and returns an element $g$ in the range. We denote this by $g = H_{hk}(m, \ell, c)$.

$h$ is the "projected hashing" algorithm. It takes a projected key $hp$, an element $(m, \ell, c)$ of the domain, *and a string $r$, supposedly the randomness that was used to generate the ciphertext $c$.* It returns an element $g$ in the range. We denote this by $g = h_{hp}(m, \ell, c; r)$.

The properties that we need of this family are the following:

**Projection.** For any encryption public key $pke$, any message $m$, label $\ell$, and randomness $r$, and any hashing key $hk \in \mathcal{H}(pke)$, if we set $c = E_{pke}(m; \ell; r)$ and $hp = \alpha(hk; \ell, c)$, then $H_{hk}(m, \ell, c) = h_{hp}(m, \ell, c; r)$.

**Smoothness.** Fix any encryption public key $pke$, any $m, \ell$, and any $c \in E_{pke}$ such that $c$ *is not* an encryption of $m$ with respect to $pke$ and label $\ell$.[9] Then, for a random hashing key $hk \in \mathcal{H}(pke)$ and $hp = \alpha(hk; \ell, c)$, the element $g = H_{hk}(m, \ell, c)$ is (statistically close to) uniform in the group $G$, independent of $(hp, m, \ell, c)$.

Gennaro and Lindell [19] describe CPA-secure and CCA-secure encryption schemes with associated smooth projective hashing families, based on a variety of (standard) number-theoretic cryptographic assumptions. These constructions can be used here, with straightforward modifications to incorporate labels. (Specifically, we need to add the label as an argument to the collision-resistant hash function; see [19].)

**Simulation-sound zero-knowledge (SSZK) proofs [30, 15, 18, 27].** As we have described in Section 3.1, a zero-knowledge proof system is said to be (unbounded) *simulation-sound* if it has the property that an adversary cannot give a convincing proof for a false statement, even if the adversary has oracle access to the zero-knowledge simulator (except, of course, by copying proofs that were directly obtained from the simulator). We do not provide a formal definition here but refer the reader to [18] instead. We denote a proof protocol for the statement $x \in L$ in which the verifier has input $x$ and the prover has input $x, y$ by:

$$\left\{ \frac{\text{PRV}(x, y) \quad \longleftrightarrow \quad \text{VRF}(x)}{x \in L} \right\}$$

**One-time signatures.** We assume the reader is familiar with the standard notion of a one-time signature scheme [17]. For this, we denote the signing algorithm by $\sigma = \mathsf{Sign}_{sk}(m)$ and the verification by $\mathsf{Verify}_{vk}(m, \sigma)$. We say that a one-time signature scheme is secure if it is existentially unforgeable against an adaptive chosen one-message attack.

## 4.2 The Protocol

The full description of our password-based key-exchange protocol $\Pi$ can be found in Figure 5. It follows the high-level ideas presented in Figure 4.

[9] We note that the constructions of [19] hold for a more relaxed notion of smoothness. Specifically, there exists

**UC Password Protocol $\Pi$**

**Common reference string:** A triple $(pke, pke', \gamma)$, where $pke$ is a public key for a labeled CCA-secure encryption scheme $E$, $pke'$ is a public key for a CPA-secure encryption scheme $E'$, and $\gamma$ is a reference string for a simulation-sound zero-knowledge proof system for the language $L_{pke'}$ defined below.

**Protocol Steps:**

0. When $P_i$ is activated with input $(\mathsf{NewSession}, sid, ssid, i, j, w, \text{role})$, it does the following. If role = $\mathsf{server}$ it does nothing (except waiting for a $\mathsf{flow\text{-}zero}$ message as described below). If role = $\mathsf{client}$, it chooses $r_0$, encrypts $c_0 = E'_{pke'}(w; r_0)$, and sends the message $(\mathsf{flow\text{-}zero}, c_0)$ to $P_j$. From this point on, assume that $P_i$ is a party activated with input $(\mathsf{NewSession}, sid, ssid, i, j, w, \mathsf{client})$, and that $P_j$ is a party activated with input $(\mathsf{NewSession}, sid, ssid, j, i, w', \mathsf{server})$.

1. When $P_j$ (who was waiting for a $\mathsf{flow\text{-}zero}$ message) receives a message $(\mathsf{flow\text{-}zero}, c_0)$, it checks that $c_0 \in E'_{pke'}$ as defined in Section 4.1. Then it generates a key pair $(VK, SK)$ for a one-time signature scheme, chooses $r_1$, sets the label $\ell_1 = j \circ i \circ ssid \circ VK$, encrypts $c_1 = E_{pke}(w'; \ell_1; r_1)$, and sends $(\mathsf{flow\text{-}one}, c_1, VK)$ to $P_i$.

2a. When $P_i$ receives a message $(\mathsf{flow\text{-}one}, c_1, VK)$, it checks that $c_1 \in E_{pke}$. Then it chooses a key $hk$ for the smooth projective hash function family $\mathcal{H}$ with respect to $pke$, sets the label $\ell_1 = j \circ i \circ ssid \circ VK$, and computes the projection $hp = \alpha(hk; \ell_1, c_1)$. Next it chooses $r_2$, encrypts $c_2 = E'_{pke'}(w; r_2)$, and sends $(\mathsf{flow\text{-}two}, hp, c_2)$ to $P_j$.

2b. Let $L_{pke'}$ be the language of pairs of ciphertexts that encrypt the same message w.r.t. $pke'$:

$$L_{pke'} \stackrel{\text{def}}{=} \{(c_0, c_2) \mid \exists\, (w, r_0, r_2) \text{ s.t. } c_0 = E'_{pke'}(w; r_0) \text{ and } c_2 = E'_{pke'}(w; r_2)\}$$

Player $P_i$ proves to $P_j$ that $(c_0, c_2) \in L_{pke'}$, by engaging in a simulation-sound zero-knowledge proof protocol using the witness $w, r_0, r_2$ that it knows and the reference-string $\gamma$. Namely, the players engage in a proof protocol

$$\left\{ \frac{\text{PRV}(c_0, c_2, \gamma; w, r_0, r_2) \;\longleftrightarrow\; \text{VRF}(c_0, c_2, \gamma)}{(c_0, c_2) \in L_{pke'}} \right\}$$

3. When $P_j$ receives a message $(\mathsf{flow\text{-}two}, hp, c_2)$, it checks that $c_2 \in E'_{pke'}$. Then it plays the verifier role in the simulation-sound zero-knowledge proof protocol (as discussed in the previous step). If $P_j$ rejects the proof, it aborts. Otherwise, it does the following:

   (a) Choose a key $hk'$ for the smooth projective hash family $\mathcal{H}'$ with respect to $pke'$ and compute the projection $hp' = \alpha'(hk'; c_2)$.

   (b) Compute $\sigma = \mathsf{Sign}_{SK}(c_0, c_2, hp, hp')$.

   $P_j$ then sends $(\mathsf{flow\text{-}three}, hp', \sigma)$ to $P_i$, computes session key $sk' = h_{hp}(w', \ell_1, c_1; r_1) + H'_{hk'}(w', c_2)$, outputs $(sid, ssid, sk')$, and terminates the session.

4. When $P_i$ receives a message $(\mathsf{flow\text{-}three}, hp', \sigma)$, it checks that $\mathsf{Verify}_{VK}((c_0, c_2, hp, hp'), \sigma) = 1$. If not, it aborts the session outputting nothing. Otherwise, it computes session key $sk = H_{hk}(w, \ell_1, c_1) + h'_{hp'}(w, c_2; r_2)$, outputs $(sid, ssid, sk)$, and terminates the session.

Figure 5: A framework for universally composable password-based key exchange

# 5 Proof of security

## 5.1 Some Aspects of the Universal Composability (UC) Framework

We assume basic familiarity with the workings of the UC framework, and refer the reader to [9] for complete details. As a reminder, in this framework the security of a task is captured via an ideal functionality, which is essentially the code for a "trusted party" that interacts with parties in a network. The formal model for testing whether a protocol $\pi$ realizes an ideal functionality $\mathcal{F}$ involves an environment $\mathcal{Z}$ that provides inputs to and obtains outputs from either **(a)** parties running a *single execution* of $\pi$, plus an adversary $\mathcal{A}$ that controls some of the parties and *all* the communication, or **(b)** "dummy parties" that communicate only with $\mathcal{F}$ by sending it their inputs and receiving back outputs, plus an adversary $\mathcal{S}$ that also interacts with $\mathcal{F}$. Protocol $\pi$ is said to securely realize $\mathcal{F}$ if for every $\mathcal{A}$ there exists an $\mathcal{S}$ such that $\mathcal{Z}$ can not distinguish whether it is in case (a) or case (b) with any non-negligible advantage over a random guess. The universal composition theorem then asserts that any larger protocol that uses multiple copies of $\mathcal{F}$ continues to behave essentially the same when the copies of $\mathcal{F}$ are replaced by instances of protocol $\pi$. Intuitively, one can view this larger protocol as modeling arbitrary network activity. The composition theorem thus guarantees that a secure protocol behaves like the ideal functionality, even when run in an arbitrary network.

In this work, we focus on static adversaries that cannot corrupt parties during the execution. (Nevertheless, as we show in Appendix A, this implies the "weak corruption" model of [3] in which passwords can be adaptively obtained.)

**The "canonical real-world adversary".** In the UC framework, the real-life adversary is allowed to follow any arbitrary PPT strategy. However, it has been shown [9] that the notion of security remains unchanged even if we restrict attention to the case of a "dummy adversary" that only reports all messages sent by the parties to the environment, and exactly follows the instructions of the environment regarding what messages to deliver to which parties. In this work we concentrate on this simpler (but equivalent) formulation of the UC notion of security, and we sometimes refer to this dummy adversary as the "canonical real-world adversary".

**Session identifiers.** The communication model of the UC framework differs from some other models in the way it uses session IDs. In the UC framework there are potentially many copies of any ideal functionality $\mathcal{F}$, and each copy is assumed to have a unique identifier, called the session ID (SID) of that copy. Each message sent to a copy of $\mathcal{F}$ should contain the SID of that copy, and each message from a copy of $\mathcal{F}$ to a party contains the SID of that copy. Similarly, a protocol $\pi$ that realizes $\mathcal{F}$ expects each input to contain the SID that is unique to that copy, and each output contains the SID. This formalism puts the burden of choosing the SIDs on the calling protocol. In particular, this implies that the calling protocol must make sure that all the parties that communicate with the same copy of $\mathcal{F}$ (or alternatively, participate in an instance of $\pi$) use the same SID, and that different copies use different SIDs. This choice simplifies some technicalities, almost without any loss of generality. Indeed it was shown in [1] that in most cases there are simple ways for realizing this formalism. The case of two-party protocols is particularly

a hard-to-sample subset of tuples $(m, \ell, c)$ for which $c$ is not an encryption of $m$ (w.r.t. $pke$ and label $\ell$) and yet smoothness is not required to hold. Since this set is hard-to-sample it makes no difference to our proof. See [19, Sections 8.1 & 8.2] for details.

simple: a two-party protocol that expects an input SID can be preceded by an exchange of nonces, and then the original protocol can be run using the concatenation of the nonces as the input SID.

**Joint state and multi-session extensions.** The protocols developed in this work remain secure even when different sessions use the same common reference string (as is clearly necessary in practice). For this purpose, we use the "universal composability with joint state" (JUC) formalism that was introduced by Canetti and Rabin [12]. The problem addressed by that work is that the original UC composition theorem from[9] guarantees that security of a protocol $\rho$ is preserved under arbitrary composition *only as long as the executions of the different instances of $\rho$ are independent of each other*. In our case, this independence does not hold since all executions use the same common reference string.

The formalism from [12] provides a "wrapper layer" above the functionality that deals with "joint state" among different copies of the protocol. Specifically, defining a functionality $\mathcal{F}$ also defines (implicitly) the multi-session extension of $\mathcal{F}$ (denoted $\hat{\mathcal{F}}$) as follows: $\hat{\mathcal{F}}$ runs multiple independent copies of $\mathcal{F}$, where the copies are distinguished via sub-session IDs (SSIDs). In this extension, each incoming message to $\hat{\mathcal{F}}$ is assumed to have (in addition to the standard SID) an additional identifier, the SSID. The message is then forwarded by $\hat{\mathcal{F}}$ to the internal copy of $\mathcal{F}$ whose SID is the SSID contained in the message. If no such copy exists then a new one is invoked.

Then, instead of proving that a protocol $\rho$ realizes the functionality $\mathcal{F}$, one proves that it realizes the multi-session extension $\hat{\mathcal{F}}$. The JUC theorem [12] asserts that composing a protocol $\pi$ that uses multiple independent copies of $\mathcal{F}$, with a *single copy* of a protocol $\rho$ that realizes $\hat{\mathcal{F}}$, preserves the security of $\pi$. This holds even for the special case where $\pi$ consists of multiple independent instances of a simpler protocol (e.g., multiple exchanges of a key), and each instance calls different copies of $\mathcal{F}$. In a nutshell, this means that once we have proven that our password-based key-exchange protocol securely realizes $\hat{\mathcal{F}}_{\mathsf{pwKE}}$, all higher-level protocols can refer to the simpler single-session functionality $\mathcal{F}_{\mathsf{pwKE}}$, and security is preserved even if all of these calls use the same common reference string.

## 5.2 Our Main Theorem

Let $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ be the multi-session extension of $\mathcal{F}_{\mathsf{pwKE}}$ and let $\mathcal{F}_{\mathsf{crs}}$ be the ideal functionality that provides a common reference string to all parties.

**Theorem 1** *(main theorem – restated): Assume that $E$ is a CCA-secure labeled encryption scheme, that $E'$ is a CPA-secure encryption scheme, that $\mathcal{H}$ is a family of smooth projective hash functions with respect to $E$, that $\mathcal{H}'$ is a family of smooth projections hash functions with respect to $E'$, that the proof system is simulation-sound zero-knowledge, and that the one-time signature scheme is secure. Then protocol $\Pi$ of Figure 5 securely realizes the $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ functionality in the $\mathcal{F}_{\mathsf{crs}}$-hybrid model, in the presence of static-corruption adversaries.*

In order to prove this theorem, we need to show that for any real-world adversary $\mathcal{A}$ (interacting with parties running $\Pi$), there is an ideal-world adversary/simulator $\mathcal{S}$ (interacting with dummy parties and the functionality $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$) such that any environment $\mathcal{Z}$ can distinguish between an execution with $\mathcal{A}$ in the real world and $\mathcal{S}$ in the ideal world with at most negligible probability. In Section 5.3 we describe the simulator $\mathcal{S}$, and in Section 5.4 we prove indistinguishability between $\mathcal{A}$ in the real world and $\mathcal{S}$ in the ideal world.

### 5.3 Description of the Simulator

When initialized with security parameter $k$, the simulator first runs the key-generation algorithms of the encryption schemes $E$ and $E'$, both with security parameter $k$, thus obtaining key pairs $(ske, pke)$ and $(ske', pke')$. The simulator also chooses $\hat{w}$ at random from the intersection of the domains of $E$ and $E'$.[10] $S$ also uses the first part of the zero-knowledge simulator $S_{ZK}$ to obtain $(\tau, \gamma) \leftarrow S^1_{ZK}(k)$. Then $S$ initializes the real-world adversary $\mathcal{A}$, giving it the triple $(pke, pke', \gamma)$ as the common reference string. Thereafter, the simulator $S$ interacts with the environment $Z$, the functionality $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$, and its subroutine $\mathcal{A}$. For the most part, this interaction is implemented by the simulator $S$ just following the protocol $\Pi$ on behalf of all the honest players. The only differences between the simulated players and the real honest players are in the passwords that $S$ uses to simulate the players, and in the behavior of $S$ in the zero-knowledge proof protocol. Specifically, when $S$ starts a simulation of a session $ssid$ for an uncorrupted party $P$, it runs $\Pi$ using the dummy password $\hat{w}$. However, if $\mathcal{A}$ modifies a flow-zero or flow-one message that is delivered to $P$ in session $ssid$, then $S$ decrypts that ciphertext (using $ske'$ or $ske$) and uses the recovered message $w$ in a TestPwd query to $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$. If this is a "correct guess", then $S$ replaces the "dummy password" $\hat{w}$ in the state of session $ssid$ of $P$ with the "correct password" $w$, and proceeds with the simulation. In addition, instead of following the honest prover strategy, $S$ uses the zero-knowledge simulator in all the proofs. More details follow.

#### 5.3.1 Session Initialization

On message $(\mathsf{NewSession}, sid, ssid, i, j, \mathrm{role})$ from $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$, $S$ starts simulating a new session of the protocol $\Pi$, for party $P_i$, peer $P_j$, session identifier $ssid$, and common reference string $(pke, pke', \gamma)$. We denote this session $(P_i, ssid)$. $S$ initializes the session $(P_i, ssid)$ with the "dummy password" $\hat{w}$. If role $=$ client, $S$ generates the flow-zero message as in $\Pi$: it chooses $r_0$, generates a ciphertext $c_0 = E'_{pke'}(\hat{w}; r_0)$, and gives $(\mathsf{flow\text{-}zero}, c_0)$ to $\mathcal{A}$ on behalf of $(P_i, ssid)$.

#### 5.3.2 Protocol Messages

Assume $\mathcal{A}$ sends a message $M$ to an active session of some party. If this message is formatted differently from what is expected by the session, then $S$ aborts that session and notifies $\mathcal{A}$. Otherwise, we have the following cases (where we denote a party in the client role as $P_i$ and a party in the server role as $P_j$):

**Step 1.** Assume a session $(P_j, ssid)$ receives a message $M = (\mathsf{flow\text{-}zero}, c)$. As per the protocol, it must be the case that $P_j$ is a server and $M$ is the first message received by this instance. If $c$ is not equal to any ciphertext $c_0$ that was generated by $S$ for a flow-zero message, then $S$ uses its secret key $ske'$ to decrypt the ciphertext and obtain $w = D_{ske'}(c)$,[11] and then $S$ makes a call $(\mathsf{TestPwd}, sid, ssid, j, w)$ to $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$. If this is a "correct guess", then $S$ resets the password of this server session to $w$. (If this is a "wrong guess" then $S$ keeps its password as $\hat{w}$.)

Either way, $S$ generates the flow-one message as in $\Pi$. Denote by $w'$ the password now held by this session. Then $S$ generates $(VK, SK)$ for a one-time signature scheme, chooses $r_1$, sets

[10] we assume for convenience that the intersection of the domains of $E$ and $E'$ includes all passwords and is super-polynomial in size. In particular, $\hat{w}$ will be different from all passwords input to the protocol with all but negligible probability (and we assume this to be the case).

[11] Decryption may result in $w = \perp$. In this case $S$ will set $w = \hat{w}$, to ensure that $w$ is a "wrong guess" in a TestPwd query. $S$ will make an analogous substitution in Step 2 below.

$\ell_1 = j \circ i \circ ssid \circ VK$, computes the ciphertext $c_1 = E_{pke}(w'; \ell_1; r_1)$, and gives $(\mathsf{flow\text{-}one}, c_1, VK)$ to $\mathcal{A}$ on behalf of $(P_j, ssid)$.

**Step 2.** Assume a session $(P_i, ssid)$ receives a message $M = (\mathsf{flow\text{-}one}, c, VK')$. As per the protocol, it must be the case that $P_i$ is a client who sent a flow-zero message and is now waiting for the response.

Say $(P_j, ssid')$ is a *peer session* (to $(P_i, ssid)$) if $ssid = ssid'$, session $(P_i, ssid)$ has peer $P_j$, session $(P_j, ssid)$ has peer $P_i$, and these two sessions have opposite roles (client/server). If the pair $(c, VK')$ is not equal to the pair $(c_1, VK)$ that was generated by $S$ for a flow-one message from peer session $(P_j, ssid)$ (or if no such ciphertext was generated yet, or no such peer session exists) then $S$ sets $\ell_1 = j \circ i \circ ssid \circ VK$, uses its secret key $ske$ to compute $w = D_{ske}(c; \ell_1)$, and then $S$ makes a call $(\mathsf{TestPwd}, sid, ssid, i, w)$ to $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$. If this is a "correct guess", $S$ resets the password of this client session to $w$. (If this is a "wrong guess" then $S$ keeps the password as $\hat{w}$.)

Either way, $S$ generates the flow-two message as in $\Pi$. Denote by $w'$ the password now held by this instance. $S$ chooses a key $hk$ for the hash family $\mathcal{H}$ and computes the projection $hp = \alpha(hk; \ell_1, c)$. It chooses $r_2$, generates a ciphertext $c_2 = E'_{pke'}(w'; r_2)$, and gives $(\mathsf{flow\text{-}two}, hp, c_2)$ to $\mathcal{A}$ on behalf of $(P_i, ssid)$.

Next $S$ engages on behalf of $(P_i, ssid)$ in the proof protocol, but instead of using the honest prover strategy, it uses the second part of the zero-knowledge simulator, $S^2_{ZK}(\tau)$, to generate a simulated proof.

**Other messages.** All other messages from $\mathcal{A}$ are handled by $S$ by simply following the protocol $\Pi$ for each session (given the current state of the session). If a session aborts or terminates, $S$ reports it to $\mathcal{A}$. If the session terminates with a session key $sk$, then $S$ makes a NewKey call to $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$, specifying the session key $sk$. (But recall that unless the session is compromised, $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ will ignore the key specified by $S$.)

### 5.3.3 Other Interactions

Messages between $Z$ and $\mathcal{A}$ are simply forwarded.

## 5.4 Proof of Indistinguishability

Roughly, the differences between the views of the environment in the real world and the ideal world (with the above simulator $S$) are the following:

1. In the real world, the clients follow the honest prover strategy to prove that $c_0$, $c_2$ are encryptions of the same password. In the simulated world, these proofs are generated by the zero-knowledge simulator.

2. In the real world, any session that terminates successfully (and in particular any session where the adversary delivers all messages unchanged) returns to the environment the sum of hashes as the key. In the simulated world, on the other hand, fresh and interrupted sessions return a random key.

3. In the real world, all the sessions between two players $P_i$ and $P_j$ use the passwords given by the environment. In the simulated world all sessions use the "dummy password" $\hat{w}$, at least until a correct password guess occurs.

Accordingly, we describe a sequence of "hybrid experiments", starting from the real world and ending in the simulated world, where in each experiment we change some aspect of the simulation and show that the environment cannot detect that change. Basically we make changes following the order given above, except that to make the proof more tractable we divide changes (2) and (3) into smaller steps. In particular, we proceed through a series of ten hybrids where in each hybrid we change certain real session keys to random session keys and then let parties in those sessions use dummy passwords, since the encryptions in those sessions will not affect the (now randomly generated) session keys. Also, by dividing the hybrids in this way, we are able to highlight the particular security assumptions necessary for each change.

Note that for our proof we will assume all ciphertexts and one-time signature verification keys generated by honest parties are unique (not previously used by any parties or the adversary) and all $ssid$'s are unique. The first assumption is true with high probability, and the second is a requirement of the universal composability framework.

Let $\mathbf{H_0}$ be the real-world experiment. Say a value is *$t$-oracle generated* if it is generated in step $t$ in some session. For a client session $(P_i, ssid)$, a pair $(c_1, VK)$ is *peer-oracle generated* if it is 1-oracle generated by a peer session $(P_j, ssid)$. Say a ciphertext $c_1$ is *valid* for $(P_i, ssid)$ if $w = D_{ske}(c_1; \ell_1)$ where $w$ is the password given to $(P_i, ssid)$ by the environment and the label $\ell_1$ is the one that is computed by $(P_i, ssid)$. Say a ciphertext $c_0$ (resp., $c_2$) is *valid* for $(P_j, ssid)$ if $w' = D'_{ske'}(c_0)$ (resp., $w' = D'_{ske'}(c_2)$) where $w'$ is the password given to $(P_j, ssid)$ from the environment. Intuitively, a ciphertext is valid if it decrypts to the "correct" password. Say $(P_i, ssid)$ and $(P_j, ssid)$ are *matching sessions* if $(P_j, ssid)$ is a peer session of $(P_i, ssid)$ and the two sessions agree on the values of $VK$, $c_1$, and $c_2$. Then we have the following hybrid experiments:

$\mathbf{H_1}$ modifies how the zero-knowledge proofs are performed. Specifically, instead of using the honest-prover strategy, all the proofs in which the prover is an honest player are simulated using the zero-knowledge simulator. (Note that the $\gamma$ value in the common reference string is also simulated.) Since the proofs are (concurrent) zero-knowledge, the environment cannot distinguish between $\mathbf{H_1}$ and $\mathbf{H_0}$. That is, if an environment could distinguish between these hybrids, one could construct an adversary to break the (unbounded) zero-knowledge property of the proof protocol.

$\mathbf{H_2}$ computes $c_0$ as an encryption of the "dummy password" $\hat{w}$ for every client session $(P_i, ssid)$ (instead of computing $c_0$ as an encryption of the password $w$ given to this session by the environment). Since the encryption scheme is semantically secure, the environment cannot distinguish between $\mathbf{H_2}$ and $\mathbf{H_1}$. Note that the zero-knowledge proofs (ZKPs) are already simulated, and thus do not require knowledge of how $c_0$ is generated. Furthermore, this knowledge is not used anywhere else in the protocol. So if an environment could distinguish between these two hybrids, then one could construct an adversary to break the semantic security of $E'$ using a simple hybrid argument.[12]

$\mathbf{H_3}$ substitutes a random session key $sk$ for each client session $(P_i, ssid)$ that receives a peer-oracle-generated pair $(c_1, VK)$ from a peer session $(P_j, ssid)$ in step 2. If $(P_i, ssid)$ and $(P_j, ssid)$ are matching sessions and have the same password given to them by the environment, and $(P_j, ssid)$ received a 0-oracle-generated $c_0$, then the same random $sk$ is given to $(P_j, ssid)$. (Actually, $sk$ is given to $P_j$ first, since $(P_j, ssid)$ terminates first.) In Section 5.5 we prove the environment cannot distinguish between $\mathbf{H_3}$ and $\mathbf{H_2}$.

[12]Note that the hybrids we refer to in this reduction are "sub-hybrids" between $\mathbf{H_2}$ and $\mathbf{H_1}$, based on the number of ciphertexts $c_0$ generated by honest parties.

**$\mathbf{H_4}$** computes $c_2$ as an encryption of the "dummy password" $\hat{w}$ for every client session $(P_i, ssid)$ that receives a peer-oracle-generated pair $(c_1, VK)$ in step 2 (instead of computing $c_2$ as an encryption of the password $w$ given to this session by the environment). Then, as in $\mathbf{H_2}$, since the encryption scheme $E'$ is semantically secure, the environment cannot distinguish between these cases. Note that the session key of such client sessions are already chosen randomly, and the ZKPs are already simulated. Thus, neither depend on how $c_2$ is generated.

**$\mathbf{H_5}$** substitutes a random session key $sk$ for all server sessions $(P_j, ssid)$ that receive a 2-oracle-generated ciphertext $c_2$ from a client session $(P_i, ssid')$ in step 3, where $(P_i, ssid')$ received a peer-oracle-generated pair $(c_1, VK)$ in step 2. If $(P_i, ssid')$ and $(P_j, ssid)$ are matching sessions and were given the same password by the environment, and $(P_j, ssid)$ received a 0-oracle-generated $c_0$, then the session key of $(P_j, ssid)$ is already random (from $\mathbf{H_3}$), and is not changed in this hybrid. In Section 5.5 we prove the environment cannot distinguish between $\mathbf{H_5}$ and $\mathbf{H_4}$.

**$\mathbf{H_6}$** checks whether ciphertext $c_1$ is valid in client sessions $(P_i, ssid)$ that receive a non-peer-oracle-generated pair $(c_1, VK)$ in step 2. (To perform this validity test, one needs to generate the key pair $(ske, pke)$ and use $pke$ in the common reference string.) If not, the session key $sk$ is chosen randomly. This change is indistinguishable since $H_{hk}(w, \ell_1, c_1)$ is statistically close to uniform, given $pke$, $hp$, $w$, $\ell_1$, and $c_1$.

**$\mathbf{H_7}$** computes $c_2$ as an encryption of the "dummy password" $\hat{w}$ for every client session $(P_i, ssid)$ that receives a non-peer-oracle-generated pair $(c_1, VK)$ in step 2 with non-valid ciphertext $c_1$ (instead of computing $c_2$ as an encryption of the password $w$ given to this session by the environment). Note that the session key of such client sessions are already chosen randomly, and the ZKPs are already simulated; thus, neither depend on how $c_2$ is generated. Also note that in the reduction to the security of $E'$ with public key $pke'$, one can still test validity of $c_1$ encryptions (as required for hybrid $\mathbf{H_6}$), since such ciphertexts are encrypted with the other public key $pke$ and can be decrypted using $ske$.

As in $\mathbf{H_2}$ and $\mathbf{H_4}$, since the encryption scheme $E'$ is semantically secure the environment cannot distinguish between $\mathbf{H_6}$ and $\mathbf{H_7}$.

**$\mathbf{H_8}$** substitutes a random session key $sk'$ for all server sessions $(P_j, ssid)$ that receive a 0-oracle-generated ciphertext $c_0$ in step 1. In Section 5.5 we prove the environment cannot distinguish between $\mathbf{H_8}$ and $\mathbf{H_7}$.

**$\mathbf{H_9}$** checks whether ciphertext $c_0$ is valid in server sessions $(P_j, ssid)$ that receive a non-0-oracle-generated ciphertext $c_0$ in step 1. (To perform this validity test, one needs to generate the key pair $(ske', pke')$ and set $pke'$ in the common reference string.) If not, the session key $sk'$ is chosen randomly. This is indistinguishable since $H'_{hk'}(w, c_2)$ is statistically close to uniform, given $pke'$, $hp'$, $w$, and $c_2$. Note that we rely on the simulation-soundness of the zero-knowledge proof here to guarantee that $c_2$ is not valid for $(P_j, ssid)$. In particular, no client session $(P_i, ssid')$ will generate a ZKP for a tuple with $c_0$ as the first component, and thus (by simulation-soundness) any proof with $c_0$ as first component must, except with negligible probability, be of a valid tuple $(c_0, c_1, \ell_1, c_2)$. But then $c_2$ is not valid for $(P_j, ssid)$, since $c_0$ is not valid for $(P_j, ssid)$.

**$\mathbf{H_{10}}$** computes $c_1$ as an encryption of the "dummy password" $\hat{w}$ for every server session $(P_j, ssid)$ that receives a 0-oracle-generated ciphertext $c_0$ in step 1 or a non-0-oracle-generated and non-valid ciphertext $c_0$ in step 1 (this is instead of computing $c_1$ as an encryption of the password

$w$ given to this session by the environment). Since $E$ is CCA-secure, the environment cannot distinguish between $\mathbf{H_9}$ and $\mathbf{H_{10}}$. (Note that the session key $sk'$ is already chosen randomly in this case, and thus does not depend on $c_1$. Also note that in the reduction to the security of $E$ with public key $pke$, one can still test the validity of $c_0$ encryptions since they are encrypted with the other public key $pke'$ and can be decrypted using $ske'$, and one can use the decryption oracle to decrypt any non-peer-oracle-generated encryption $c_1$.)

Now we argue that $\mathbf{H_{10}}$ is perfectly indistinguishable from the ideal world experiment with simulator $S$, as long as the adversary does not forge a one-time signature corresponding to a verification key sent by a server session. By the security of the one-time signature scheme, this will imply that the real world experiment and ideal world experiment are indistinguishable. We use $\mathsf{IWE}$ to denote the ideal world experiment with simulator $S$.

First we show that the behavior of a server session $(P_j, ssid)$ is identical. We break this into two cases, depending on whether the ciphertext $c_0$ received by $(P_j, ssid)$ was 0-oracle-generated. For the case of 0-oracle-generated $c_0$, we divide this into subcases depending on whether $(P_j, ssid)$ has a matching session $(P_i, ssid)$ or not. For the case of non-0-oracle-generated $c_0$, we divide this into subcases depending on whether the ciphertext $c_0$ is valid or not.

1. *0-oracle-generated $c_0$:* In both $\mathbf{H_{10}}$ and $\mathsf{IWE}$, since $(P_j, ssid)$ receives a 0-oracle-generated $c_0$, we have that $c_1$ is an encryption of $\hat{w}$ (see $\mathbf{H_{10}}$ and step 1 of $\mathsf{IWE}$).

   In $\mathbf{H_{10}}$, if $(P_j, ssid)$ does not abort, its session key will be random (see $\mathbf{H_8}$). In $\mathsf{IWE}$, the session key will also be random, since no $\mathsf{TestPwd}$ query is made to $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ for $(P_j, ssid)$, and thus the ideal server session will be fresh. However, we still must show that the session key is set equal to the session key of a client session $(P_i, ssid')$ in $\mathbf{H_{10}}$ if and only if it is set equal to the session key of $(P_i, ssid')$ in $\mathsf{IWE}$.

   (a) *matching session $(P_i, ssid)$:*
   Note that $(P_j, ssid)$ is the peer session of $(P_i, ssid)$. First consider the case where the same password is given by the environment to both $(P_j, ssid)$ and $(P_i, ssid)$. In $\mathbf{H_{10}}$, the session key of $(P_j, ssid)$ will be set to the (random) session key of $(P_i, ssid)$ (see $\mathbf{H_3}$). To see that this is what occurs in the ideal world, note that the functionality $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ will receive $(\mathsf{NewSession}, sid, ssid, i, j, w, \mathrm{role})$ and $(\mathsf{NewSession}, sid, ssid, j, i, w, \mathrm{role})$ queries, and $S$ will not send any $(\mathsf{TestPwd}, sid, ssid, j, w')$ or $(\mathsf{TestPwd}, sid, ssid, i, w')$ queries (since $c_0$ is 0-oracle-generated and $(c_1, VK)$ is from the peer session of $(P_i, ssid)$). Thus, $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ will return a matching (random) key in the $\mathsf{NewKey}$ query for each session.

   Now consider the case where different passwords are passed in from the environment. In $\mathbf{H_{10}}$, the session key $sk$ of $(P_j, ssid)$ will be independent of the session key of any client session, because $sk$ is not set in $\mathbf{H_3}$. In $\mathsf{IWE}$, the session key of $(P_j, ssid)$ will also be independent of the session key of any client session, by definition of $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$.

   (b) *no matching conversation:*
   It is clear that in $\mathbf{H_{10}}$, the session key $sk$ for $(P_j, ssid)$ is independent of the session key of any client session, because $sk$ is not set in $\mathbf{H_3}$. In $\mathsf{IWE}$, note that the only way $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ will generate matching keys is if $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ receives $(\mathsf{NewSession}, sid, ssid, i, j, w, \mathrm{role})$ and $(\mathsf{NewSession}, sid, ssid, j, i, w, \mathrm{role})$ queries, and $S$ sends $\mathsf{NewKey}$ queries for sessions $(P_i, ssid)$ and $(P_j, ssid)$ without sending any $\mathsf{TestPwd}$ queries for those sessions. But if these two sessions do not have a matching conversation, then they must differ in either $VK$, $c_1$, or $c_2$. If they differ in $c_1$ or $VK$, $S$ will send a $\mathsf{TestPwd}$ query for session

$(P_i, ssid)$, because $(c_1, VK)$ will be non-peer-oracle-generated. If they differ only in $c_2$, then by the security of the one-time signature scheme, $(P_i, ssid)$ will abort, because the signature generated by $(P_j, ssid)$ will be for a different $c_2$, and we have assumed no forgeries.

2. *non-0-oracle-generated* $c_0$*:*

   (a) $c_0$ *is valid:* In both $\mathbf{H_{10}}$ and IWE, we will show that since $(P_j, ssid)$ receives a valid non-0-oracle-generated $c_0$, the ciphertext $c_1$ and the session key are generated as in the real protocol.

   First we examine the generation of $c_1$. Computation of $c_1$ in the case we are dealing with here (i.e., when the incoming $c_0$ is valid and non-0-oracle-generated) is not changed in any of the hybrid games $\mathbf{H_1}$–$\mathbf{H_{10}}$, and so $c_1$ is generated as in the real protocol in this case. In IWE, $S$ determines $w$ by making a TestPwd query in Step 1 that returns "correct guess." Then $S$ will generate $c_1$ as in the real protocol.

   Now we examine the generation of the session key. Generation of the session key in the case we are dealing with here is not changed in any of the hybrid games (at least not without $(P_j, ssid)$ aborting). In particular, note that $(P_j, ssid)$ will not have its session key set in $\mathbf{H_3}$, because it received a non-0-oracle-generated $c_0$. Also, it will not have its session key set in $\mathbf{H_5}$ without aborting, because every 2-oracle-generated $c_2$ generated by a session $(P_i, ssid)$ that received a peer-oracle-generated $(c_1, VK)$ is invalid (see $\mathbf{H_4}$), and — by simulation-soundness of the ZKP — if $(P_j, ssid)$ receives a 2-oracle-generated $c_2$, it will reject any ZKP generated (i.e., simulated) by any client (because the client will have a different $c_0$) or produced by the adversary (because $c_0$ is valid but $c_2$ is not).

   In IWE, recall that $S$ will determine $w$ by making a TestPwd query in Step 1 that returns "correct guess." Then $S$ will run $(P_j, ssid)$ as in the real protocol. At this point, the ideal session is compromised, so when $S$ calls NewKey with the $sk$ value it has generated, $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ will set the session key to $sk$.

   (b) $c_0$ *is invalid:* In $\mathbf{H_{10}}$, since $(P_j, ssid)$ receives an invalid non-0-oracle-generated $c_0$, the ciphertext $c_1$ is an encryption of $\hat{w}$ (see $\mathbf{H_{10}}$). In IWE, $c_1$ is also an encryption of $\hat{w}$, since $S$ makes a TestPwd query in Step 1 that returns "wrong guess."

   In $\mathbf{H_{10}}$, the session key of $(P_j, ssid)$ is generated randomly (see $\mathbf{H_9}$). Also, it is independent of the session key generated by any client session. The reason is that it was not set in $\mathbf{H_3}$ since $(P_j, ssid)$ received a non-0-oracle-generated $c_0$. In IWE, the session key of $(P_j, ssid)$ is random and independent of any other session keys because the TestPwd query causes the ideal session to be interrupted.

Now we argue that the behavior of a client session $(P_i, ssid)$ is identical in $\mathbf{H_{10}}$ and IWE. First note that in both $\mathbf{H_{10}}$ and IWE, every $c_0$ generated by a client is an encryption of $\hat{w}$, and every ZKP is simulated. (See $\mathbf{H_1}$ and $\mathbf{H_2}$.)

We break the remainder of the analysis into two cases, depending on whether the pair $(c_1, VK)$ received by $(P_i, ssid)$ was peer-oracle generated or not. For the case of non-peer-oracle-generated $(c_1, VK)$, we divide this into subcases depending on whether $c_1$ is valid.

1. *peer-oracle-generated* $(c_1, VK)$*:*

In $\mathbf{H_{10}}$, since $(P_i, ssid)$ receives a peer-oracle-generated pair $(c_1, VK)$ (say from a peer session $(P_j, ssid)$), $c_2$ is an encryption of $\hat{w}$ (see $\mathbf{H_4}$). In IWE, $c_2$ is also an encryption of $\hat{w}$ (see step 2).

In $\mathbf{H_{10}}$, the session key of $(P_i, ssid)$ will be random (see $\mathbf{H_3}$). In IWE, the session key of $(P_i, ssid)$ will also be random, since $S$ will not call TestPwd for this session, and thus the ideal session will be fresh. (Whether the session key is equal to the session key of a server session $(P_j, ssid)$ is taken care of in our analysis of the server sessions above.)

2. *non-peer-oracle-generated* $(c_1, VK)$*:* Note that there will be no matching session, since no peer server session $(P_j, ssid)$ generated $(c_1, VK)$.

   (a) $c_1$ *valid:* In both $\mathbf{H_{10}}$ and the ideal world, we will show that since $(P_i, ssid)$ receives a valid, non-peer-oracle-generated pair $(c_1, VK)$, the ciphertext $c_2$ and the session key are generated as in the real protocol.

   First we examine the generation of $c_2$. Computation of $c_2$ in the case we are dealing with here (i.e., when the incoming $c_1$ is valid and non-peer-oracle-generated) is not changed in any of the hybrid games $\mathbf{H_1}$–$\mathbf{H_{10}}$, so it is generated as in the real protocol. In IWE, $S$ will determine $w$ by making a TestPwd query in Step 2 that returns "correct guess." Then $S$ will generate $c_2$ as in the real protocol.

   Now we examine the generation of the session key. Computation of the session key in the case we are dealing with here is not changed in any of the hybrid games, so it is computed as in the real protocol. In particular, note that $(P_i, ssid)$ will not have its session key set in $\mathbf{H_3}$ because it received a non-peer-oracle-generated $c_1$. Also, such a session will not have its session key set in $\mathbf{H_6}$ because $c_1$ is valid.

   In IWE, $S$ will determine $w$ by making a TestPwd query in step 2 that returns "correct guess." Then $S$ will run $(P_i, ssid)$ as in the real protocol. At this point, the ideal session is compromised, so when $S$ calls NewKey with the $sk$ value it has generated, $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ will set the session key to $sk$.

   (b) $c_1$ *invalid:* In $\mathbf{H_{10}}$, since $(P_i, ssid)$ receives a non-peer-oracle-generated pair $(c_1, VK)$ with invalid $c_1$, ciphertext $c_2$ is an encryption of $\hat{w}$ (see $\mathbf{H_7}$). In IWE, $c_2$ is also an encryption of $\hat{w}$ since $S$ makes a TestPwd query in Step 2 that returns "wrong guess."

   In $\mathbf{H_{10}}$, the session key of $(P_i, ssid)$ is generated randomly (see $\mathbf{H_6}$), independent of any server session. In IWE, the session key of $(P_j, ssid)$ is random and independent of any other session keys because the TestPwd query causes the ideal session to be interrupted

### 5.5 Further Details

**Indistinguishability of $\mathbf{H_3}$ and $\mathbf{H_2}$** First we show that no PPT environment can distinguish $\mathbf{H_3}$ from $\mathbf{H_2}$ with non-negligible probability. We split the proof into two parts. Define hybrid Mix just like $\mathbf{H_3}$ except that a random $sk$ is only substituted when $c_1$ is invalid.

- *Indistinguishability of* Mix *and* $\mathbf{H_2}$*:* By the definition of smooth projective hashing, given an invalid $c_1$, with $P_i$'s password $w$ and the label $\ell_1$ computed by $P_i$, the distribution $\{pke, c_1, \ell_1, w, hp, H_{hk}(w, \ell_1, c_1)\}$ is statistically close to $\{pke, c_1, \ell_1, w, hp, g\}$, where $g \leftarrow_R G$. Since the session key $sk$ for $P_i$ has $H_{hk}(w, \ell_1, c_1)$ as a component, the session key generated is statistically close to uniform, and thus contributes negligibly to the distinguishability of Mix and $\mathbf{H_2}$.

Note that if $c_1$ is invalid, then the corresponding sessions of $P_i$ and $P_j$ either do not have a matching conversation, or were given different passwords by the environment, so $P_j$ will not be given the same $sk$ as $P_i$ in Mix.

- *Indistinguishability of* $\mathbf{H_3}$ *and* Mix*:* We show that any environment that distinguishes $\mathbf{H_3}$ from Mix in this case can be used to construct a distinguisher between Expt-Hash and Expt-Unif as defined in [19][Section 3.3], violating Corollary 3.3 in that paper.

  We first state the explicit constructions of the experiments Expt-Hash and Expt-Unif in our terminology:

  Expt-Hash$(D)$: A public key $pke'$ is chosen and given to the machine $D$. $D$ can query two oracles: Encrypt$(\cdot)$ and Hash$(\cdot)$. Oracle Encrypt$(m)$ generates randomness $r$ and a ciphertext $c = E'_{pke'}(m; r)$. Oracle Hash$(c)$, for some $c$ output by Encrypt$(m)$, chooses $hk'$ from $\mathcal{H}'$ with respect to $pke'$ and computes the projection $hp' = \alpha'(hk', c)$; then it outputs $(hp', H'_{hk'}(m, c))$. (Note that Hash$(c)$ only answers when its input $c$ is output by the Encrypt oracle.) Finally, the output of Expt-Hash$(D)$ is the output of $D$.

  Expt-Unif$(D)$: This experiment is exactly the same as Expt-Hash$(D)$, except that the Hash oracle outputs a random element $g$ from $G'$ (where $G'$ is the range of $H'$) in place of the output of the smooth projective hash function.

  Now we show that $|\Pr[\text{Expt-Hash}(D) = 1] - \Pr[\text{Expt-Unif}(D) = 1]|$ bounds (to within a negligible amount) the probability of the environment distinguishing $\mathbf{H_3}$ from Mix.

  Let $D$ be a machine that receives a randomly chosen public key $pke'$ and emulates the experiment $\mathbf{H_3}$ with the following changes. On receiving a flow-one query to $(P_i, ssid)$ containing a valid peer-oracle-generated $(c_1, VK)$, $D$ does not directly compute the ciphertext $c_2$. Instead, it queries Encrypt$(w)$ (where $w$ is the password given to $(P_i, ssid)$ by the environment) and sets $c_2$ equal to the value returned. If $(P_j, ssid)$ receives the unmodified ciphertext $c_2$ in its flow-two message, $D$ queries Hash$(c_2)$ and receives $(s', \eta')$. Then it sets $hp' \leftarrow s'$, and uses $\eta'$ in place of $h'_{hp'}(w, c_2; r_2)$). (Note that $\eta' = h'_{hp'}(w, c_2; r_2)$ in Expt-Hash, where $r_2$ was the randomness used in the Encrypt$(w)$ query that generated $c_2$.) Then if $(P_i, ssid)$ receives the unmodified projected key $hp'(= s')$, it also uses $\eta'$ for the appropriate portion of the session key. At the conclusion of the execution $D$ outputs whatever the environment does.

  First, if $(P_i, ssid)$ received a valid peer-oracle-generated $(c_1, VK)$ where the peer session $(P_j, ssid)$ received a non-0-oracle-generated $c_0$, then $(P_i, ssid)$ will abort in $\mathbf{H_3}$ and Mix (with all but negligible probability) due to an invalid signature. Specifically, the signature of $(P_j, ssid)$ will use a different $c_0$, and by the security of the signature scheme the adversary will not be able to forge a signature, except with negligible probability.

  Second, if $(P_i, ssid)$ receives any projected key other than $hp'$ (i.e., where $hp'$ is generated by peer session $(P_j, ssid)$ that also generated the pair $(c_1, VK)$ (received by $(P_i, ssid)$) and received an unmodified $c_2$), then $(P_i, ssid)$ will again abort in $\mathbf{H_3}$ and Mix with all but negligible probability, due to an invalid signature. Specifically, the signature of $(P_j, ssid)$ will use a different $hp'$, and by the security of the signature scheme, the adversary will not be able to forge a signature, except with negligible probability.

Now if $D$ interacts in Expt-Hash, then $D$ emulates Mix. Specifically, when $(P_i, ssid)$ receives a valid peer-oracle-generated $(c_1, VK)$ (say from peer session $(P_j, ssid)$) and does not abort, then $(P_j, ssid)$ must have received a 0-oracle-generated $c_0$, and the session key of $(P_i, ssid)$ will equal $H_{hk}(w, \ell_1, c_1) + \eta'$, where $\eta' = h'_{hp'}(w, c_2; r_2)$. Furthermore, if $(P_i, ssid)$ and $(P_j, ssid)$ have the same $c_1$ and $c_2$ in their views (even if $(P_i, ssid)$ aborts), and $(P_j, ssid)$ receives a 0-oracle-generated $c_0$, then $(P_j, ssid)$ also receives the $\eta'$ portion of the session key. Note that if $(P_i, ssid)$ does not abort and assuming the adversary does not forge a one-time signature, $(P_i, ssid)$ will have the same values of $c_0$, $c_1$, $c_2$, $hp$, and $hp'$ as $(P_j, ssid)$. Thus they will compute the same session key, and in particular, the session key would be computed as in Mix (and the real protocol).

If $D$ interacts in Expt-Unif, then $D$ emulates $\mathbf{H_3}$. Specifically, the session keys for the following instances are chosen uniformly: (1) any $(P_i, ssid)$ receiving a valid peer-oracle-generated $(c_1, VK)$ and not aborting, since it uses $\eta'$ which was generated uniformly, and (2) for each such $(P_i, ssid)$ in (1), a peer session $(P_j, ssid)$ that receives a 0-oracle-generated $c_0$, generates the pair $(c_1, VK)$ and receives $c_2$ from $(P_i, ssid)$, since $(P_j, ssid)$ also uses $\eta'$ which was generated uniformly. In (2), if $(P_i, ssid)$ does not abort, then $(P_i, ssid)$ and $(P_j, ssid)$ generate the same session key (unless the one-time signature is forged).

Thus, we conclude that $D$ distinguishes between Expt-Hash and Expt-Unif with probability negligibly close to the probability that the environment distinguishes Mix from $\mathbf{H_3}$. By [19, Corollary 3.3], this is negligible.

**Indistinguishability of $\mathbf{H_5}$ and $\mathbf{H_4}$** This is similar to the indistinguishability of $\mathbf{H_3}$ and $\mathbf{H_2}$, except that in experiment $\mathbf{H_4}$ (and thus $\mathbf{H_5}$), it is impossible for $(P_j, ssid)$ to receive a valid 2-oracle-generated ciphertext $c_2$ that was generated in a session $(P_i, ssid')$ that received a peer-oracle-generated pair $(c_1, VK)$, since in $\mathbf{H_4}$, such a session of $P_i$ would generate $c_2$ as an encryption of $\hat{w}$. Thus we only need to consider the case where $c_2$ is invalid. In this case, by the definition of smooth projective hashing, with $(P_j, ssid)$'s password $w'$, the distribution $\{pke', c_2, w', hp', H'_{hk'}(w', c_2)\}$ is statistically close to $\{pke', c_2, w', hp', g\}$, where $g \leftarrow_R G'$. Since the session key $sk$ for $(P_j, ssid)$ has $H'_{hk'}(w', c_2)$ as a component, the session key generated is statistically close to uniform and thus contributes negligibly to the distinguishability of $\mathbf{H_5}$ and $\mathbf{H_4}$.

**Indistinguishability of $\mathbf{H_8}$ and $\mathbf{H_7}$** If $(P_j, ssid)$ receives a ciphertext $c_2$ that is invalid, then by the definition of smooth projective hashing, $H'_{hk'}(w', c_2)$ is statistically close to uniform and so substituting a random session key makes at most a negligible difference. The following two cases show that if $(P_j, ssid)$ receives a ciphertext $c_2$ that is valid, then that session will abort.

**Case 1** Assume $c_2$ is 2-oracle-generated and valid for $(P_j, ssid)$. Then it must have been generated by a client session $(P_i, ssid')$ that received a non-peer-oracle-generated $(c_1, VK)$. If this pair is not generated by $(P_j, ssid)$, then the ZKP generated by $(P_i, ssid')$ will be rejected for using a different $c_1$ or $\ell_1$ value than $(P_j, ssid)$ (since our ZKPs are uniquely applicable). If $(c_1, VK)$ was generated by $(P_j, ssid)$, then $(P_j, ssid)$ is not a peer session of $(P_i, ssid')$, so the ZKP generated by $(P_i, ssid')$ will be rejected for using a different $\ell_1$ value than $(P_j, ssid)$ (again since our ZKPs are uniquely applicable). Any ZKP from any other session $(P_{i'}, ssid'')$ will be rejected because it uses a different $c_2$. Finally, by simulation-soundness, any adversarially-generated ZKP will be rejected because $c_0$ is an encryption of $\hat{w}$ whereas $c_2$ is valid (and thus is not an encryption of $\hat{w}$.) Therefore, when $c_2$ is 2-oracle-generated and valid for $P_j$, $P_j$ will reject the ZKP and abort.

**Case 2** Assume $c_2$ is non-2-oracle-generated and valid for $(P_j, ssid)$. Then a ZKP generated by any client $P_i$ will be rejected because it uses a different $c_2$ value (and our ZKPs are uniquely applicable). Also, by simulation-soundness, any adversarially-generated ZKP will be rejected because $c_0$ is an encryption of $\hat{w}$ whereas $c_2$ is valid (and thus is not an encryption of $\hat{w}$.) Therefore $P_j$ will reject the ZKP and abort.

## 6 Password-Based Secure Channels

Arguably, the typical use of a key-exchange protocol is the establishment of *secure channels* that enable the parties to communicate privately and reliably. The case of password-based key-exchange is no exception. In this section, we show that our definition of password-based key-exchange suffices for obtaining secure channels (of course, with the limitation that secure channels are *not* obtained if the adversary succeeds in guessing the secret password).

We begin by presenting a password-based secure channels functionality $\mathcal{F}_{\mathsf{pwSC}}$ in Figure 6. This definition is analogous to the password-based key exchange functionality as presented in Section 2. One key difference in the addition of a ready state, whose purpose is to signify that the parties have finished *establishing* the channel and are now ready to *use* it. Technically, we introduce this state in order to preclude protocols that allow the adversary to make password guesses *after* the parties have begun communicating using the secure channel they have established.

Notice that if two parties have sent NewSession queries with the same identifiers and passwords, and the adversary has not guessed this password or interrupted the session, then the functionality faithfully passes messages from one party to the other. Furthermore, the adversary learns only the length of the message sent and is unable to send any messages of his own. Thus, the functionality provides reliable and private communication, as desired.

**Realizing the $\mathcal{F}_{\mathsf{pwSC}}$ functionality.** Given a secure password-based key-exchange protocol, realizing the password-based secure channels functionality $\mathcal{F}_{\mathsf{pwSC}}$ is quite straightforward: simply use the key-exchange protocol to generate a shared secret key, and then use the shared secret key to protect the traffic using standard encryption and message authentication (along with some shared counter to protect against replay attacks).

Formally, in Figure 7, we present a protocol that securely realizes $\mathcal{F}_{\mathsf{pwSC}}$ in the $\mathcal{F}_{\mathsf{pwKE}}$-hybrid model. The protocol uses the key generated by $\mathcal{F}_{\mathsf{pwKE}}$ in order to apply symmetric encryption and secure message authentication to messages that parties wish to send to each other. In the figure, MAC designates a secure message authentication algorithm, and $(E, D)$ designates a symmetric encryption scheme that is semantically secure against chosen plaintext attacks.

**Theorem 2** *If* MAC *is a secure message authentication algorithm and $(E, D)$ is a semantically secure symmetric encryption scheme, each with key-length of $k/2$ bits, then protocol SC from Figure 7 securely realizes functionality $\mathcal{F}_{\mathsf{pwSC}}$ in the $\mathcal{F}_{\mathsf{pwKE}}$-hybrid model, against static-corruption adversaries.*

**Proof (sketch):** We need to exhibit a simulator $S$ in the $\mathcal{F}_{\mathsf{pwSC}}$ model for every hybrid-world adversary $A$ in the $\mathcal{F}_{\mathsf{pwKE}}$-hybrid model. The only interesting case is when neither $P_i$ nor $P_j$ are corrupted (since otherwise $S$ knows all the "secrets" and can perfectly simulate everything that the non-corrupted player does). In this case, the simulator proceeds as follows.

- Upon receiving a message $(\mathsf{NewSession}, sid, P_i, P_j, \text{role})$ from $\mathcal{F}_{\mathsf{pwSC}}$, $S$ records this session (and the fact that it is still fresh) and forward the message to $A$ as if coming from $\mathcal{F}_{\mathsf{pwKE}}$.

**Functionality $\mathcal{F}_{\mathsf{pwSC}}$**

The functionality $\mathcal{F}_{\mathsf{pwSC}}$ is parameterized by a security parameter $k$, and maintains a list of "messages in transit" which is initially empty and a counter that is initialized to zero. It interacts with an adversary $S$ and a set of parties via the following queries:

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

Send $(\mathsf{NewSession}, sid, P_i, P_j, \text{role})$ to $S$. In addition, if this is the first $\mathsf{NewSession}$ query, or if this is the second $\mathsf{NewSession}$ query and there is a record $(P_j, P_i, pw')$, then record $(P_i, P_j, pw)$ and mark this record $\mathsf{fresh}$.

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

If there is a record of the form $(P_i, P_j, pw)$ which is $\mathsf{fresh}$, then do: If $pw = pw'$, mark the record $\mathsf{compromised}$ and reply to $S$ with "correct guess". If $pw \neq pw'$, mark the record $\mathsf{interrupted}$ and reply with "wrong guess".

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

If there is a record of the form $(P_i, P_j, pw)$ which is $\mathsf{fresh}$, and if neither $P_i$ nor $P_j$ are corrupted, and if there is a record $(P_j, P_i, pw')$ with $pw' = pw$ that is either $\mathsf{fresh}$ or $\mathsf{ready}$, then mark the record $(P_i, P_j, pw)$ as $\mathsf{ready}$ and reply to $S$ with "ready". Otherwise reply with "not ready".

**Upon receiving a query** $(\mathsf{Send}, sid, m)$ **from** $P_i$**:**

If there is a record of the form $(P_i, P_j, pw)$ then:

- If this record is $\mathsf{compromised}$ or $P_i$ is corrupted, then send $(sid, \mathsf{Send}, P_i, m)$ to $S$.
- If this record is $\mathsf{ready}$ then send $(sid, \mathsf{Send}, P_i, |m|, t)$ to $S$ where $t$ is the current counter value, record the pair $(m, t)$ in the list of messages and increment the counter.
- In any other case, ignore this query.

**Upon receiving a query** $(\mathsf{Deliver}, sid, P_i, P_j, m, t)$ **from the adversary** $S$**:**

If $P_j$ is corrupted or if there is a record of the form $(P_j, P_i, pw)$ which is $\mathsf{compromised}$, then send $(\mathsf{Received}, sid, m)$ to $P_j$.

Otherwise, if there is a record of the form $(P_j, P_i, pw)$ which is $\mathsf{ready}$ and the list of messages contains a pair $(m', t)$ (with the same counter value $t$ as in the query), then send $(\mathsf{Received}, sid, m')$ to $P_j$ and remove the pair $(m', t)$ from the list.

Figure 6: The password-based secure channels functionality $\mathcal{F}_{\mathsf{pwSC}}$

**Password-based secure channels protocol** $SC$

**Protocol steps:**

1. When $P_i$ is activated with input ($\mathsf{NewSession}, sid, P_i, P_j, pw, \text{role}$), it forwards it to $\mathcal{F}_{\mathsf{pwKE}}$. Upon receiving a key $K$ from $\mathcal{F}_{\mathsf{pwKE}}$, it partitions the key into two keys $K_E$ and $K_A$ to be used for encryption and authentication, respectively.
2. When $P_i$ is activated with input ($\mathsf{Send}, sid, m$) after receiving a key $K$ from $\mathcal{F}_{\mathsf{pwKE}}$, it computes $c = E_{K_E}(m)$ and $a = \mathsf{MAC}_{K_A}(c, i, l)$, where $l$ is a counter that is initialized to zero and incremented for each message sent, and sends $(sid, c, l, a)$ to $P_j$. (If $P_i$ has not obtained a key $K$ then it ignores this request.)
3. Upon receiving a message $(sid, c, l, a)$ after obtaining a key $K$ from $\mathcal{F}_{\mathsf{pwKE}}$, party $P_i$ first verifies that $a$ is a valid tag for $(c, j, l)$ with respect to key $K_a$ and that no message with counter value $l$ was previously received in this session. If so, it computes $m = D_{K_e}(c)$ and outputs ($\mathsf{Received}, sid, m$). (If $P_i$ does not yet have a key $K$ then it ignores this request.)
4. When $P_i$ is activated with input ($\mathsf{ExpireSession}, sid$), it erases all the session state (local keys and randomness). All future messages are ignored.

Figure 7: Realizing secure channels using key-exchange

- If $A$ makes a query ($\mathsf{TestPwd}, sid, P_i, pw'$) aimed at $\mathcal{F}_{\mathsf{pwKE}}$ then $S$ makes the same query to $\mathcal{F}_{\mathsf{pwSC}}$ and reports the reply back to $A$. Note that the reply from $\mathcal{F}_{\mathsf{pwSC}}$ is sufficient for $S$ to determine the status of the session (i.e., $\mathsf{compromised}$ or $\mathsf{interrupted}$).
- When $A$ makes a query ($\mathsf{NewKey}, sid, P_i, sk$) aimed at the functionality $\mathcal{F}_{\mathsf{pwKE}}$, then $S$ makes the query ($\mathsf{Ready}$,$sid$,$P_i$) to $\mathcal{F}_{\mathsf{pwSC}}$. If the session $(sid, P_i, P_j)$ is $\mathsf{compromised}$ then $S$ records the secret key $sk$ for that session.

  Else, if $\mathcal{F}_{\mathsf{pwKE}}$ replies with "ready" and $S$ already recorded a secret key $sk'$ for a session $(sid, P_j, P_i)$, then $S$ records the same secret key $sk'$ also for the session $(sid, P_i, P_j)$.

  Else, $S$ chooses a new random $k$-bit string $sk''$ and records it for the session $(sid, P_i, P_j)$.
- When $S$ gets a message ($\mathsf{Send}, sid, P_i, m$) from $\mathcal{F}_{\mathsf{pwSC}}$ (i.e., when $P_i$ is corrupted or the session is $\mathsf{compromised}$) it does the following: If $S$ has a recorded key $K$ for $(sid, P_i, P_j)$ (for some $P_j$) then it partitions the key into two segments, treated as two keys $K_E$ and $K_A$, computes $c = E_{K_E}(m)$ and $a = \mathsf{MAC}_{K_A}(c, i, l)$, where $l$ is a counter that is initialized to zero and incremented for each message sent, and sends $(sid, c, l, a)$ to $A$ on behalf of $P_i$.

  Upon receiving a message ($\mathsf{Send}, sid, P_i, |m|, t$) from $\mathcal{F}_{\mathsf{pwSC}}$, $S$ behaves similarly except that it computes $c = E_{K_E}(0^{|m|})$. (I.e., it encrypts the all-zero message of the appropriate length instead of the "real message" that it doesn't know.) Also, in this case $S$ records the tuple $(sid, c, l, a, t)$ in a list of "fake ciphertexts".
- When $S$ receives a message $(sid, c, l, a)$ from $A$ to $P_i$ on behalf of $P_j$, $S$ does the following:

  If $S$ finds a tuple $(sid, c, l, a, t)$ in the list of "fake ciphertexts" (for the same $(sid, c, l, a)$ and some $t$), then it makes a query ($\mathsf{Receive}, sid, P_i, P_j, \perp, t$) to $\mathcal{F}_{\mathsf{pwSC}}$ and removes that record from the fake-ciphertext list.

  Otherwise, $S$ looks up the session $(sid, P_i, P_j)$. If it has a key $K$ for this session then it partitions the key into two segments, treated as two keys $K_E$ and $K_A$, and checks that $a = \mathsf{MAC}_{K_A}(c, j, l)$ and that no message with counter value $l$ was previously received in this session. If so, $S$ computes $m = D_{K_e}(c)$ and makes a query $(\mathsf{Receive}, sid, P_i, P_j, m, \perp)$ to $\mathcal{F}_{\mathsf{pwSC}}$.

It is fairly straightforward to prove indistinguishability between the $\mathcal{F}_{\mathsf{pwSC}}$ world with $S$ and the $\mathcal{F}_{\mathsf{pwKE}}$-hybrid world with $A$ and $SC$. The only differences in the views of the environment in the two worlds are:

- In the $\mathcal{F}_{\mathsf{pwKE}}$-hybrid world with $A$ and $SC$ the environment always sees encryptions of the "right" message $m$ when it makes a query $(\mathsf{Send}, sid, P_i, m)$, whereas in the $\mathcal{F}_{\mathsf{pwSC}}$ world with $S$ it sometimes sees an encryption of $0^{|m|}$ instead.

- In the $\mathcal{F}_{\mathsf{pwKE}}$-hybrid world with $A$ and $SC$, a received message that was not sent by the "honest peer" but still passes the MAC verification will always be decrypted and returned to the environment. In the $\mathcal{F}_{\mathsf{pwSC}}$ world with $S$, on the other hand, it will only be returned to the environment if the receiver is corrupted or the session is compromised (cf. the description of the $(\mathsf{Deliver}, \ldots)$ query in $\mathcal{F}_{\mathsf{pwSC}}$).

It can be shown that the second difference is only present if the transcript of the interaction in the $\mathcal{F}_{\mathsf{pwKE}}$-hybrid world includes MAC forgeries, and therefore only occurs with negligible probability assuming MAC is a secure message authentication algorithm. Also, it is easy to reduce the first difference to the security of the encryption scheme. □

**Some variations.** There are some standard variations that can be made to the functionality $\mathcal{F}_{\mathsf{pwSC}}$ above. In particular, to capture forward-secresy we could add an expiry interface, namely a query $(\mathsf{ExpireSession}, sid)$ that player $P_i$ can send. (This will change the state of the recorded session $(P_i, P_j, pw)$ from ready to interrupted.) Also, we could augement $\mathcal{F}_{\mathsf{pwSC}}$ to ensure ordered delivery of message by replacing the list of messages with a FIFO queue. Additionally, we could allow "labeled messages" $(\ell, m)$ instead of the simple messages $m$, where on query $(\mathsf{Send}, sid, \ell, m)$ the adversary $S$ is shown $\ell$ and $|m|$.

# 7 Impossibility of Realizing $\mathcal{F}_{\mathsf{pwKE}}$ in the Plain Model

In this section, we show that the $\mathcal{F}_{\mathsf{pwKE}}$ functionality cannot be securely realized in the "plain" model (e.g., without using a common random string or some other augmentation to the basic model). Our proof follows very similar lines to the proofs of impossibility in [11]. The basic idea is as follows. Consider the environment $\mathcal{Z}$ that internally runs the code of one of the honest parties. The ideal-model simulator $\mathcal{S}$ for $\mathcal{Z}$ must succeed while interacting with $\mathcal{Z}$ in a "straight-line" manner (i.e., with black-box access to $\mathcal{Z}$ and without rewinding, as required by the UC framework). Now, if simulation can be carried out in such a scenario, then the (different) environment $\mathcal{Z}'$ which internally runs $\mathcal{S}$ can carry out such a simulation while interacting with a real honest party. In other words, anything the ideal-model simulator $\mathcal{S}$ can do with respect to $\mathcal{Z}$, the second environment $\mathcal{Z}'$ can do with respect to an honest party in the real world. In particular, in order for $\mathcal{S}$'s simulation of $\mathcal{Z}$ to succeed, $\mathcal{S}$ must be able to set the session key output by $\mathcal{Z}$ to be equal to the session key output by the ideal functionality. But then $\mathcal{Z}'$ can also do this, in contradiction to the security requirements of a key-exchange protocol.

The impossibility result below refers to *non-trivial* protocols. A protocol is non-trivial if two honest parties are ensured to agree on matching session keys at the conclusion of a protocol execution (except perhaps with negligible probability), provided that (1) both parties use the same password, and (2) the adversary passes all messages between the parties without modifying them or inserting any messages of its own. The non-triviality requirement is needed since the "empty" protocol where parties do nothing securely realizes $\mathcal{F}_{\mathsf{pwKE}}$ (the ideal-model simulator simply never issues a $\mathsf{NewKey}$ query to the functionality and so the parties never actually obtain keys). The notion of non-trivial protocols has been used before for similar reasons; see [11] for example.

**Theorem 3** *There does not exist a non-trivial protocol* $\Pi$ *that securely realizes the* $\mathcal{F}_{\mathsf{pwKE}}$ *functionality in the plain model.*

**Proof:** Assume for the sake of contradiction that there exists a non-trivial protocol $\Pi$ that securely realizes $\mathcal{F}_{\mathsf{pwKE}}$ in the plain model. Consider an environment $\mathcal{Z}$ who plays the role of $P_2$ and runs an execution of the protocol with $P_1$ and the "canonical real-world adversary" $\mathcal{A}$ who corrupts no parties (recall that such an $\mathcal{A}$ simply forwards messages between $\mathcal{Z}$ and the players as instructed).

The environment $\mathcal{Z}$ comes equipped with some arbitrary dictionary $\mathcal{D}$ of size at least 2, from which it chooses passwords. $\mathcal{Z}$ chooses a random password $pw \in_R \mathcal{D}$ and provides $(\mathsf{NewSession}, sid, P_1, P_2, pw, \mathsf{client})$ as input to $P_1$. Next, $\mathcal{Z}$ *internally* runs protocol $\Pi$, honestly following the instructions that $P_2$ would follow upon receiving input $(\mathsf{NewSession}, sid, P_2, P_1, pw, \mathsf{server})$. Furthermore, $\mathcal{Z}$ instructs $\mathcal{A}$ to forward all messages between itself and $P_1$. When the protocol terminates, $\mathcal{Z}$ compares the session-key $sk_2$ that it obtained by playing $P_2$ and compares it to the session-key $sk_1$ that $P_1$ writes to its output tape. If $sk_1 = sk_2 \neq \perp$ and neither $P_1$ or $P_2$ have been corrupted, then $\mathcal{Z}$ outputs 1.[13] Otherwise, $\mathcal{Z}$ outputs 0. Now, recall that $\mathcal{A}$ is the canonical (dummy) real-life adversary. Therefore, it just forwards messages between $\mathcal{Z}$ and $P_1$ without doing anything else. Since $\mathcal{Z}$ plays the honest $P_2$ and party $P_1$ is not corrupted, it follows that this execution is the same as one between two honest parties that use the same, randomly chosen password. (A crucial point in this argument is the fact that protocol $\Pi$ runs in the *unauthenticated channels* model. Therefore, $P_1$ cannot distinguish messages sent by the real (honest) $P_2$ and messages sent by $\mathcal{Z}$ in the name of $P_2$.) By the assumption that $\Pi$ is non-trivial, we have that in a real execution $\mathcal{Z}$ outputs 1 except with negligible probability. That is,

$$\Pr[\ sk_2 = sk_1 \neq \perp \mid \mathcal{Z} \text{ in real world }\ ] \ \geq\ 1 - \mathsf{negl}(k).$$

Next, consider the ideal-world simulator $\mathcal{S}$ that is guaranteed to exist by the security of $\Pi$. Simulator $\mathcal{S}$ interacts with the $\mathcal{F}_{\mathsf{pwKE}}$ functionality and with $\mathcal{Z}$ (who follows the same strategy as above). We claim that except with negligible probability, it holds that $sk_2 = sk_1 \neq \perp$ in this execution; otherwise, the probability that $\mathcal{Z}$ outputs 1 in the ideal execution is non-negligibly far from the probability that it outputs 1 in the real execution, in contradiction to the security of $\Pi$. That is,

$$\Pr[\ sk_2 = sk_1 \neq \perp \mid \mathcal{Z} \text{ in ideal world }\ ] \ \geq\ 1 - \mathsf{negl}(k).$$

Intuitively, it is difficult for $\mathcal{S}$ to provide a "good" simulation for $\mathcal{Z}$ since $\mathcal{S}$ must somehow send the correct password $pw$ to $\mathcal{F}_{\mathsf{pwKE}}$, while its only way of obtaining information about $pw$ is through a "real" execution of the protocol with $\mathcal{Z}$ (who plays the honest $P_2$). See Figure 8 for a diagram describing the flow of the proof; so far, we have seen cases (a) and (b).

**A different scenario.** Until now, we have considered an environment $\mathcal{Z}$ who internally plays the role of $P_2$, while $P_1$ is played by the (real) honest party. We now switch sides and consider an environment $\mathcal{Z}'$ who internally plays the role of $P_1$ (albeit in a strange way), while $P_2$ is played by the

[13]For simplicity, we denote the output of a party who failed to successfully conclude the protocol by $\perp$. In the definition of $\mathcal{F}_{\mathsf{pwKE}}$, in such a case no output would be generated. This makes no difference.

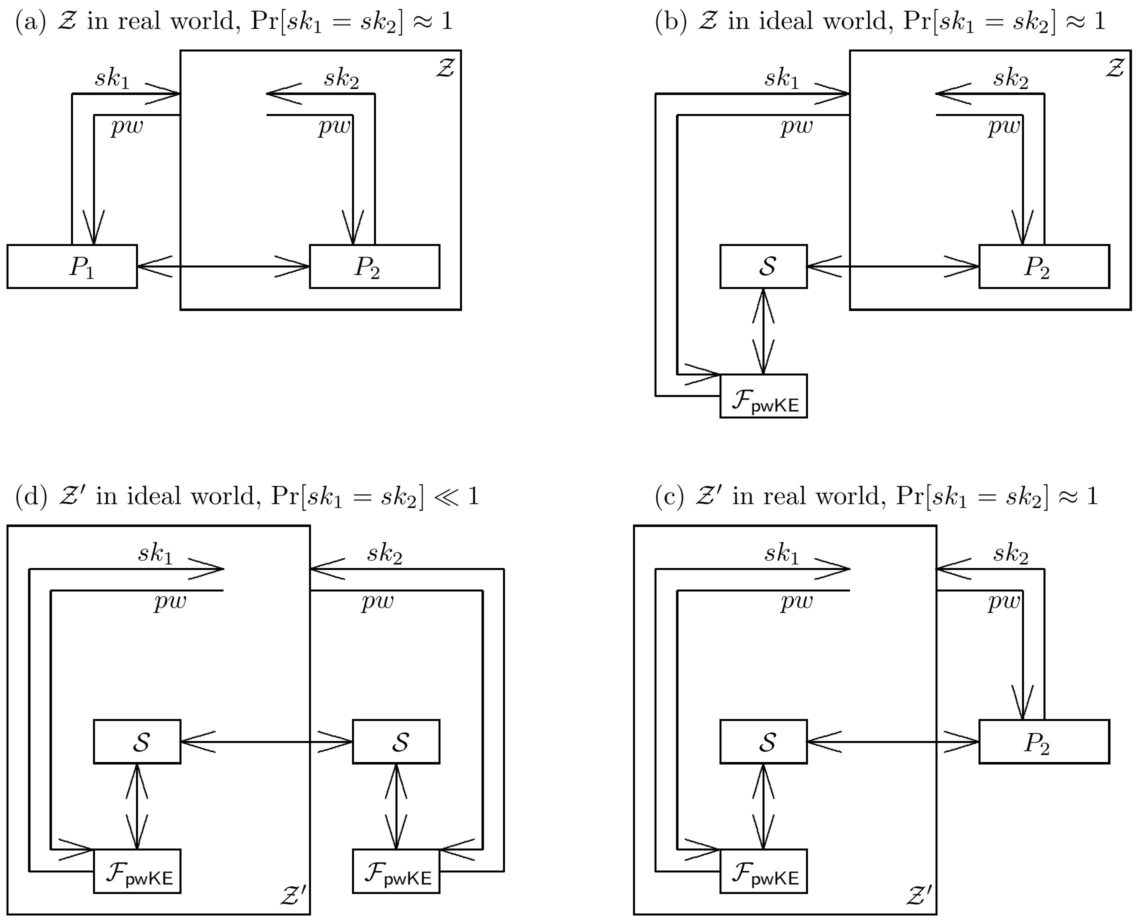


Figure 8: The four cases in the proof of Theorem 3.

real honest party. As above, in the real world $\mathcal{Z}'$ runs together with a dummy adversary $\mathcal{A}'$ who corrupts no parties and forwards all messages as instructed. The aim of $\mathcal{Z}'$ in its behavior is to emulate the ideal execution of $\mathcal{Z}$ with $\mathcal{S}$, while interacting with the external (real) $P_2$. Such an emulation is conceivable because in the ideal world from above, the simulator $\mathcal{S}$ essentially interacts with the honest $P_2$ (this is the case because $\mathcal{Z}$ plays the honest $P_2$, and $\mathcal{S}$ interacts with $\mathcal{Z}$); see Figure 8. Specifically, $\mathcal{Z}'$ internally plays the simulator $\mathcal{S}$ from above, a copy of the functionality $\mathcal{F}_{\mathsf{pwKE}}$, and the honest ideal-model party $P_1$. That is, $\mathcal{Z}'$ uses the same dictionary $\mathcal{D}$ that $\mathcal{Z}$ used, it chooses a random password $pw \in_R \mathcal{D}$ and internally hands the message $(\mathsf{NewSession}, sid, P_1, P_2, pw, \mathsf{client})$ to its internal ideal-model $P_1$ (which will simply forward this to the internal simulation of $\mathcal{F}_{\mathsf{pwKE}}$). Then, $\mathcal{Z}'$ internally invokes $\mathcal{S}$ and a copy of the functionality $\mathcal{F}_{\mathsf{pwKE}}$, forwarding all appropriate messages between them and between the internal (ideal) $P_1$. Finally, $\mathcal{Z}'$ hands the external, honest real-model party $P_2$ the input $(\mathsf{NewSession}, sid, P_2, P_1, pw, \mathsf{server})$ and forwards to it all the messages that $\mathcal{S}$ wishes to send to $\mathcal{Z}$; likewise, messages received from $P_2$ are forwarded back to $\mathcal{S}$ as if they were received from $\mathcal{Z}$. (This communication between $\mathcal{Z}'$ and $P_2$ is carried out through the canonical real-model adversary $\mathcal{A}'$ who, as in the previous scenario, has corrupted no parties.) $\mathcal{Z}'$ continues until $P_2$ outputs a session key $sk_2$ and the internal copy of $\mathcal{F}_{\mathsf{pwKE}}$ sends a key $sk_1$ to the internal copy of $P_1$. Once this happens, $\mathcal{Z}'$ outputs 1 if and only if the above keys fulfill the condition that $sk_1 = sk_2 \neq \perp$, and neither $P_1$ nor $P_2$ were corrupted. The main observation of this proof is that the real execution of $\mathcal{Z}'$ with $\mathcal{A}'$ and the real honest $P_2$ is *identical* to the ideal execution of $\mathcal{Z}$ above with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{pwKE}}$. It therefore holds that

$$\Pr[sk_2 = sk_1 \neq \perp \mid \mathcal{Z}' \text{ in real world}] \;=\; \Pr[sk_2 = sk_1 \neq \perp \mid \mathcal{Z} \text{ in ideal world}] \;\geq\; 1 - \mathsf{negl}(k)$$

and so by the definition of $\mathcal{Z}'$, it outputs 1 in this real execution except with negligible probability (recall that the real adversary $\mathcal{A}'$ never corrupts $P_1$ or $P_2$).

We now reach the last scenario, in which $\mathcal{Z}'$ runs in the ideal world. Notice that in this scenario, $\mathcal{Z}'$ runs an internal copy of $\mathcal{S}$ (by its above described behavior), while interacting with an external copy of $\mathcal{S}$ (because this is an ideal-world execution); we denote these respective copies by $\mathcal{S}_{\text{int}}$ and $\mathcal{S}_{\text{ext}}$. (Similarly, $\mathcal{Z}'$ runs an internal copy of $\mathcal{F}_{\mathsf{pwKE}}$, while $\mathcal{S}_{\text{ext}}$ and the honest $P_2$ interact with an external copy of $\mathcal{F}_{\mathsf{pwKE}}$.) We claim that $\mathcal{Z}'$ outputs 1 in this scenario with probability that is non-negligibly less than 1. In order to see this, notice that when $\mathcal{Z}'$ runs in the ideal world, all of the messages that the "real simulator" $\mathcal{S}_{\text{ext}}$ sees are generated by the "internal simulator" $\mathcal{S}_{\text{int}}$ that is run by $\mathcal{Z}'$. Furthermore, neither copy of $\mathcal{S}$ is given the password $pw$ that $\mathcal{Z}'$ hands to the internal $P_1$ and the external (ideal-world) honest $P_2$. Therefore, as long as neither simulator makes a password guess to its copy of $\mathcal{F}_{\mathsf{pwKE}}$, their execution is independent of the password $pw$. Now, $\mathcal{Z}'$ outputs 1 if and only if the session-key $sk_1$ that is generated by the internal copy of $\mathcal{F}_{\mathsf{pwKE}}$ for the internal $P_1$ equals the session-key $sk_2$ that is generated by the external copy of $\mathcal{F}_{\mathsf{pwKE}}$ for the external $P_2$. Since neither $P_1$ nor $P_2$ are corrupted, it follows that the copies of $\mathcal{S}$ can influence the session-key if and only if they both correctly guess the value of $pw$ (this is the only way that $\mathcal{F}_{\mathsf{pwKE}}$ will not choose a random $sk \in_R \{0,1\}^k$). From the definition of $\mathcal{Z}'$, we have that $pw \in_R \mathcal{D}$ and so the correct password can be guessed with probability at most $1/|\mathcal{D}|$. That is,

$$\Pr[sk_2 = sk_1 \neq \perp \mid \mathcal{Z}' \text{ in ideal world}] \leq \frac{1}{|\mathcal{D}|} + 2^{-k}$$

and so $\mathcal{Z}'$ outputs 1 in this scenario with probability at most $1/|\mathcal{D}| + 2^{-k}$. Since $\mathcal{D}$ is of size at least 2, it follows that $\mathcal{Z}'$ outputs 1 in the ideal world with probability that is non-negligibly less than the probability that it outputs 1 in the real world. This contradicts the assumed security of $\Pi$ and concludes the proof.

We remark that the proof holds as long as $\mathcal{Z}$ chooses passwords according to any distribution that does not place almost all of the support on a single point. Specifically, all that is required is that a password guess by $\mathcal{S}$ should fail to be correct (i.e., equal the password chosen by $\mathcal{Z}'$) with non-negligible probability. ■

## References


[1] B. Barak, Y. Lindell, and T. Rabin, Protocol Initialization for the Framework of Universal Composability. Manuscript. Available from the ePrint archive, report 2004/006 from `http://eprint.iacr.org`.

[2] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. *Advances in Cryptology – Crypto 1998*, LNCS vol. 1462, Springer-Verlag, pp. 26–45, 1998

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. *Advances in Cryptology – Eurocrypt 2000*, LNCS vol. 1807, Springer-Verlag, pp. 139–155, 2000.

[4] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *Advances in Cryptology – Crypto 1993*, LNCS vol. 773, Springer-Verlag, pp. 232–249, 1993.

[5] S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. Proc. *IEEE Security and Privacy*, IEEE, pp. 72–84, 1992.

[6] M. Boyarsky. Public-Key Cryptography and Password Protocols: The Multi-User Case. *7th Annual Conference on Computer and Communications Security*, ACM, pp. 63–72, 1999.

[7] V. Boyko. On All-or-Nothing Transforms and Password-Authenticated Key Exchange. PhD thesis, MIT, EECS department, 2000.

[8] V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password Authentication and Key Exchange Using Diffie-Hellman. *Advances in Cryptology – Eurocrypt 2000*, LNCS vol. 1807, Springer-Verlag, pp. 156–171, 2000.

[9] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. *42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE, pp. 136–145, 2001.

[10] R. Canetti and H. Krawczyk. Universally Composable Notions of Key Exchange and Secure Channels. *Advances in Cryptology – Eurocrypt 2002*, LNCS vol. 2332, Springer-Verlag, pp. 337–351, 2002.

[11] R. Canetti, E. Kushilevitz and Y. Lindell. On the Limitations of Universal Composable Two-Party Computation Without Set-Up Assumptions. In *EUROCRYPT 2003,* Springer-Verlag (LNCS 2656), pages 68–86, 2003.

[12] R. Canetti and T. Rabin. Universal Composition with Joint State. *Advances in Cryptology – Crypto 2003*, LNCS vol. 2729, Springer-Verlag, pp. 265–281, 2003.

[13] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. *Advances in Cryptology – Crypto 1994*, LNCS vol. 839, Springer-Verlag, pp. 174–187, 1994.

[14] R. Cramer and V. Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. *Advances in Cryptology – Eurocrypt 2002*, LNCS vol. 2332, Springer-Verlag, pp. 45–64, 2002.

[15] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-Interactive Zero-Knowledge. *Advances in Cryptology – Crypto 2001*, LNCS vol. 2139, Springer-Verlag, pp. 566–598, 2001.

[16] D. Dolev, C. Dwork, and M. Naor. Non-Malleable Cryptography. *SIAM J. Computing* 30(2): 391–437, 2000.

[17] S. Even, O. Goldreich, and S. Micali. On-Line/Off-Line Digital Signatures. *J. Cryptology* 9(1):35-67, 1996.

[18] J. Garay, P. MacKenzie, and K. Yang. Strengthening Zero-Knowledge Protocols Using Signatures. *Advances in Cryptology – Eurocrypt 2003*, LNCS vol. 2656, Springer-Verlag, pp. 177–194, 2003.

[19] R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. *Advances in Cryptology – Eurocrypt 2003*, LNCS vol. 2656, Springer-Verlag, pp. 524–543, 2003.

[20] O. Goldreich and Y. Lindell. Session-Key Generation using Human Passwords Only. *Advances in Cryptology – Crypto 2001*, LNCS vol. 2139, Springer-Verlag, pp. 408–432, 2001.

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

[22] L. Gong, M. Lomas, R. Needham, and J. Saltzer. Protecting Poorly Chosen Secrets from Guessing Attacks. *IEEE Journal on Selected Areas in Communications*, 11(5): 648–656, 1993.

[23] Shai Halevi and Hugo Krawczyk. Public-Key Cryptography and Password Protocols. *ACM Trans. on Information and Systems Security,* 2(3):230–268, 1999.

[24] J. Katz, P. MacKenzie, G. Taban, and V. Gligor. Two-Server Password-Only Authenticated Key Exchange. *Applied Cryptography and Network Security (ACNS) 2005*, LNCS vol. 3531, Springer-Verlag, pp. 1–16, 2005.

[25] J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. *Advances in Cryptology – Eurocrypt 2001*, LNCS vol. 2045, Springer-Verlag, pp. 475–494, 2001.

[26] P. MacKenzie, S. Patel, and R. Swaminathan. Password-Authenticated Key Exchange Based on RSA. *Adv. in Cryptology — Asiacrypt 2000*, LNCS vol. 1976, Springer-Verlag, pp. 599–613, 2000.

[27] P. MacKenzie and K. Yang. On Simulation-Sound Trapdoor Commitments. *Advances in Cryptology – Eurocrypt 2004*, LNCS vol. 3027, Springer-Verlag, pp. 382–400, 2004. Available from the ePrint archive, report 2003/252 from `http://eprint.iacr.org`.

[28] M.H. Nguyen and S. Vadhan. Simpler Session-Key Generation from Short Random Passwords. Proceedings of the *1st Theory of Cryptography Conference* (TCC'04), LNCS vol. 2951, Springer-Verlag, pp. 442–445, 2004.

[29] C. Rackoff and D. Simon. Non-Interactive Zero-Knowledge Proof of Knowledge and Chosen Ciphertext Attack. *Advances in Cryptology – Crypto 1991*, LNCS vol. 576, Springer-Verlag, pp. 433–444, 1991.

[30] A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security. *40th IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE, pp. 543–553, 1999.

[31] V. Shoup. A Proposal for an ISO Standard for Public Key Encryption (version 2.1). December, 2001. Available from the ePrint archive, report 2001/112 from `http://eprint.iacr.org`.

## A Relation to Prior Notions of Security

Perhaps the best justification for our definition of the ideal functionality in Section 2 is that the notion suffices to obtain password-based secure channels, as discussed in Section 6. When introducing a new definition, however, it is always useful to explore how the definition compares to prior definitions (as a "sanity check", if nothing else). We show here that our definition ensures some security properties one would naturally expect, and in fact that any protocol secure with respect to our definition is also secure with respect to the commonly used definition of Bellare, et al. [3]. (Due to technical differences between the definitions regarding the use of session identifiers, the implication actually only holds when the protocol that is secure under our definition is modified in a trivial way so that it doesn't assume externally-provided session-identifiers; see Section 5.1.)

In this section we consider two-party protocols for password-based key exchange with the following properties: **(1)** the protocol always terminates in some (fixed) polynomial number of rounds; and **(2)** if two parties execute the protocol honestly using matching passwords, and without any interference from the adversary, then the parties obtain matching (non-null) session keys with probability one.

**Security against eavesdroppers.** We begin with a technical lemma which may be interesting in its own right. Let $\Pi$ be a protocol that securely realizes $\hat{\mathcal{F}}_{\mathsf{pwKE}}$ (i.e., the multi-session extension[14] of $\mathcal{F}_{\mathsf{pwKE}}$), and let $S$ be a simulator for the "dummy adversary" who interacts with the parties executing $\Pi$ while exactly following the instructions of the environment. We show that if the environment instructs the adversary to only eavesdrop on a particular session, then $S$ will *not* make a $\mathsf{TestPwd}$ query for this session. This result indicates that any protocol that securely realizes $\mathcal{F}_{\mathsf{pwKE}}$ is secure against an eavesdropping adversary, even if this adversary knows the password being used.

Formally, consider the "canonical" real-world adversary interacting with players executing $\Pi$ and fix any simulator $S$ for this adversary. For any environment $\mathcal{Z}$, define $\mathsf{SpuriousGuess}(S, \mathcal{Z}; k)$ to be the event that $\mathcal{Z}$ initiates a session between two players, instructs the adversary/simulator to pass messages between these players unmodified (i.e., to only eavesdrop on this session), and yet $S$ makes a $\mathsf{TestPwd}$ query to $\hat{\mathcal{F}}_{\mathsf{pwKE}}$ for this session.

**Lemma A.1** *If a protocol $\Pi$ securely realizes $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$, then for every environment $\mathcal{Z}$ it holds that* $\Pr[\mathsf{SpuriousGuess}(S, \mathcal{Z}; k)] = \mathsf{negl}(k)$.

**Proof (sketch):** On a high level, this is because the simulator $S$ cannot have any information on the passwords that $\mathcal{Z}$ actually gives to the parties. Thus, even if $S$ "believes" it knows the password that will be used, $\mathcal{Z}$ can always "switch" the password and "catch" $S$ making an inappropriate $\mathsf{TestPwd}$ query. Details follow.

Consider an environment $\mathcal{Z}'$ that behaves just as $\mathcal{Z}$, except that it chooses at random one session during the run and selects the password $pw$ for both players in that session at random from $\{0,1\}^k$. If that session happens to be the one in which event $\mathsf{SpuriousGuess}$ occurs, then in the real world (i.e., when interacting with the canonical adversary who follows instructions) $\mathcal{Z}'$ observes that the players in that session compute matching session keys with probability 1. However, in the simulated world (i.e., when interacting with $S$), matching session keys result only with negligible probability (matching keys will occur only if $S$ correctly guesses $pw$, or if it makes an incorrect guess but the independently-generated session keys happen to be equal). The environment can therefore distinguish the real and ideal executions with probability that is negligibly close to the value $\Pr[\mathsf{SpuriousGuess}(S, \mathcal{Z}; k)]$. The lemma follows from the security of $\Pi$. $\Box$

**The definition of [3].** We now show that any protocol that securely realizes $\hat{\mathcal{F}}_{\mathsf{pwKE}}$ is secure also with respect to the commonly-used definition of Bellare, et al. [3] (modulo technicalities relating to

[14]Of course, Lemma A.1 is true also for $\mathcal{F}_{\mathsf{pwKE}}$ (and the proof is even a bit simpler). But we need the multi-session extension for our proof of Proposition A.2.

session identifiers). Recall that the definition in [3] adapts the definition of security used for "basic" key exchange [4] to the password-based setting. In particular, the adversary interacts with "oracles" that represent instances of players. All the communication in the network, as well as all corruptions and key exposures, are done via oracle queries of the adversary. In addition, the adversary has a special "test" query: in that query the adversary specifies one "fresh" instance, and is given either the actual key for that instance or a random key. In the definition of [3], a password-based protocol is secure if whenever the passwords are chosen uniformly at random from a dictionary $\mathcal{D}$ and the adversary "actively interacts" with at most $q$ user instances,[15] the advantage of the adversary in distinguishing the real key from a random key is at most $q/|\mathcal{D}|$ (plus some negligible quantity).

To show that our notion of security implies the notion of [3], we first need to reconcile a small discrepancy between the communication models used by the two definitions. As discussed in Section 5.1, the UC framework assumes that each communicating party obtains a session-identifier (SID) as an input to the protocol (and in particular, in advance of protocol execution), and that different sessions always have different SIDs. The model of [3], on the other hand, does not assume that SIDs are agreed upon in advance of protocol execution; instead, in their model the SID of an instance is defined after execution of the protocol (based on the resulting transcript). As explained in Section 5.1, however, any two-party protocol $\Pi$ that assumes unique SIDs can be trivially transformed to a protocol $\Pi'$ that does not: $\Pi'$ simply has the parties exchange nonces and then uses the concatenation of these nonces as a SID for an execution of the original protocol $\Pi$. In this case, we refer to $\Pi'$ as the "SID-enhancement" of $\Pi$. What we show here, then, is that if a protocol $\Pi$ is secure according to our notion, then its "SID-enhancement" $\Pi'$ is secure according to the definition of [3]. We stress that our proposition relates to the *weak corruption model* defined in [3] in which a $\mathsf{Corrupt}$ query reveals the password used by that party (but not its internal state).

**Proposition A.2** *Let $\Pi$ and $\Pi'$ be as discussed above. If $\Pi$ securely realizes $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ in the presence of static adversaries, then $\Pi'$ is secure according to the definition of* [3] *for the weak corruption model.*

**Proof (sketch):** On a high level, we view an adversary $\widehat{A}$ within the model of [3] as an environment in our model (once the issue of SIDs is taken care of as discussed above). The security of $\Pi$ in our model then implies that $\widehat{A}$ cannot distinguish an interaction with the real protocol from an interaction with the simulator $S$ and the ideal functionality $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$. In the latter case, however, the session keys are chosen uniformly at random (and hence $\widehat{A}$ has advantage 0) unless the simulator makes a successful $\mathsf{TestPwd}$ query. Now, by Lemma A.1, the number of $\mathsf{TestPwd}$ queries is at most $q$ (since $S$ does not make a $\mathsf{TestPwd}$ queries when $\widehat{A}$ only eavesdrops), and hence the probability of a successful $\mathsf{TestPwd}$ query is at most $q/|\mathcal{D}|$. (We assume passwords chosen at uniform from $\mathcal{D}$, but the proof can be modified appropriately for arbitrary password distributions.)

We now provide the details. (The description below assumes the reader is familiar with the model of [3] and the terms used therein.) Fix some dictionary $\mathcal{D}$ (with $|\mathcal{D}| \geq 2$) and any PPT adversary $\widehat{A}$ attacking $\Pi'$ within the model of [3]. The advantage of $\widehat{A}$ (in the sense of [3]) is denoted by $\mathsf{Adv}_{\widehat{A}}(k)$. Let $q = q(k)$ denote the number of $\mathsf{Send}$ queries made by $\widehat{A}$.

We construct an environment $\mathcal{Z}$ which either interacts with the "canonical" real-world adversary $\mathcal{A}$ (who simply follows the instructions of $\mathcal{Z}$) and players executing the real protocol $\Pi$, or with a simulator $S$ and $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$. Environment $\mathcal{Z}$ will run $\widehat{A}$ as a subroutine, simulating executions of $\Pi'$ for $\widehat{A}$ as follows:

[15] In the model of [3], "active interactions" are measured by the number of queries to a "send" oracle (as opposed to queries to an "execute" oracle which indicate passive eavesdropping).

- $\mathcal{Z}$ obtains from $\mathcal{A}$ any common random string used for $\Pi$, and gives this string to $\widehat{A}$. The environment also chooses a password $pw_{i,j}$ uniformly from $\mathcal{D}$ for each pair of parties $P_i, P_j$.

- When $\widehat{A}$ queries $\mathsf{Execute}(i, j)$,[16] environment $\mathcal{Z}$ chooses random nonces $n_i$ and $n_j$ of length $k$ and sets $sid = n_i|n_j$. It then gives input $(\mathsf{NewSession}, sid, i, j, pw_{i,j}, \mathsf{client})$ to $P_i$, input $(\mathsf{NewSession}, sid, j, i, pw_{i,j}, \mathsf{server})$ to $P_j$, and the command $(\mathsf{execute}, sid, i, j)$ to $\mathcal{A}$. Adversary $\mathcal{A}$ then forwards messages to and from $P_i$ and $P_j$, and returns the resulting transcript $\tau$ to $\mathcal{Z}$. Finally, $\mathcal{Z}$ hands $\widehat{A}$ the transcript $(n_i, n_j, \tau)$.

- When $\widehat{A}$ requests (via a $\mathsf{Send}$ query) that $P_i$ initiate an interaction with $P_j$, environment $\mathcal{Z}$ chooses a random string $n_i$ of length $k$ and gives $n_i$ to $\widehat{A}$. If $\widehat{A}$ later sends message $n_j$ to this instance, $\mathcal{Z}$ sets $sid = n_i|n_j$ and gives input $(\mathsf{NewSession}, sid, i, j, pw_{i,j}, \mathsf{client})$ to $P_i$ and the command $(\mathsf{send}, sid, i, \mathsf{client})$ to $\mathcal{A}$. Adversary $\mathcal{A}$ then forwards messages to and from (this instance of) $P_i$ and $\mathcal{Z}$, who in turn forwards these messages to and from $\widehat{A}$ each time $\widehat{A}$ makes a subsequent $\mathsf{Send}$ query to this instance.

  When $\widehat{A}$, impersonating $P_i$ in the role of a client, initiates an interaction with $P_j$ by sending a message $n_i$, environment $\mathcal{Z}$ chooses a random $n_j$ of length $k$, returns $n_j$ to $\widehat{A}$, sets $sid = n_i|n_j$, gives input $(\mathsf{NewSession}, sid, j, i, pw_{i,j}, \mathsf{server})$ to $P_j$, and gives the command $(\mathsf{send}, sid, j, \mathsf{server})$ to $\mathcal{A}$. Then $\mathcal{Z}$ and $\mathcal{A}$ proceed as above.

- Note that any time a party completes execution of the protocol, $\mathcal{Z}$ learns the value of the session key output by that party (this is because the environment obtains the outputs directly from the parties' output tapes). Therefore, when $\widehat{A}$ makes a $\mathsf{Reveal}$ query for a particular instance, $\mathcal{Z}$ finds the corresponding session key and returns that key to $\widehat{A}$.

- When $\widehat{A}$ makes a $\mathsf{Corrupt}$ query for a particular instance, $\mathcal{Z}$ replies with the password that was chosen for that instance (i.e., for the pair of parties taking part in that instance).[17]

- When $\widehat{A}$ makes a $\mathsf{Test}$ query for a particular instance, $\mathcal{Z}$ finds the corresponding session key $sk$ and flips a coin $b$. If $b = 0$ then $\mathcal{Z}$ gives $sk$ to $\widehat{A}$; otherwise, it gives $\widehat{A}$ a randomly-chosen key of the appropriate length.

Finally, $\mathcal{Z}$ outputs 1 iff $\widehat{A}$ correctly guesses the value of $b$. Clearly, the probability that $\mathcal{Z}$ outputs 1 in the real world is $1/2 \cdot (1 + \mathsf{Adv}_{\widehat{A}}(k))$ by definition of the advantage of $\widehat{A}$.

Since $\Pi$ securely realizes $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$, there exists a simulator $S$ for the adversary $\mathcal{A}$ described above. By Lemma A.1, $S$ does not make any $\mathsf{TestPwd}$ queries to its functionality when it receives a command $(\mathsf{execute}, sid, i, j)$ from $\mathcal{Z}$ (except possibly with negligible probability). We now observe the following:

1. $S$ is (essentially) limited to at most $q$ $\mathsf{TestPwd}$ queries (corresponding to the $q$ $\mathsf{Send}$ queries of $\widehat{A}$). Thus, $\mathcal{S}$ will obtain a "correct guess" in response to one of these queries with probability at most $q/|\mathcal{D}|$.

2. Assuming $S$ never correctly guesses a password, the session keys generated by the functionality (and thus obtained by $\mathcal{Z}$) are random, independent of the rest of the experiment, and in

[16] Technically, an $\mathsf{Execute}$ query should also specify particular instances of parties $i$ and $j$, but we have left these out for simplicity since they are inconsequential to our proof.

[17] In the model of [3] the adversary is also allowed to replace a password when it makes a $\mathsf{Corrupt}$ query. In such a case, $\mathcal{Z}$ will just use the new (adversarially-chosen) password in all subsequent (new) protocol executions for the relevant parties.

particular independent of the view of $\widehat{A}$. In such a case, $\widehat{A}$ guesses $b$ correctly with probability exactly $1/2$.

We therefore conclude that the probability that $\mathcal{Z}$ outputs 1 when interacting with $S$ is at most $q/|\mathcal{D}| + (1 - q/|\mathcal{D}|) \cdot 1/2 = q/2|\mathcal{D}| + 1/2$. Since $\Pi$ securely realizes $\widehat{\mathcal{F}}_{\mathsf{pwKE}}$ (implying that the probability that $\mathcal{Z}$ outputs 1 in the real world is negligibly close to the probability that it outputs 1 in the ideal world), we have that $|\mathsf{Adv}_{\widehat{A}}(k) - q/|\mathcal{D}||$ is negligible, as desired. $\square$