# Efficient and Secure Authenticated Key Exchange Using Weak Passwords

Jonathan Katz[*] Rafail Ostrovsky[†] Moti Yung[‡]

**Abstract**

Mutual authentication and authenticated key exchange are fundamental techniques for enabling secure communication over public, insecure networks. It is well-known how to design secure protocols for achieving these goals when parties share *high-entropy* cryptographic keys in advance of the authentication stage. Unfortunately, it is much more common for users to share weak, *low-entropy* passwords which furthermore may be chosen from a known space of possibilities (say, a dictionary of English words). In this case, the problem becomes much more difficult as one must ensure that protocols are immune to *off-line dictionary attacks* in which an adversary exhaustively enumerates all possible passwords in an attempt to determine the correct one.

We propose a 3-round protocol for password-only authenticated key exchange, and provide a rigorous proof of security for our protocol based on the decisional Diffie-Hellman assumption. The protocol assumes only public parameters — i.e., a "common reference string" — which can be "hard-coded" into an implementation of the protocol; in particular, and in contrast to some previous work, our protocol does *not* require either party to generate and share a public key in addition to sharing a password. The protocol is also remarkably efficient, requiring computation only (roughly) 4 times greater than "classical" Diffie-Hellman key exchange which provides no authentication at all. Ours is the first protocol for password-only authentication which is both *practical* and *provably-secure using standard cryptographic assumptions.*

## 1 Introduction

Protocols for mutual authentication of two parties and generation of a cryptographically-strong shared key between them (*authenticated key exchange*) are fundamental primitives for achieving secure communication over public, insecure networks. Indeed, protocols for mutual authentication are necessary because one needs to know "with whom one is communicating", while key-exchange protocols are required because private-key encryption schemes and message authentication codes rely on shared cryptographic keys which must be refreshed

[*]`jkatz@cs.umd.edu`. Department of Computer Science, University of Maryland. Work done while at Columbia University and Telcordia Technologies.

[†]`rafail@cs.ucla.edu`. Department of Computer Science, UCLA. Work done while at Telcordia Technologies. Work at UCLA was supported in part by a gift from Teradata, an Intel equipment grant, NSF Cybertrust grant #0430254, an OKAWA research award, the B. John Garrick Foundation, and a Xerox Innovation Group award.

[‡]`moti@cs.columbia.edu`. Department of Computer Science, Columbia University.

periodically. Higher-level protocols are frequently developed and analyzed assuming the existence of "secure channels" between all parties, yet this assumption cannot be realized without a secure mechanism for implementing such channels using previously-shared information.

The importance of secure key exchange was recognized as early as the seminal work of Diffie and Hellman [19], which shows how two parties can share a cryptographically-strong key that remains hidden from any *passive* eavesdropper. The Diffie-Hellman protocol, however, does not provide any form of authentication (i.e., a guarantee that the intended partners are sharing the key with *each other*), and in particular it does not protect against an *active* adversary who may inject messages, impersonate one (or both) of the parties, or otherwise control the communication in the network. Achieving any form of authentication inherently requires some information to be shared between the communicating parties in advance of the authentication stage. Historically, authentication protocols were designed under the assumption that the shared information takes the form of high-entropy cryptographic keys: either a *secret key* which can be used for symmetric-key encryption or message authentication, or *public keys* (exchanged by the parties, while the corresponding private keys are kept secret) which can be used for public-key encryption or digital signatures. Extensive attention has been given to the problem of designing two-party authentication and authenticated key-exchange protocols under such assumptions, and a number of provably-secure protocols relying on shared cryptographic keys are known (see, e.g., [8, 20, 4, 1, 46, 14, 15]).

It is important to recognize that protocols which assume cryptographically-strong pre-shared information simply do not apply — or are trivially seen to be insecure — when the parties share a "short", low-entropy password instead. As one trivial example to indicate the difficulties that arise, consider the simple (unidirectional) authentication protocol in which one party sends a random nonce $r$ and the other party replies with $y = H(pw, r)$, where $pw$ represents the shared password and $H$ is a cryptographic hash function. While this protocol can be proven secure (if $H$ is modeled as a random oracle) when $pw$ has high entropy, it is completely insecure when the entropy of $pw$ is small. Indeed, in the latter case a passive adversary who obtains a single transcript $(r, y)$ can recover the password by mounting an *off-line dictionary attack*, trying all values of $pw'$ until one satisfying $y = H(pw', r)$ is found. Other naïve attempts at "bootstrapping" a cryptographic key from a weak password can also be shown to be insecure; as an example, using $pw$ as a "seed" to generate a public key $\mathsf{VK}$ for a secure signature scheme also enables an off-line dictionary attack if $\mathsf{VK}$ is sent in the clear, or even if a signature is sent in the clear (as an adversary can enumerate all potential public keys and then attempt to verify the signature with respect to each of them). Note also that "scrambling" $pw$ by, say, hashing it will also be of no help, as the entropy of the output of the hash cannot be greater than the entropy of its input.

The above represents a serious problem in practice, as it is well-known that most users choose passwords poorly [43, 38, 48, 51], and that compromise of even a single user's account can lead to compromise of an entire network. Unfortunately, it has proven difficult to design password-based protocols offering rigorous security guarantees (see [37, Chap. 12]), and there is a long history of protocols for this domain being proposed and subsequently broken (cf. the attacks shown in [6, 45, 41]). Theoretical progress toward developing provably-secure solutions for the password-based setting has been slow, with the first formal models and proofs of security appearing only recently (see below). The problem is difficult in part because, as we have mentioned above, it requires "bootstrapping" from a weak shared secret to a strong one; furthermore, due to the strong adversarial model considered (cf. Sections 2.1 and 2.2), it is not even *a priori* clear that a solution is possible.

Initial consideration of *password-based authentication* assumed a "hybrid" model in which the client stores the server's public key in addition to sharing a password with the server. In this setting, Gong, Lomas, Needham, and Saltzer [39, 27] were the first to present authentication protocols with heuristic resistance to off-line dictionary attacks. Formal definitions and rigorous proofs of security in this setting were first given by Halevi and Krawczyk [29]; see also [9].

The above-described "hybrid" setting suffers from the disadvantage that the client must store the server's public key (and if the client will need to authenticate to multiple servers, the client must store multiple public keys); in some sense, this obviates the reason for considering password-based protocols in the first place: namely, that human users cannot remember or securely store long, high-entropy keys. This drawback has motivated research on *password-only* protocols in which the client needs to remember *only* a (short) password. Bellovin and Merritt [6] were the first to consider this, more challenging, setting: they show a number of attacks that can arise, and introduce a set of protocols for so-called "encrypted key exchange" (EKE) that have formed the basis for much future work in this area [7, 28, 49, 30, 31, 40, 50]. Each of the works just mentioned, however, provides only heuristic and informal security arguments for the protocols they propose; in fact, attacks against many of these protocols have been shown [45, 41], emphasizing the need for rigorous proofs of security in a formal, well-defined model.

Formal models of security for the password-only setting were given independently by Bellare, Pointcheval, and Rogaway [2] (building on [4, 5, 40]) and Boyko, MacKenzie, Patel, and Swaminathan [41, 11, 10] (building on [1, 46]). Bellare et al. [2] also present a proof of security for the two-flow protocol at the "core" of Diffie-Hellman-based EKE [6] in the *ideal cipher model*,[1] while Boyko et al. [41, 11] introduce new RSA- and Diffie-Hellman-based protocols, and prove their security in the *random oracle model*.[2] Subsequently, Goldreich and Lindell [25] introduced a different model of security for this setting and showed the first protocol for password-only key exchange which is provably secure under standard cryptographic assumptions (and no ideal ciphers/random oracles); their protocol does not require any additional setup beyond the passwords shared by the parties. This work is remarkable in that it shows that password-based authentication is possible under very weak assumptions. On the other hand, their protocol does not lead to a practical solution as it requires techniques from generic two-party secure computation (making their protocol computationally inefficient) and concurrent zero-knowledge (making the round-complexity of their protocol prohibitive). Thus, their protocol should best be viewed as a feasibility result which does not settle the question of whether an *efficient* and *practical* solution is possible. Note that efficiency is particularly important in this setting, which is motivated by inherently practical considerations: i.e., human users' inability to remember long keys. We

[1]The ideal cipher model assumes a public "oracle" which implements a random keyed, invertible permutation; in practice, this oracle is instantiated with a block cipher. See further discussion in Section 1.1.

[2]The random oracle model [3] assumes a public "oracle" which implements a random function; in practice, this oracle is instantiated with a cryptographic hash function. (Note that the ideal cipher model implies the random oracle model, while the converse is not known.) See further discussion in Section 1.1.

also remark that the Goldreich-Lindell protocol, unlike the protocol presented here, does not tolerate concurrent executions of the protocol by the same party.

## 1.1 Our Contributions

As the above discussion indicates, previously-suggested protocols for password-only key exchange can be classified as either (1) having no rigorous proof of security; (2) having a rigorous proof of security based on standard cryptographic assumptions, but impractical; or (3) having a rigorous proof of security in an idealized model (such as the ideal cipher or random oracle models). We have already discussed why proofs of security are crucial in this setting, and why developing *practical* solutions to the problem of password-only key exchange is of great importance. As for the third class of solutions, it is widely recognized in the cryptographic community (see, e.g., [3, 12]) that proofs of security in the various idealized models discussed earlier are useful as a "sanity check" insofar as they lend a measure of confidence to protocols whose security would otherwise only be heuristic; however, they do *not* provide an "iron-clad" guarantee of security in the real world. As an example [12], encryption and signature schemes are known which can be proven secure in these idealized models but for which *any instantiation of the scheme in the real world* (where, e.g., the random oracle is instantiated by a cryptographic hash function) *is insecure.* Thus, proofs of security in idealized models are at best unsatisfying, and at worst dangerous.

We present here the first protocol for password-only key exchange which is both *practical* and *provably-secure under standard cryptographic assumptions* (in particular, without assuming any idealized model). Security is proven in the model of Bellare et al. [2] based on the decisional Diffie-Hellman assumption [19], a well-studied cryptographic assumption used in constructing and analyzing previous protocols for authenticated key-exchange (in the password-based setting and otherwise). Our protocol is remarkably efficient even when compared to the original key-exchange protocol of Diffie and Hellman — which, as we have noted, provides no authentication at all: our protocol uses only three rounds of communication, and requires computation only (roughly) four times greater than the original Diffie-Hellman scheme.

Our protocol assumes public parameters available to all parties in the network. These parameters are assumed to be generated in a trusted manner by, say, the same entity who writes the software implementing the protocol. (Note that this entity is anyway being trusted to implement the protocol correctly, and not to embed any malware or covert channels into the implementation.) The assumption of public parameters is significantly weaker — in both a theoretical and practical sense — than the "hybrid" model discussed earlier in which clients are required to store a public key for each server with whom they wish to communicate or, alternatively, to store the public key of a certificate authority (CA) who certifies servers' public keys. Advantages of using public parameters as opposed to public keys include:

- The parameters required by our protocol can be generated in such a way that there is no "trapdoor" information associated with them. This, in turn, implies that there is no "master" secret key whose compromise would render the entire system insecure.

- Avoiding the use of public keys avoids the need to handle revocation, and also eliminates the need to transmit certificates and verify certificate chains.

- Once the public parameters are established, any client and server who share a password can use the protocol without having to first obtain a certificate from a CA.

As an additional contribution, we propose a formal definition of *forward secrecy* for password-only key-exchange protocols which we believe simplifies and unifies previous definitional work in this area. Roughly speaking, forward secrecy guarantees that a protocol remains secure even against an adversary who may obtain some clients' passwords and who may also change the passwords stored at various servers. It also guarantees protection against malicious clients (who may choose "bad" passwords in an attempt to learn other users' passwords). As an indication of what can go wrong in this stronger adversarial model, we show that a seemingly-innocuous change to our protocol allows for an explicit attack by such an adversary. We then prove, however, that our protocol as specified does indeed achieve forward secrecy.

### 1.2 Subsequent Progress

Due in large part to the practical significance and importance of password-based authentication, active research continues in this area. We only highlight those results most related to what is shown here. Perhaps most interesting in this regard is the work of Gennaro and Lindell [24], who make explicit the intuition underlying the core ideas of our construction and also present and prove secure an abstraction (as well as a generalization) of the framework we use; a nice consequence is that they obtain as corollaries of their work efficient protocols based on alternative number-theoretic assumptions. Nguyen and Vadhan [44] show a simpler version of the Goldreich-Lindell protocol which achieves a weaker level of security than that considered in either [25] or here. Though their protocol is somewhat more efficient than that of [25], it relies on generic two-party secure computation and is therefore still not efficient enough to be used in practice. Jiang and Gong [32] show an efficient protocol for password-based key exchange based on the decisional Diffie-Hellman assumption; the main advantage of their protocol in comparison to the one presented here is that their protocol achieves explicit mutual authentication in three rounds (whereas the protocol shown here would require an additional, fourth round). In [13], a stronger definition of security for password-based key exchange is proposed; a protocol satisfying that stronger definition, building on the work here and in [24], is shown there as well. The above (in addition to [25] and the present work) are — to the best of our knowledge — the only schemes currently known with proofs of security under standard cryptographic assumptions.

More efficient variants of our protocol are suggested in [13, 34]. Threshold variants of our protocol (in which the password is shared across multiple servers to defend against server compromise) have also been suggested [21, 34].

## 2 Definitions and Preliminaries

We begin with an informal overview of the model of security used here, intended primarily for those unfamiliar with this area. Section 2.2 contains a formal specification of the security model, following [2]. In Section 2.3 we propose an extension to the basic security model which enables a rigorous definition of *forward secrecy* [20] in the password-only setting. Finally, Section 2.4 reviews a number of cryptographic components used by our protocol.

## 2.1 Informal Overview of the Adversarial Model

In our setting two parties within a larger network who share a weak (low-entropy) password wish to authenticate each other and generate a cryptographically-strong session key to secure their future communication. They wish to do so in the presence of a powerful adversary who controls *all* communication in the network; this adversary may view, tamper with, deliver out-of-order, or refuse to deliver messages sent by the honest parties. The adversary may also initiate concurrent (arbitrarily-interleaved) executions of the protocol between the honest parties; during these executions, the adversary may attempt to impersonate one (or both) of the parties or may simply eavesdrop on honest executions of the protocol. The adversary may also expose multiple previous session keys in a way we describe below. The adversary succeeds (informally) if it can distinguish any session key generated by an honest party from a randomly-chosen session key.[3]

The description above corresponds to the "basic" adversarial model formalized in Section 2.2. We may additionally consider an adversary who can corrupt parties and thereby either (1) obtain a client's password, or (2) modify the client's password stored on a server. We consider this type of adversary when defining "forward secrecy" in Section 2.3.

A notion of security in the password-based setting must be defined carefully. Since, by their nature, passwords are chosen from a small space of possibilities, an adversary can always try each possible password one-by-one in an *on-line impersonation* attack; once the correct password is found, the adversary can clearly succeed in impersonating either party. With this in mind, we say a password-only protocol is secure (informally) if an exhaustive, on-line guessing attack of this sort is the *best* an adversary can do (cf. Definition 1, below). In particular, a secure protocol is resistant to *off-line* attacks in which an adversary passively monitors executions of the protocol and attempts to determine the correct password by examining the transcripts of these executions. A protocol satisfying our definition of security is also resilient to other types of attacks: for example, the definition rules out "man-in-the-middle" attacks in which an adversary shares a (separate) key with each party although the parties believe they have shared a (single) key with each other.

From a practical point of view, on-line attacks are the hardest for an adversary to mount, and they are also the easiest to detect. Furthermore, on-line attacks may be limited by, for example, shutting down a user's account after a certain number of failed authentication attempts. It is therefore realistic to assume that the number of on-line attacks an adversary is able to execute is severely limited, while other attacks (e.g., eavesdropping, off-line password guessing) are not. In any case, as noted above, *any* password-based protocol can ultimately be broken by an on-line attack and so the present definition is the best one can hope for in the password-based setting.

[3]The adversary does not succeed by simply "disrupting" the protocol and preventing the parties from agreeing on a shared key; in our model such an attack is impossible to prevent, as the adversary can simply block *all* communication between the honest parties. Note also that we are primarily concerned with the setting of *implicit* authentication; see Section 2.2.1 for a discussion of *explicit* authentication.

## 2.2 Definition of Security for Password-Based AKE

We essentially follow the definition of Bellare, Pointcheval, and Rogaway [2], which is based on prior work by Bellare and Rogaway in the non-password setting [4, 5].

**Participants, passwords, and initialization.** We assume a fixed set[4] of protocol participants (also called principals or users) each of which is either a client $C \in \mathsf{Client}$ or a server $S \in \mathsf{Server}$, where $\mathsf{Client}$ and $\mathsf{Server}$ are disjoint. Each $C \in \mathsf{Client}$ is assumed to have a password $pw_C$, while each $S \in \mathsf{Server}$ is assumed to have a vector $PW_S = \langle pw_{S,C} \rangle_{C \in \mathsf{Client}}$ which contains the passwords of each of the clients. (The assumption that every client shares a password with every server is made merely for convenience. One could just as well assume that arbitrary pairs of users share passwords.)

Prior to any execution of the protocol we assume that an initialization phase occurs during which public parameters (if any) are established and passwords $pw_C$ are chosen for each client $C$. Recall that $pw_C$ is assumed to be a low-entropy secret; therefore, we assume that $pw_C$ (for each client $C$) is chosen independently and uniformly at random[5] from a "dictionary" $D$ of size $N$, where $N$ is a fixed constant which is independent of the security parameter. The correct passwords are then stored at each server so that $pw_{S,C} = pw_C$ for all $C \in \mathsf{Client}$ and $S \in \mathsf{Server}$.

In general, it is possible for additional information to be generated during this initialization phase. For example, in the "hybrid" model [29, 9] public/secret key pairs are generated for each server and the secret key is given as input to the appropriate server, while the public key is provided to all participants (and is also given to the adversary). For the protocol presented here, we require only the weaker requirement of a single set of public parameters which are provided to each party and also to the adversary.

**Execution of the protocol.** In the real world, a protocol determines how principals behave in response to input from their environment. In the formal model, these inputs are provided by the adversary. Each principal is assumed to be able to execute the protocol multiple times (possibly concurrently) with different partners; this is modeled by allowing each principal to have an unlimited number of *instances* [5, 2] with which to execute the protocol. We denote instance $i$ of user $U$ as $\Pi_U^i$. A given instance may be used only once. The adversary is given oracle access to these different instances; furthermore, each instance maintains (local) state which is updated during the course of the experiment. In particular, each instance $\Pi_U^i$ has associated with it the following variables, initialized as NULL or FALSE (as appropriate) during the initialization phase:

- $\mathsf{sid}_U^i$, $\mathsf{pid}_U^i$, and $\mathsf{sk}_U^i$ are variables containing the *session id*, *partner id*, and *session key* for an instance, respectively. Computation of the session key is, of course, the ultimate goal of the protocol. The session id is simply a way to keep track of the

[4]This is for simplicity only, and one could augment the model to allow the adversary to dynamically add new users (and choose their names) during the course of the experiment. By examining the proof, it is easy to see that our protocol remains secure in such a scenario.

[5]The assumption of independent, uniform passwords from a constant-size dictionary is made for simplicity only, and our proof of security extends easily to handle other cases such as dictionaries whose size depends on the security parameter, non-uniform distributions on passwords, or different distributions for different clients (of course, Definition 1 must be suitably modified in each case). See Section 2.2.1. An even more general model for password selection is considered in [13].

different executions of a particular user $U$; without loss of generality, we simply let this be the (ordered) concatenation of all messages sent and received by instance $\Pi_U^i$. The partner id denotes the identity of the user with whom $\Pi_U^i$ believes it is interacting. (Note that $\mathsf{pid}_U^i$ can never equal $U$.)

- $\mathsf{acc}_U^i$ and $\mathsf{term}_U^i$ are boolean variables denoting whether a given instance has accepted or terminated, respectively. Termination means that a given instance is done sending and receiving messages; acceptance indicates successful termination, where the semantics of "success" depend on the protocol. In our case, acceptance implies that the instance believes it has established a session key with its intended partner; thus, when an instance $\Pi_U^i$ accepts, $\mathsf{sid}_U^i$, $\mathsf{pid}_U^i$, and $\mathsf{sk}_U^i$ are no longer null.

- $\mathsf{state}_U^i$ records any state necessary for execution of the protocol by $\Pi_U^i$.

- $\mathsf{used}_U^i$ is a boolean variable denoting whether an instance has begun executing the protocol; this is a formalism which will ensure that each instance is used only once.

As highlighted earlier, the adversary is assumed to have complete control over all communication in the network and the adversary's interaction with the principals (more specifically, with the various instances) is modeled via access to *oracles* which we describe now. The state of an instance may be updated during an oracle call, and the oracle's output may depend upon the state of the relevant instance. The oracle types are:

- $\mathsf{Send}(U, i, M)$ — This sends message $M$ to instance $\Pi_U^i$. Assuming $\mathsf{term}_U^i = \text{FALSE}$, this instance runs according to the protocol specification, updating state as appropriate. The output of $\Pi_U^i$ (i.e., the message sent by the instance) is given to the adversary, who also receives the updated values of $\mathsf{sid}_U^i$, $\mathsf{pid}_U^i$, $\mathsf{acc}_U^i$, and $\mathsf{term}_U^i$.

- $\mathsf{Execute}(C, i, S, j)$ — If $\Pi_C^i$ and $\Pi_S^j$ have not yet been used (where $C \in \mathsf{Client}$ and $S \in \mathsf{Server}$), this oracle executes the protocol between these instances and outputs the transcript of this execution. This oracle call represents passive eavesdropping of a protocol execution. In addition to the transcript, the adversary receives the values of $\mathsf{sid}$, $\mathsf{pid}$, $\mathsf{acc}$, and $\mathsf{term}$, for both instances, at each step of protocol execution.

- $\mathsf{Reveal}(U, i)$ — This outputs the current value of session key $\mathsf{sk}_U^i$. This oracle call models possible leakage of session keys due to, for example, improper erasure of session keys after use, compromise of a host computer, or cryptanalysis.

- $\mathsf{Test}(U, i)$ — This oracle does not model any real-world capability of the adversary, but is instead used to define security. A random bit $b$ is generated; if $b = 1$ the adversary is given $\mathsf{sk}_U^i$, and if $b = 0$ the adversary is given a random session key. The adversary is allowed only a single $\mathsf{Test}$ query, at any time during its execution.

**Partnering.** Let $C \in \mathsf{Client}$ and $S \in \mathsf{Server}$. We say that instances $\Pi_C^i$ and $\Pi_S^j$ are *partnered* if: (1) $\mathsf{sid}_C^i = \mathsf{sid}_S^j \neq \text{NULL}$; and (2) $\mathsf{pid}_C^i = S$ and $\mathsf{pid}_S^j = C$. The notion of partnering will be fundamental in defining both correctness and security.

**Correctness.** To be viable, a key-exchange protocol must satisfy the following notion of correctness: if $\Pi_C^i$ and $\Pi_S^j$ are partnered then $\mathsf{acc}_C^i = \mathsf{acc}_S^j = \text{TRUE}$ and $\mathsf{sk}_C^i = \mathsf{sk}_S^j$ (i.e., they both accept and conclude with the same session key).

**Advantage of the adversary.** Informally, the adversary succeeds if it can guess the bit $b$ used by the $\mathsf{Test}$ oracle. Before formally defining the adversary's success, we must first define a notion of *freshness.* An instance $\Pi_U^i$ is *fresh* unless one of the following is true at the conclusion of the experiment: (1) at some point, the adversary queried $\mathsf{Reveal}(U, i)$; or (2) at some point, the adversary queried $\mathsf{Reveal}(U', j)$, where $\Pi_{U'}^j$ and $\Pi_U^i$ are partnered. We will allow the adversary to succeed only if its $\mathsf{Test}$ query is made to a fresh instance. Note that this is necessary for any reasonable definition of security; otherwise, the adversary could always succeed by, e.g., submitting a $\mathsf{Test}$ query for an instance for which it had already submitted a $\mathsf{Reveal}$ query.

We say an adversary $\mathcal{A}$ *succeeds* if it makes a single query $\mathsf{Test}(U, i)$ to a fresh instance $\Pi_U^i$, with $\mathsf{acc}_U^i = \text{TRUE}$ at the time of this query, and outputs a single bit $b'$ with $b' = b$ (recall that $b$ is the bit chosen by the $\mathsf{Test}$ oracle). We denote this event by $\mathsf{Succ}$. The advantage of adversary $\mathcal{A}$ in attacking protocol $P$ is then given by:

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

where the probability is taken over the random coins used by the adversary and the random coins used during the course of the experiment (including the initialization phase).

It remains to define what we mean by a secure protocol. Note that a probabilistic polynomial-time (PPT) adversary can always succeed by trying all passwords one-by-one in an on-line impersonation attack; this is possible since the size of the password dictionary is constant. Informally, a protocol is secure if this is the best an adversary can do. Formally, an instance $\Pi_U^i$ represents an *on-line attack* if both the following are true at the time of the $\mathsf{Test}$ query: (1) at some point, the adversary queried $\mathsf{Send}(U, i, *)$; and (2) at some point, the adversary queried $\mathsf{Reveal}(U, i)$ or $\mathsf{Test}(U, i)$. In particular, instances with which the adversary interacts via $\mathsf{Execute}$ queries are not counted as on-line attacks. The number of on-line attacks represents a bound on the number of passwords the adversary could have tested in an on-line fashion. This motivates the following definition:

**Definition 1** Protocol $P$ is a secure protocol for password-only authenticated key-exchange if, for all dictionary sizes $N$ and for all PPT adversaries $\mathcal{A}$ making at most $Q(k)$ on-line attacks, there exists a negligible function $\varepsilon(\cdot)$ such that:

$$\mathsf{Adv}_{\mathcal{A},P}(k) \leq Q(k)/N + \varepsilon(k).$$

$\diamondsuit$

The above definition ensures that the adversary can (essentially) do no better than guess a single password during each on-line attack. Calls to the $\mathsf{Execute}$ oracle, which are not included in $Q(k)$, are of no help to the adversary in breaking the security of the protocol; this means that passive eavesdropping and off-line dictionary attacks are of no use.

We remark that some definitions of security (e.g., [2, 25, 44]) allow the adversary to guess more than one password per on-line attempt. We believe the strengthening given by the above definition (in which the adversary can guess only a *single* password per on-line attempt) is important. The space of possible passwords is assumed small to begin with, so any degradation in security should be avoided if possible. This is not to say that protocols not meeting the above definition are fundamentally "insecure"; however, before using such a protocol one must then be aware of the constant implicit in the proof of security. Interestingly, for at least one protocol [50] an *explicit* attack is known which allows an adversary to guess two passwords per on-line attack.

### 2.2.1 Extensions of the Definition

We briefly discuss two extensions of the previous definition.

**More general password distributions.** As noted in footnote 5, it is possible to consider more general classes of password distributions. For example, instead of assuming that passwords are chosen uniformly from a fixed dictionary $D$, we may instead take $D$ to be a probabilistic polynomial-time algorithm that, on input $1^k$, outputs a password $pw$. This allows both for non-uniform distributions, as well as a dependence on the security parameter. For a set $S$, let $\mathsf{weight}_{D(1^k)}(S) \overset{\text{def}}{=} \Pr[D(1^k) \in S]$ and define

$$\mathsf{best\text{-}guess}_{D(1^k)}(Q) \overset{\text{def}}{=} \max_{S\,:\,|S|=Q} \{\mathsf{weight}_{D(1^k)}(S)\}.$$

Then Definition 2.2 would be modified, in the obvious way, to require that

$$\mathsf{Adv}_{\mathcal{A},P}(k) \leq \mathsf{best\text{-}guess}_{D(1^k)}(Q(k)) + \varepsilon(k).$$

It can be easily verified that the proof of security we give for our protocol extends to the case where passwords are chosen as above. We remark that that requirement that passwords are generated by a *polynomial-time* algorithm $D$ seems necessary for our proof, but it is not necessary for $\mathsf{best\text{-}guess}_{D(1^k)}(\cdot)$ to be efficiently computable.

**Explicit mutual authentication.** The definitions we have given are intended only for protocols achieving *implicit*, rather than *explicit*, authentication. (Indeed, the protocol we present here achieves only implicit authentication.) If an honest party $S$ interacts with some party $U$ (who is either the intended partner $C$ or an adversarial impersonator) then, very roughly speaking, $S$ *implicitly authenticates* $C$ if the session key $\mathsf{sk}$ that is generated by $S$ is known to $U$ if and only if $U = C$. In contrast, $S$ *explicitly authenticates* $C$ if the output $(\mathsf{sk}, \mathsf{acc})$ of $S$ is such that $\mathsf{sk}$ is known to $U$ *and* $\mathsf{acc} = 1$ if and only if $U = C$.

Turning this around, this means that in the case of protocols for explicit authentication an adversary successfully attacks the protocol if it can either determine the session key held by some instance, or if it can cause an instance to output $\mathsf{acc} = \text{TRUE}$ even though that instance is not partnered with any other instance. A subtlety here is that partnering must be redefined so that instances are considered partnered if their session id's match except possibly for the final message; otherwise it would be trivial for an adversary to succeed by simply forwarding all messages of the protocol *except the last* between two honest parties. (See [4, 2] for further discussion.) We also now say that an instance $\Pi_U^i$ represents an on-line attack if the adversary ever queried $\mathsf{Send}(U, i, *)$ (i.e., without requiring the adversary to also make a $\mathsf{Reveal}$ of $\mathsf{Test}$ query for this instance).

Using pseudorandom functions in a fairly standard manner, it is easy to add explicit authentication to any protocol achieving implicit authentication [2].

## 2.3 Forward Secrecy

The security definition of the previous section does not offer protection against an adversary who may compromise, say, a server and thereby either (1) obtain the password of a particular client, or (2) modify the value of a particular client's password stored by the server. Building on the framework of [2], we provide a definition of *forward secrecy* which addresses these concerns. (Note that forward secrecy [20] classically refers to ensuring security in case long-term secret information — e.g., passwords — are exposed. Following [2], however, we additionally allow the adversary to *modify* long-term secret information.) A number of definitions of forward secrecy in the password-only setting have been proposed (for example, four different notions of forward secrecy are mentioned in [2] alone); we believe our definition here simplifies and unifies previous definitional work.

To define forward secrecy, we must modify the basic model of the previous section in two orthogonal ways. First, we augment the adversarial model by introducing a new $\mathsf{Corrupt}$ oracle to which the adversary will be given access (actually, we will introduce two types of $\mathsf{Corrupt}$ oracles); this will allow us to model the adversarial actions listed above. Second, we modify the definition of security to obtain a meaningful definition (for example, once an adversary obtains a client's password it can trivially impersonate that client regardless of what protocol is used; this must be reflected somehow in the security definition). Actually, we will only modify our definition of freshness; the rest of the definition will remain the same. We now discuss each of these changes in turn.

**The corruption model.** We introduce two new oracles to model the two types of attacks listed above. Oracle $\mathsf{Corrupt}_1$ models the adversary's ability to learn clients' passwords: formally, $\mathsf{Corrupt}_1(C)$ returns $pw_C$ for $C \in \mathsf{Client}$.[6] Oracle $\mathsf{Corrupt}_2$ models the adversary's ability to modify passwords stored on a server: formally, $\mathsf{Corrupt}_2(S, C, pw)$ (for $S \in \mathsf{Server}$ and $C \in \mathsf{Client}$) sets $pw_{S,C} := pw$. We emphasize that the adversary can install different passwords on different servers for the same client. In the definition of [2], an adversary who installs a password $pw_{S,C}$ also learns the "actual" password $pw_C$; we make no such assumption here (instead, the adversary must make an explicit $\mathsf{Corrupt}_1$ query to obtain this password). In the case of a poorly-administered server, it may be easy to modify clients' passwords without learning their "actual" passwords. The oracle query $\mathsf{Corrupt}_2(S, C', pw)$ also models the case of a corrupt client $C'$ who selects his own password $pw$ in an arbitrary manner, and then stores it on server $S$.

**"Freshness".** Introduction of the $\mathsf{Corrupt}$ oracles necessitates new definitions of partnering, freshness, and on-line attacks. Let $C \in \mathsf{Client}$ and $S \in \mathsf{Server}$. We now say that instances $\Pi_C^i$ and $\Pi_S^j$ are *partnered* if: (1) $\mathsf{sid}_C^i = \mathsf{sid}_S^j \neq \text{NULL}$; (2) $\mathsf{pid}_C^i = S$ and $\mathsf{pid}_S^j = C$; and (3) $pw_C = pw_{S,C}$ at the time these instances were activated. (This last requirement is now necessary since an adversary can cause $pw_C \neq pw_{S,C}$ using a $\mathsf{Corrupt}_2$ query.) We

[6]Note that it does not matter here whether the adversary obtains the password by corrupting a server and learning a client's password stored thereon, or whether the adversary obtains the password directly from the client using, e.g., social engineering.

say that an instance $\Pi_U^i$ (with $\mathsf{pid}_U^i = U'$) is *fresh* unless one of the following is true at the conclusion of the experiment: (1) at some point, the adversary queried $\mathsf{Reveal}(U, i)$ or $\mathsf{Reveal}(U', j)$ where $\Pi_{U'}^j$ and $\Pi_U^i$ are partnered; (2) the adversary queried $\mathsf{Corrupt}_1(U)$ before a query $\mathsf{Send}(U, i, *)$; or (3) the adversary queried $\mathsf{Corrupt}_1(U')$ or $\mathsf{Corrupt}_2(U, U', *)$ before a query $\mathsf{Send}(U, i, *)$. The second and third conditions reflect the fact that if an adversary learns the password of some client $C$, then it can trivially impersonate any server to $C$, or $C$ to any server; the third condition also reflects the fact that once the adversary sets the value of $pw_{S,C}$ to some known value, it can then impersonate $C$ to $S$. Finally, we define *on-line attack* as follows:

- An instance $\Pi_C^i$ with $C \in \mathsf{Client}$ represents an on-line attack if (1) the adversary queried $\mathsf{Send}(C, i, *)$ before it queried $\mathsf{Corrupt}_1(C)$; and (2) the adversary subsequently queried $\mathsf{Reveal}(C, i)$, $\mathsf{Test}(C, i)$, or $\mathsf{Corrupt}_1(C)$.

- An instance $\Pi_S^i$ with $S \in \mathsf{Server}$ and $\mathsf{pid}_S^i = C$ represents an on-line attack if (1) the adversary queried $\mathsf{Send}(S, i, *)$ before it queried $\mathsf{Corrupt}_1(C)$ or $\mathsf{Corrupt}_2(S, C)$; and (2) the adversary subsequently queried $\mathsf{Reveal}(S, i)$, $\mathsf{Test}(S, i)$, or $\mathsf{Corrupt}_1(C)$.

Although the above definition is slightly cumbersome, it exactly matches the intuitive notion of what an on-line attack is: the number of on-line attacks exactly represents the maximum number of "password guesses" the adversary can make, and we do not count as on-line attacks scenarios in which the adversary already "knows" the password being used by an instance (e.g., in case the adversary had already queried $\mathsf{Corrupt}_1(C)$ before it makes a query $\mathsf{Send}(C, i)$).

Having defined partnering, freshness, and on-line attacks, the remainder of the security definition is exactly as in the basic case. That is:

- An adversary $\mathcal{A}$ *succeeds* (denoted by $\mathsf{Succ}$) if it makes a single query $\mathsf{Test}(U, i)$ to a fresh instance $\Pi_U^i$, with $\mathsf{acc}_U^i = \text{TRUE}$ at the time of this query, and outputs a single bit $b'$ with $b' = b$.

- The advantage of adversary $\mathcal{A}$ in attacking protocol $P$ is defined as
$$\mathsf{Adv}_{\mathcal{A},P}(k) \overset{\text{def}}{=} 2 \cdot \Pr[\mathsf{Succ}] - 1.$$

- Let $P$ be a protocol for password-only authenticated key-exchange that is secure in the basic sense of Definition 2.2. We say $P$ achieves forward secrecy if for all dictionary sizes $N$ and all PPT adversaries $\mathcal{A}$ making at most $Q(k)$ on-line attacks there exists a negligible function $\varepsilon(\cdot)$ such that:
$$\mathsf{Adv}_{\mathcal{A},P}(k) \leq Q(k)/N + \varepsilon(k).$$

We stress that although the above is syntactically equivalent to the basic definition given earlier, the differences are that $\mathcal{A}$ now has access to the two Corrupt oracles, and the notions of "freshness" and "on-line attacks" are modified appropriately.

## 2.4 Cryptographic Building Blocks

We briefly review the decisional Diffie-Hellman assumption (which underlies the security of our scheme), as well as some cryptographic building blocks used to construct our protocol.

**The decisional Diffie-Hellman (DDH) assumption [19].** Let $\mathcal{G}$ be an efficient algorithm which, on input $1^k$, outputs a description of a cyclic (multiplicative) group $\mathbb{G}$ of prime order $q$ where furthermore $|q| = k$. We associate $\mathbb{G}$ with a description of this group, and assume for simplicity that $q$ is implicit in $\mathbb{G}$. Additionally, we assume that group operations (namely, multiplication, membership testing, and finding a generator) in groups output by $\mathcal{G}(1^k)$ can be done in (expected) polynomial-time in $k$. Exponentiation and selecting a random group element can then also be done in polynomial time. We defer a concrete example of a candidate algorithm $\mathcal{G}$ to the end of this section.

We define $\bar{\mathbb{G}} \overset{\text{def}}{=} \mathbb{G} \setminus \{1\}$; since the order of $\mathbb{G}$ is prime, $\bar{\mathbb{G}}$ is exactly the set of generators of $\mathbb{G}$. Given a generator $g \in \bar{\mathbb{G}}$, a random group element can be selected by choosing $x \in \mathbb{Z}_q$ uniformly at random and computing $g^x$. If, instead, $x$ is chosen uniformly at random from $\mathbb{Z}_q^*$ we obtain a random generator. Given a group $\mathbb{G}$ as above, we define the set of *Diffie-Hellman tuples* as all tuples of the form $(g, h, g^a, h^a) \in \bar{\mathbb{G}}^4$ where $g$ and $h$ are generators and $a \in \mathbb{Z}_q^*$. We define the set of *random tuples* as all tuples of the form $(g, h, g^a, h^b) \in \bar{\mathbb{G}}^4$ where $g$ and $h$ are generators, $a, b \in \mathbb{Z}_q^*$, and furthermore $a \neq b$. Informally, the DDH problem for a group $\mathbb{G}$ is to distinguish Diffie-Hellman tuples from random tuples, and the DDH problem is "hard" in $\mathbb{G}$ if no poly-time algorithm can solve the DDH problem in this group with probability much better than 1/2 (i.e., guessing at random). To be more formal, we must in fact talk about the hardness of the DDH problem in groups output by $\mathcal{G}$. This leads to the following definition.

**Definition 2** Given $\mathcal{G}$ as above, and for any algorithm $D$, define

$$\begin{aligned}&\mathsf{Rand}_{D,\mathcal{G}}(k) \overset{\text{def}}{=} \\ &\qquad \Pr\left[\mathbb{G} \leftarrow \mathcal{G}(1^k); g, h \leftarrow \bar{\mathbb{G}}; a \leftarrow \mathbb{Z}_q^*; b \leftarrow \mathbb{Z}_q^* \setminus \{a\} : D(\mathbb{G}, g, h, g^a, h^b) = 1\right]\end{aligned}$$

and

$$\mathsf{DH}_{D,\mathcal{G}}(k) \overset{\text{def}}{=} \Pr\left[\mathbb{G} \leftarrow \mathcal{G}(1^k); g, h \leftarrow \bar{\mathbb{G}}; a \leftarrow \mathbb{Z}_q^* : D(\mathbb{G}, g, h, g^a, h^a) = 1\right].$$

We say the DDH problem is hard for $\mathcal{G}$ if $|\mathsf{DH}_{D,\mathcal{G}}(k) - \mathsf{Rand}_{D,\mathcal{G}}(k)|$ is negligible for all PPT algorithms $D$. The DDH assumption is that there exists a $\mathcal{G}$ for which the DDH problem is hard. If $\mathbb{G}$ is a group output by some $\mathcal{G}$ for which the DDH problem is hard, we will sometimes (informally) say the DDH problem is hard in $\mathbb{G}$. $\diamondsuit$

A standard example of an algorithm $\mathcal{G}$ for which the DDH problem is believed to be hard is the following: on input $1^k$ choose primes $p, q$ such that $p = 2q + 1$ and $|q| = k$; let $\mathbb{G}$ be the subgroup of quadratic residues in $\mathbb{Z}_p^*$. Note that $\mathbb{G}$ has order $q$, as desired. Of course, other choices for $\mathcal{G}$ are also possible.

**One-time signature schemes.** We provide a self-contained definition (following [26]) of the type of signature scheme required by our protocol.

**Definition 3** A signature scheme $\Sigma$ is a triple of PPT algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ such that:

- The key generation algorithm $\mathsf{Gen}$ takes as input a security parameter $1^k$ and returns verification key $\mathsf{VK}$ and signing key $\mathsf{SK}$.
- The signing algorithm $\mathsf{Sign}$ takes as input a signing key $\mathsf{SK}$ and a message $m$, and returns a signature $\mathsf{Sig}$. We denote this by $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(m)$.
- The verification algorithm $\mathsf{Vrfy}$ is a deterministic algorithm that takes as input a verification key $\mathsf{VK}$, a message $m$, and a signature $\mathsf{Sig}$ and returns a single bit. We denote this by $b = \mathsf{Vrfy}_{\mathsf{VK}}(m, \mathsf{Sig})$.

We require that for all $k$, all $(\mathsf{VK}, \mathsf{SK})$ output by $\mathsf{Gen}(1^k)$, all $m$, and all $\mathsf{Sig}$ output by $\mathsf{Sign}_{\mathsf{SK}}(m)$, we have $\mathsf{Vrfy}_{\mathsf{VK}}(m, \mathsf{Sig}) = 1$. $\diamondsuit$

The following definition describes the level of security which the signature scheme used in our protocol must satisfy. The required level of security is relatively weak; thus, signature schemes meeting this level of security are not only easy to construct, but can also be more efficient than typical signature schemes used in practice (which usually satisfy the stronger security notion introduced by [26]).

**Definition 4** $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is a secure one-time signature scheme if the following is negligible for all PPT algorithms $\mathcal{F}$:

$$\Pr[(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^k); (m, \mathsf{Sig}) \leftarrow \mathcal{F}^{\mathsf{Sign}_{\mathsf{SK}}(\cdot)}(1^k, \mathsf{VK}) : \mathsf{Vrfy}_{\mathsf{VK}}(m, \mathsf{Sig}) = 1],$$

where $\mathcal{F}$ makes only a single query to $\mathsf{Sign}_{\mathsf{SK}}(\cdot)$ and we require that $\mathsf{Sig}$ was not previously output by $\mathsf{Sign}_{\mathsf{SK}}(\cdot)$ on input $m$. $\diamondsuit$

The above definition corresponds to what is sometimes termed a "strong" signature scheme, in that it is also infeasible for the adversary to output a *different* (valid) signature corresponding to the same message $m$ that it submitted to its signing oracle.

A secure (strong) one-time signature scheme may be constructed based on any one-way function, and hence may be based on the DDH assumption (which implies the discrete logarithm assumption and thus a one-way function).

**Collision-resistant hashing.** Informally, a function $H$ is collision-resistant if it is infeasible to find any $x \neq x'$ for which $H(x) = H(x')$. Formally, let $\mathsf{CRHF}$ be an efficient algorithm which, on input $1^k$, outputs a description of an efficient hash function $H$ mapping $\{0,1\}^*$ to $\{0,1\}^k$. We say that $\mathsf{CRHF}$ is a collision-resistant hash family if the following is negligible for all PPT algorithms $A$:

$$\Pr[H \leftarrow \mathsf{CRHF}(1^k); (x, x') \leftarrow A(1^k, H) : x \neq x' \wedge H(x) = H(x')].$$

For our application, we will actually require that the $H$ output by $\mathsf{CRHF}$ map strings to $\mathbb{Z}_q$ for some given $q$ of length $k$. However, it is easy to modify any collision-resistant hash family as described above to obtain one satisfying this requirement.

A collision-resistant hash family may be constructed based on the DDH assumption (in fact, the weaker discrete logarithm assumption suffices) [17].

**Cramer-Shoup encryption using labels.** The Cramer-Shoup public-key encryption scheme [16] was the first known example of a practical encryption scheme with a rigorous proof of security against chosen-ciphertext attacks under standard cryptographic assumptions. Our protocol relies on a modification of the Cramer-Shoup scheme, described here for convenience. For completeness, we also provide a definition of security against chosen-ciphertext attacks incorporating the notion of *labels* (see [47]). We rely on this definition in the proof of security for our protocol.

Before continuing, we emphasize that our protocol does *not* use the (modified) Cramer-Shoup scheme for *encryption* per se. Indeed, in our application no party publishes a public key or holds any secret key associated with the scheme, and "decryption" is never performed during execution of our protocol. Our proof of security, however, does rely on the fact that decryption is possible. Gennaro and Lindell [24] have since shown that this feature is not essential, and that non-malleable *commitment* [22] suffices.[7] We first define the semantics of public-key encryption with labels.

**Definition 5** A public-key encryption scheme (supporting labels) is a tuple of PPT algorithms $(\mathsf{KeyGen}, \mathcal{E}, \mathcal{D})$ such that:

- The key generation algorithm KeyGen takes as input a security parameter $1^k$ and returns a public key $pk$ and a secret key $sk$.
- The encryption algorithm $\mathcal{E}$ takes as input a public key $pk$, a label LABEL, and a message $m$. It returns a ciphertext $C$. We write this as $C \leftarrow \mathcal{E}_{pk}(\text{LABEL}, m)$.
- The decryption algorithm $\mathcal{D}$ takes as input a secret key $sk$, a label LABEL, and a ciphertext $C$. It returns a message $m$ or a distinguished symbol $\perp$. We write this as $m = \mathcal{D}_{sk}(\text{LABEL}, C)$.

We require that for all $k$, all $pk, sk$ output by $\mathsf{KeyGen}(1^k)$, any LABEL, all $m$ in the (implicit) message space, and any $C$ output by $\mathcal{E}_{pk}(\text{LABEL}, m)$ we have $\mathcal{D}_{sk}(\text{LABEL}, C) = m$. $\diamondsuit$

Our definition of security against chosen-ciphertext attacks is equivalent to the standard definition except for our inclusion of labels, which follows [47]. In the following, we define a left-or-right encryption oracle $\mathcal{E}_{pk,b}(\cdot,\cdot,\cdot)$ (where $b \in \{0,1\}$) as follows:

$$\mathcal{E}_{pk,b}(\text{LABEL}, m_0, m_1) \stackrel{\text{def}}{=} \mathcal{E}_{pk}(\text{LABEL}, m_b).$$

**Definition 6** A public-key encryption scheme $(\mathsf{KeyGen}, \mathcal{E}, \mathcal{D})$ is secure against adaptive chosen-ciphertext attacks if the following is negligible for all PPT algorithms $A$:

$$\left| 2 \cdot \Pr[(pk, sk) \leftarrow \mathsf{KeyGen}(1^k); b \leftarrow \{0,1\} : A^{\mathcal{E}_{pk,b}(\cdot,\cdot,\cdot), \mathcal{D}_{sk}(\cdot,\cdot)}(1^k, pk) = b] - 1 \right|,$$

where $A$'s queries are restricted as follows: if $A$ makes a query $\mathcal{E}_{pk,b}(\text{LABEL}, m_0, m_1)$ then $m_0, m_1$ must be in the (implicit) message space with $|m_0| = |m_1|$; furthermore, if $A$ receives ciphertext $C$ in response to this query, then $A$ cannot later query $\mathcal{D}_{sk}(\text{LABEL}, C)$. $\diamondsuit$

We now describe our modification of the Cramer-Shoup encryption scheme [16]. On input $1^k$, the key generation algorithm KeyGen first runs $\mathcal{G}(1^k)$ to generate a group $\mathbb{G}$ (as

[7]Security of the Cramer-Shoup scheme against adaptive chosen-ciphertext attacks implies that it is a non-malleable commitment scheme in the common reference string model [18].

described earlier in the context of the DDH assumption); it also selects random generators $g_1, g_2 \leftarrow \bar{\mathbb{G}}$. Next, it selects three pairs $(z_1, z_2), (x_1, x_2), (y_1, y_2)$ at random from the set $\{(a, b) \in \mathbb{Z}_q \times \mathbb{Z}_q \mid g_1^a g_2^b \neq 1\}$, and sets $h = g_1^{z_1} g_2^{z_2}$, $c = g_1^{x_1} g_2^{x_2}$, and $d = g_1^{y_1} g_2^{y_2}$. Finally, it runs $\mathsf{CRHF}(1^k)$ to generate a hash function $H$, where $\mathsf{CRHF}$ is a collision-resistant hash family as described above. The public key consists of $\mathbb{G}, g_1, g_2, h, c, d, H$; the secret key is $z_1, z_2, x_1, x_2, y_1, y_2$; and the message space is $\mathbb{G}$.

To encrypt a message $m \in \mathbb{G}$ using label $\textsc{label}$, the sender chooses a random $r \in \mathbb{Z}_q$ and computes $u_1 = g_1^r$, $u_2 = g_2^r$, and $e = h^r m$. It then sets $\alpha = H(\textsc{label}|u_1|u_2|e)$ and outputs the ciphertext $\langle u_1|u_2|e|(cd^\alpha)^r\rangle$. To decrypt a ciphertext $\langle u_1|u_2|e|v\rangle$ using label $\textsc{label}$, the receiver first verifies that all components of the ciphertext lie in $\mathbb{G}$. If not, it outputs $\perp$; otherwise, it then computes $\alpha = H(\textsc{label}|u_1|u_2|e)$ and checks whether $u_1^{x_1+\alpha y_1} u_2^{x_2+\alpha y_2} \stackrel{?}{=} v$. If not, the decryption algorithm outputs $\perp$; otherwise, it outputs $m = e/u_1^{z_1} u_2^{z_2}$. It is not hard to see that decryption (of honestly-generated ciphertexts) always succeeds.

The following theorem follows immediately from the results of [16] (see [33]):

**Theorem 1** *Assuming (1) the DDH problem is hard for $\mathcal{G}$ and (2) $\mathsf{CRHF}$ is a collision-resistant hash family, the above scheme is secure against adaptive chosen-ciphertext attacks.*

## 3 A Provably-Secure Protocol for Password-Only AKE

### 3.1 Description of the Protocol

A high-level depiction of the protocol is given in Figure 1, and a more detailed description follows. A completely formal specification of the protocol appears in Section 3.2, where we give a proof of security for the protocol in the "basic" adversarial model of Section 2.2. In Section 3.3 we show that the protocol does *not* achieve forward secrecy if it is modified in a seemingly-innocuous way (this is meant to motivate our definition of forward secrecy, as well as to illustrate the importance of rigorous proofs of security). Then, we prove that the protocol as formally defined *does* achieve forward secrecy.

The protocol as described here is different from what appears in previous versions of this work [33, 35, 36], as we have introduced small modifications in order to simplify the proof of security (the proofs of security given previously [33, 35, 36] are correct). These modifications do not significantly affect the efficiency of the protocol, nor do they necessitate additional assumptions: our proof still requires only the decisional Diffie-Hellman assumption. We point out in particular one modification: we now assume that $H$ is collision resistant, whereas in [33, 35, 36] (and also in [24]) it was only assumed that $H$ is universal one-way. The protocol shown here uses $H$ to hash different information than in [33, 35, 36], and this change seems to require collision resistance of $H$. We have introduced this change because it substantially simplifies the proof of security without having any significant drawbacks: collision-resistant hash functions can be constructed based on the decisional Diffie-Hellman assumption that we already require, and in practice one would likely use a cryptographic hash function such as SHA-1 which is assumed to be collision resistant anyway.

**Initialization.** For a given security parameter $k$, the public parameters will contain a group $\mathbb{G}$ (written multiplicatively) having prime order $q$ with $|q| = k$; we assume the hardness of the DDH problem in $\mathbb{G}$, and one suggestion for $\mathbb{G}$ is given in Section 2.4. Additionally, the

$$
\begin{array}{rcl}
\multicolumn{3}{c}{\text{Public: } \mathbb{G};\ g_1, g_2, h, c, d \in \bar{\mathbb{G}};\ H : \{0,1\}^* \to \mathbb{Z}_q} \\
\text{Client} & & \text{Server} \\
(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^k) & & \\
r_1 \leftarrow \mathbb{Z}_q & & \\
A := g_1^{r_1};\ B := g_2^{r_1} & & \\
C := h^{r_1} \cdot pw_C & & \\
\alpha := H(\mathit{PIDs}\,|\mathsf{VK}|A|B|C) & & \\
D := (cd^{\alpha})^{r_1} & \xrightarrow{\mathit{Client}\,|\,\mathsf{VK}\,|\,A\,|\,B\,|\,C\,|\,D} & \\
 & & x_2, y_2, z_2, w_2, r_2 \leftarrow \mathbb{Z}_q \\
 & & \alpha' := H(\mathit{PIDs}\,|\mathsf{VK}|A|B|C) \\
 & & E := g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2} \\
 & & F := g_1^{r_2};\ G := g_2^{r_2} \\
 & & I := h^{r_2} \cdot pw_C \\
 & & \beta := H(\mathsf{msg}_1|\mathit{Server}\,|E|F|G|I) \\
 & \xleftarrow{\mathit{Server}\,|\,E\,|\,F\,|\,G\,|\,I\,|\,J} & J := (cd^{\beta})^{r_2} \\
x_1, y_1, z_1, w_1 \leftarrow \mathbb{Z}_q & & \\
\beta' := H(\mathsf{msg}_1|\mathit{Server}\,|E|F|G|I) & & \\
K := g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta'})^{w_1} & & \\
\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{msg}_1|\mathsf{msg}_2|K) & \xrightarrow{K\,|\,\mathsf{Sig}} & \\
 & & \text{if } \mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{msg}_1|\mathsf{msg}_2|K,\ \mathsf{Sig}) \\
 & & \quad C' := C/pw_C \\
I' := I/pw_C & & \quad \mathsf{sk}_S := A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} K^{r_2} \\
\mathsf{sk}_C := E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1} & & \text{else } \mathsf{sk}_S := \text{NULL}
\end{array}
$$

Figure 1: A protocol for password-only authenticated key-exchange. The client name is *Client*, the server name is *Server*, and we let $\mathit{PIDs} :=$ "*Client*|*Server*". The first and second messages of the protocol are called $\mathsf{msg}_1$ and $\mathsf{msg}_2$, respectively.

parameters include random generators $g_1, g_2, h, c, d \in \bar{\mathbb{G}}$ and a hash function $H : \{0,1\}^* \to \mathbb{Z}_q$ chosen at random from a collision-resistant hash family.

As part of the initialization, a password $pw_C$ is chosen randomly for each client $C$, and $pw_C$ is stored by each server (cf. Section 2.2). We assume that all passwords lie in $\mathbb{G}$ or can be mapped in a one-to-one fashion to $\mathbb{G}$; this is typically easy: for example, if passwords are represented as integers less than $q$ then the password $pw$ can be mapped to $g_1^{pw} \in \mathbb{G}$. For ease of exposition, we will simply assume that passwords lie in the set $\{g_1^1, \ldots, g_1^N\}$ where $N$ represents the size of the dictionary from which passwords are chosen. (We also implicitly assume that $N < q$, which will certainly be true in practice.)

**Protocol execution.** When a client $\mathit{Client} \in \mathsf{Client}$ with password $pw_C$ wants to connect to a server $\mathit{Server} \in \mathsf{Server}$, the client computes a Cramer-Shoup "encryption" of $pw_C$, as described in Section 2.4, using a particular value for the label. In more detail, let *PIDs* denote the string "*Client*|*Server*". The client begins by running a key-generation algorithm for a one-time signature scheme, giving $\mathsf{VK}$ and $\mathsf{SK}$. The client chooses random $r_1 \in \mathbb{Z}_q$ and computes $A = g_1^{r_1}$, $B = g_2^{r_1}$, and $C = h^{r_1} \cdot pw_C$. The client then computes $\alpha = H(\mathit{PIDs}|\mathsf{VK}|A|B|C)$ and sets $D = (cd^{\alpha})^{r_1}$. The client sends

$$\mathsf{msg}_1 \stackrel{\text{def}}{=} \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$$

to the server as the first message of the protocol. Note that this corresponds roughly to an "encryption" of $pw_C$ using the label $\mathit{PIDs}|\mathsf{VK}$.

Upon receiving the message $\mathsf{msg}_1$ (as above), the server computes a Cramer-Shoup "encryption" of $pw_C$ (where $pw_C$ is the password corresponding to the client named in the incoming message) using a particular value for the label. In more detail, the server first chooses random $x_2, y_2, z_2, w_2 \in \mathbb{Z}_q$, computes $\alpha' = H(\mathit{PIDs}|\mathsf{VK}|A|B|C)$, and sets $E = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2}$. Additionally, a random $r_2 \in \mathbb{Z}_q$ is chosen and the server computes $F = g_1^{r_2}$, $G = g_2^{r_2}$, and $I = h^{r_2} \cdot pw_C$. The server then computes $\beta = H(\mathsf{msg}_1|\mathit{Server}|E|F|G|I)$ and sets $J = (cd^{\beta})^{r_2}$. The server sends

$$\mathsf{msg}_2 \stackrel{\text{def}}{=} \langle \mathit{Server}|E|F|G|I|J\rangle$$

to the client as the second message of the protocol. Note that this process corresponds roughly to an "encryption" of $pw_C$ using the label $\mathsf{msg}_1|\mathit{Server}|E$.

Upon receiving the message $\mathsf{msg}_2$, the client chooses random $x_1, y_1, z_1, w_1 \in \mathbb{Z}_q$, computes $\beta' = H(\mathsf{msg}_1|\mathit{Server}|E|F|G|I)$, and sets $K = g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta'})^{w_1}$. The client then signs $\mathsf{msg}_1|\mathsf{msg}_2|K$ using the secret key $\mathsf{SK}$ that it generated in the first step of the protocol. The value $K$ and the resulting signature are sent as the final message of the protocol. At this point, the client accepts and determines the session key by first computing $I' = I/pw_C$ and then setting $\mathsf{sk}_C = E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1}$.

Upon receiving the message $K|\mathsf{Sig}$, the server checks that $\mathsf{Sig}$ is a valid signature of $\mathsf{msg}_1|\mathsf{msg}_2|K$ under $\mathsf{VK}$. If so, the server accepts and determines the session key by first computing $C' = C/pw_{S,C}$ and then setting $\mathsf{sk}_S = A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} K^{r_2}$. Otherwise, the server terminates without accepting and the session key remains NULL.

Although omitted in the above description, we assume that the client and server always check that incoming messages are well-formed. In particular, when the server receives the first message it verifies that $\mathit{Client} \in \mathsf{Client}$ and that $A, B, C, D \in \mathbb{G}$ (recall that membership in $\mathbb{G}$ can be efficiently verified). When the client receives the second message, it verifies that the server name included in the message is indeed the name of the server to whom the client desired to connect, and that $E, F, G, I, J \in \mathbb{G}$. Finally, when the server receives the last message it verifies that $K \in \mathbb{G}$ in addition to verifying correctness of the signature. If an ill-formed message is ever received, the receiving party terminates immediately without accepting and the session key remains NULL. For further details, see the formal description of the protocol in Figures 3–4, below.

**Correctness.** In an honest execution of the protocol, the client and the server calculate identical session keys. To see this, first note that in an honest execution we have $\alpha = \alpha'$, $\beta = \beta'$, and the same password $pw_C$ is used by both parties. We thus have

$$\begin{aligned} E^{r_1} &= (g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2})^{r_1} \\ &= (g_1^{r_1})^{x_2} (g_2^{r_1})^{y_2} (h^{r_1})^{z_2} ((cd^{\alpha})^{r_1})^{w_2} \\ &= A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} \end{aligned}$$

and

$$\begin{aligned} K^{r_2} &= (g_1^{x_1} g_2^{y_1} h^{z_1} (cd^\beta)^{w_1})^{r_2} \\ &= (g_1^{r_2})^{x_1} (g_2^{r_2})^{y_1} (h^{r_2})^{z_1} ((cd^\beta)^{r_2})^{w_1} \\ &= F^{x_1} G^{y_1} (I')^{z_1} J^{w_1}. \end{aligned}$$

Therefore:

$$\mathsf{sk}_C = E^{r_1}(F^{x_1} G^{y_1} (I')^{z_1} J^{w_1}) = (A^{x_2} B^{y_2} (C')^{z_2} D^{w_2}) K^{r_2} = \mathsf{sk}_S$$

and the session keys are equal.

**Achieving explicit authentication.** As described, the protocol does not achieve explicit mutual authentication (that is, a party does not know whether its intended partner has successfully computed a matching session key). However, it is easy to add explicit authentication to the protocol using standard techniques; see, e.g., [2].

**Efficiency considerations.** Some remarks about the computational efficiency of the above scheme are in order. First, efficiency of the signature computation can be improved using an on-line/off-line signature scheme [23], where the off-line computation is done while the client is waiting for the server to respond. Also, the data being signed (namely, $\mathsf{msg}_1|\mathsf{msg}_2|K$) could clearly be hashed using $H$ before signature computation. By swapping the roles of the client and server (so that the server sends the first message in Figure 1), the server can use long-term public-/private-keys (for a signature scheme which is existentially unforgeable under adaptive chosen-message attacks [26]) and thereby avoid having to run the key-generation algorithm each time the protocol is executed. We stress that in this case *we do not require that the client store the server's long-term public key*; instead, the public key is included in the first message from the server but the protocol will be secure even if an active adversary replaces this key with one of his own choosing.

Algorithms for simultaneous multiple exponentiation [42, Chapter 14] can be used to speed up the computation in the protocol. If this is done, the computation for each user is (roughly) equivalent to 7–8 exponentiations in $\mathbb{G}$; this may be compared to the 2 exponentiations per user required in standard Diffie-Hellman key exchange [19], which provides no authentication at all.

Further efficiency improvements (that introduce modifications to the protocol) are discussed in [34]; we refer the reader there for additional discussion.

## 3.2 Proof of Security

We first provide a formal specification of the protocol by specifying the initialization phase and the oracles to which the adversary has access. During the initialization phase for security parameter $k$ (cf. Figure 2), algorithm $\mathsf{Initialize}$ first runs an algorithm $\mathcal{G}$ to generate a group $\mathbb{G}$ of prime order $q$ with $|q| = k$. Next, generators $g_1, g_2, h, c, d \in \bar{\mathbb{G}}$ are selected at random, and a hash function $H$ is chosen from a collision-resistant hash family $\mathsf{CRHF}$. Furthermore, the sets $\mathsf{Client}$ and $\mathsf{Server}$ are determined using some (arbitrary) algorithm $\mathsf{UserGen}$. Passwords for each client are chosen at random as discussed in the previous section; the passwords for each client are then stored at each server.

A formal specification of the $\mathsf{Execute}$, $\mathsf{Reveal}$, and $\mathsf{Test}$ oracles appears in Figure 3. The description of the $\mathsf{Execute}$ oracle matches the high-level protocol description of Figure 1, but additional details (for example, the updating of state information) are included. We let $\mathsf{status}_U^i$ denote the vector of values $\langle \mathsf{sid}_U^i, \mathsf{pid}_U^i, \mathsf{acc}_U^i, \mathsf{term}_U^i \rangle$ associated with instance $\Pi_U^i$. A formal specification of the $\mathsf{Send}$ oracle appears in Figure 4. Although the model technically has only one type of $\mathsf{Send}$ oracle (see Section 2.2), the adversary's queries to this oracle can be viewed as queries to four different oracles $\mathsf{Send}_0, \ldots, \mathsf{Send}_3$ representing the four different types of messages which may be sent as part of the protocol (this includes the three message types shown in Figure 1 as well as an "initiate" message). The third argument to each oracle is denoted by $\mathsf{msg\text{-}in}$. In the proof that follows, we will also sometimes refer to $\mathsf{msg}_1$, $\mathsf{msg}_2$, $\mathsf{msg}_3$, where these refer to messages having the correct format for rounds 1, 2, and 3, respectively. We will also use these to denote messages input to the $\mathsf{Send}_1$, $\mathsf{Send}_2$, or $\mathsf{Send}_3$ oracles, respectively, or output by the $\mathsf{Send}_0$, $\mathsf{Send}_1$, or $\mathsf{Send}_2$ oracles, respectively.

**Theorem 2** *Assuming (1) the DDH problem is hard for $\mathcal{G}$; (2)* $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ *is a secure one-time signature scheme; and (3)* $\mathsf{CRHF}$ *is a collision-resistant hash family, the protocol of Figure 1 is a secure protocol for password-only authenticated key exchange.*

Since one-time signature schemes as well as collision-resistant hash functions may be constructed based on the DDH assumption (cf. Section 2.4), we have the following corollary.

**Corollary 1** *Under the DDH assumption, there exists a secure protocol for password-only authenticated key exchange.*

Before giving the formal details, we describe the high-level structure of the proof of Theorem 2. Let $P_0$ denote the "real-world" experiment where an adversary attacks the actual protocol. We introduce a sequence of transformations to this experiment, and bound the effect of each transformation on the adversary's advantage. We then bound the adversary's advantage in the final experiment; this yields a bound on the adversary's advantage when attacking the original protocol.

In experiment $P_2$ (obtained by a sequence of modifications to $P_0$), all session keys computed in response to an $\mathsf{Execute}$ query are chosen uniformly at random and the password is not used at all; $\mathsf{Send}$ queries are treated exactly as in $P_0$. It is not too difficult to see that the adversary's advantage cannot change significantly in moving from $P_0$ to $P_2$, since the "core" of an honest execution of the protocol is based on the Diffie-Hellman key-exchange protocol (and $\mathsf{Execute}$ queries correspond to passive eavesdropping on a protocol execution), and the password is anyway "encrypted" using Cramer-Shoup encryption.

In experiment $P_3$, we (informally) consider the adversary to have succeeded as soon as it interacts with a client or server using the "correct" value of the password. (Thus, the adversary may now succeed even without guessing the correct value of $b$.) This can only increase the advantage of the adversary.

Experiment $P_5$ (preceded by an intermediate experiment $P_4$ for technical reasons) is really the crux of the proof. Roughly speaking, we now define the experiment so that whenever the adversary interacts with a server using an *incorrect* value of the password the server chooses a session key uniformly at random. We then prove that this has *no* effect on the adversary's advantage, in an information-theoretic sense; expressed differently, this means that the session key computed by a server in experiment $P_3$ when the adversary interacts

```
Initialize($1^k$) —
    $\mathbb{G} \leftarrow \mathcal{G}(1^k)$
    $g_1, g_2, h, c, d \leftarrow \bar{\mathbb{G}}$
    $H \leftarrow \mathsf{CRHF}(1^k)$
    $(\mathsf{Client}, \mathsf{Server}) \leftarrow \mathsf{UserGen}(1^k)$
    for each $C \in \mathsf{Client}$
        $pw'_C \leftarrow \{1, \ldots, N\}$
        $pw_C := g_1^{pw'_C}$
        for each $S \in \mathsf{Server}$
            $pw_{S,C} := pw_C$
    return $\mathsf{Client}, \mathsf{Server}, \mathbb{G}, g_1, g_2, h, c, d, H$
```

Figure 2: Specification of protocol initialization.

```
Execute($Client, i, Server, j$) —
    if $\left(Client \notin \mathsf{Client}$ or $Server \notin \mathsf{Server}$ or $\mathsf{used}^i_{Client}$ or $\mathsf{used}^j_{Server}\right)$
        return $\perp$
    $\mathsf{used}^i_{Client} := \text{TRUE}$; $\mathsf{used}^j_{Server} := \text{TRUE}$
    $PIDs := Client|Server$;  $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^k)$
    $x_1, x_2, y_1, y_2, z_1, z_2, w_1, w_2, r_1, r_2 \leftarrow \mathbb{Z}_q$
    $A := g_1^{r_1}$;  $B := g_2^{r_1}$;  $C := h^{r_1} \cdot pw_{Client}$;  $\alpha := H(PIDs|\mathsf{VK}|A|B|C)$
    $D := (cd^{\alpha})^{r_1}$;  $\mathsf{msg}_1 := \langle Client|\mathsf{VK}|A|B|C|D\rangle$
    $\mathsf{status}^i_{Client,1} := \langle \text{NULL}, Server, \text{FALSE}, \text{FALSE}\rangle$

    $F := g_1^{r_2}$;  $G := g_2^{r_2}$;  $I := h^{r_2} \cdot pw_{Server,Client}$;  $E := g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2}$
    $\beta := H(\mathsf{msg}_1|Server|E|F|G|I)$;  $J := (cd^{\beta})^{r_2}$
    $\mathsf{msg}_2 := \langle Server|E|F|G|I|J\rangle$
    $\mathsf{status}^j_{Server,1} := \langle \text{NULL}, Client, \text{FALSE}, \text{FALSE}\rangle$

    $K := g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta})^{w_1}$;  $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{msg}_1|\mathsf{msg}_2|K)$
    $\mathsf{msg}_3 := \langle K|\mathsf{Sig}\rangle$

    $\mathsf{sid}^i_{Client} := \mathsf{sid}^j_{Server} := \langle \mathsf{msg}_1|\mathsf{msg}_2|\mathsf{msg}_3\rangle$
    $\mathsf{pid}^i_{Client} := Server$;  $\mathsf{pid}^j_{Server} := Client$
    $\mathsf{acc}^i_{Client} := \mathsf{term}^i_{Client} := \mathsf{acc}^j_{Server} := \mathsf{term}^j_{Server} := \text{TRUE}$
    $\mathsf{sk}^i_{Client} := E^{r_1} F^{x_1} G^{y_1} (I/pw_{Client})^{z_1} J^{w_1}$
    $\mathsf{sk}^j_{Server} := A^{x_2} B^{y_2} (C/pw_{Server,Client})^{z_2} D^{w_2} K^{r_2}$
    return $\mathsf{msg}_1, \mathsf{msg}_2, \mathsf{msg}_3,$
            $\mathsf{status}^i_{Client,1}, \mathsf{status}^j_{Server,1}, \mathsf{status}^i_{Client}, \mathsf{status}^j_{Server}$

Reveal($U, i$) —
    return $\mathsf{sk}^i_U$

Test($U, i$) —
    $b \leftarrow \{0, 1\}$;  $\mathsf{sk}' \leftarrow \mathbb{G}$
    if $b = 0$ return $\mathsf{sk}'$ else return $\mathsf{sk}^i_U$
```

Figure 3: Specification of the Execute, Reveal, and Test oracles.

```
$\mathsf{Send}_0(\mathit{Client}, i, \mathit{Server})$ —
    if $(\mathit{Client} \notin \mathsf{Client}$ or $\mathit{Server} \notin \mathsf{Server}$ or $\mathsf{used}^i_{\mathit{Client}})$ return $\perp$
    $\mathsf{used}^i_{\mathit{Client}} := \text{TRUE}$;  $\mathit{PIDs} := \mathit{Client}|\mathit{Server}$;  $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^k)$;  $r \leftarrow \mathbb{Z}_q$
    $A := g_1^r$;  $B := g_2^r$;  $C := h^r \cdot pw_{\mathit{Client}}$;  $\alpha := H(\mathit{PIDs}|\mathsf{VK}|A|B|C)$
    $D := (cd^\alpha)^r$;  $\mathsf{msg\text{-}out} := \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$
    $\mathsf{status}^i_{\mathit{Client}} := \langle \text{NULL}, \mathit{Server}, \text{FALSE}, \text{FALSE}\rangle$
    $\mathsf{state}^i_{\mathit{Client}} := \langle \mathsf{SK}, r, \mathsf{msg\text{-}out}\rangle$
    return $\mathsf{msg\text{-}out}, \mathsf{status}^i_{\mathit{Client}}$

$\mathsf{Send}_1(\mathit{Server}, j, \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle)$ —
    if $\left(\mathit{Server} \notin \mathsf{Server}\text{ or }\mathsf{used}^j_{\mathit{Server}}\right)$ return $\perp$
    $\mathsf{used}^j_{\mathit{Server}} := \text{TRUE}$;  $\mathit{PIDs} := \mathit{Client}|\mathit{Server}$
    if $A, B, C, D \notin \mathbb{G}$ or $\mathit{Client} \notin \mathsf{Client}$
        $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$;  return $\mathsf{status}^j_{\mathit{Server}}$
    $x, y, z, w, r \leftarrow \mathbb{Z}_q$;  $\alpha := H(\mathit{PIDs}|\mathsf{VK}|A|B|C)$
    $F := g_1^r$;  $G := g_2^r$;  $I := h^r \cdot pw_{\mathit{Server},\mathit{Client}}$;  $E := g_1^x g_2^y h^z (cd^\alpha)^w$
    $\beta := H(\mathsf{msg\text{-}in}\,|\mathit{Server}|E|F|G|I)$;  $J := (cd^\beta)^r$;  $\mathsf{msg\text{-}out} := \langle \mathit{Server}|E|F|G|I|J\rangle$
    $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \mathit{Client}, \text{FALSE}, \text{FALSE}\rangle$
    $\mathsf{state}^j_{\mathit{Server}} := \langle \mathsf{msg\text{-}in}, x, y, z, w, r, \mathsf{msg\text{-}out}, pw_{\mathit{Server},\mathit{Client}}\rangle$
    return $\mathsf{msg\text{-}out}, \mathsf{status}^j_{\mathit{Server}}$

$\mathsf{Send}_2(\mathit{Client}, i, \langle \mathit{Server}|E|F|G|I|J\rangle)$ —
    if $(\mathit{Client} \notin \mathsf{Client}$ or $\neg\mathsf{used}^i_{\mathit{Client}}$ or $\mathsf{term}^i_{\mathit{Client}})$ return $\perp$
    if $\mathit{Server} \neq \mathsf{pid}^i_{\mathit{Client}}$ or $E, F, G, I, J \notin \mathbb{G}$
        $\mathsf{status}^i_{\mathit{Client}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$;  return $\mathsf{status}^i_{\mathit{Client}}$
    $\langle \mathsf{SK}, r, \mathsf{first\text{-}msg\text{-}out}\rangle := \mathsf{state}^i_{\mathit{Client}}$
    $x, y, z, w \leftarrow \mathbb{Z}_q$;  $\beta := H(\mathsf{first\text{-}msg\text{-}out}|\mathit{Server}|E|F|G|I)$
    $K := g_1^x g_2^y h^z (cd^\beta)^w$;  $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{first\text{-}msg\text{-}out}|\mathsf{msg\text{-}in}|K)$;
    $\mathsf{msg\text{-}out} := \langle K|\mathsf{Sig}\rangle$
    $\mathsf{sid}^i_{\mathit{Client}} := \langle \mathsf{first\text{-}msg\text{-}out}|\mathsf{msg\text{-}in}|\mathsf{msg\text{-}out}\rangle$
    $\mathsf{acc}^i_{\mathit{Client}} := \mathsf{term}^i_{\mathit{Client}} := \text{TRUE}$
    $\mathsf{sk}^i_{\mathit{Client}} := E^r F^x G^y (I/pw_{\mathit{Client}})^z J^w$
    return $\mathsf{msg\text{-}out}, \mathsf{status}^i_{\mathit{Client}}$

$\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$ —
    if $\left(\mathit{Server} \notin \mathsf{Server}\text{ or }\neg\mathsf{used}^j_{\mathit{Server}}\text{ or }\mathsf{term}^j_{\mathit{Server}}\right)$ return $\perp$
    $\langle \mathsf{first\text{-}msg\text{-}in}, x, y, z, w, r, \mathsf{first\text{-}msg\text{-}out}, pw\rangle := \mathsf{state}^j_{\mathit{Server}}$
    $\langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle := \mathsf{first\text{-}msg\text{-}in}$
    if $K \notin \mathbb{G}$ or $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{first\text{-}msg\text{-}in}|\mathsf{first\text{-}msg\text{-}out}|K, \mathsf{Sig}) \neq 1$
        $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$;  return $\mathsf{status}^j_{\mathit{Server}}$
    $\mathsf{sid}^j_{\mathit{Server}} := \langle \mathsf{first\text{-}msg\text{-}in}\,|\mathsf{first\text{-}msg\text{-}out}|\mathsf{msg\text{-}in}\rangle$
    $\mathsf{acc}^j_{\mathit{Server}} := \mathsf{term}^j_{\mathit{Server}} := \text{TRUE}$
    $\mathsf{sk}^j_{\mathit{Server}} := A^x B^y (C/pw)^z D^w K^r$
    return $\mathsf{status}^j_{\mathit{Server}}$
```

Figure 4: Specification of the Send oracle (msg-in denotes the third argument to the oracle).

with a server using an incorrect value of the password is already uniformly distributed from the point of view of the adversary,

In experiment $P_6$, we now have the server compute the first message of the protocol (in response to a $\mathsf{Send}$ query) independently of the actual password. Relying on the security of the Cramer-Shoup encryption scheme, we show that this cannot significantly affect the adversary's advantage.

Experiments $P_7$ and $P_8$ are analogous to experiments $P_5$ and $P_6$, but focusing on client-instances rather than server-instances.

In experiment $P_8$, we notice that the adversary's view is independent of any actual passwords except for the fact that the adversary succeeds (as per the modification introduced in experiment $P_3$) as soon as it interacts with a party using a "correct" password; this event therefore occurs with probability at most $Q(k)/N$. Furthermore, if this event does not happen then the bit $b$ is information-theoretically hidden from the adversary (since all session keys are chosen uniformly at random), and so the adversary succeeds with probability exactly half. The overall probability that the adversary succeeds in experiment $P_8$ is thus at most

$$Q(k)/N + \tfrac{1}{2} \cdot (1 - \tfrac{Q(k)}{N}).$$

The analysis of the above sequence of experiments shows that the adversary's success probability in the original experiment is at most negligibly greater than this.

We now give the formal details.

**Proof** Throughout the proof, we will refer to the formal specification of the protocol as it appears in Figures 2–4. We remark that it is always the case that $pw_{Server,Client} = pw_{Client}$ for all $Client \in \mathsf{Client}$ and $Server \in \mathsf{Server}$; furthermore, during execution of the $\mathsf{Send}_3$ oracle it is always the case that $pw = pw_{Client}$. (This will not necessarily be true when considering forward secrecy; see the following section.) Given an adversary $\mathcal{A}$, we imagine a simulator that runs the protocol for $\mathcal{A}$. More precisely, the simulator begins by running algorithm $\mathsf{Initialize}(1^k)$ (which includes choosing passwords for clients) and giving the public output of the algorithm to $\mathcal{A}$. When $\mathcal{A}$ queries an oracle, the simulator responds by executing the appropriate algorithm as in Figures 3 and 4; the simulator also records all state information defined during the course of the experiment. In particular, when the adversary queries the $\mathsf{Test}$ oracle, the simulator chooses (and records) the random bit $b$. When the adversary completes its execution and outputs a bit $b'$, the simulator can tell whether the adversary succeeds by checking whether (1) a single $\mathsf{Test}$ query was made, for some instance $\Pi_U^i$; (2) $\mathsf{acc}_U^i$ was true at the time of the $\mathsf{Test}$ query; (3) instance $\Pi_U^i$ is fresh; and (4) $b' = b$. Success of the adversary is denoted by event $\mathsf{Succ}$.

For any experiment $P$ we define $\mathsf{Adv}_{\mathcal{A},P}(k) \overset{\text{def}}{=} 2 \cdot \Pr_{\mathcal{A},P}[\mathsf{Succ}] - 1$, where $\Pr_{\mathcal{A},P}[\cdot]$ denotes the probability of an event when the simulator interacts with the adversary $\mathcal{A}$ in accordance with experiment $P$. We refer to the real execution of the experiment, as described above, as $P_0$. We will introduce a sequence of transformations to the original experiment and bound the effect of each transformation on the adversary's advantage. We then bound the adversary's advantage in the final experiment; this immediately yields a bound on the adversary's advantage in the original experiment.

We begin with some terminology that will be used throughout the proof. A given $\mathsf{msg}_1$ is called *oracle-generated* if it was output by the simulator in response to some oracle query (whether a $\mathsf{Send}_0$ or $\mathsf{Execute}$ query). This message is said to be *adversarially-generated* otherwise. These notions are defined analogously for a given $\mathsf{msg}_2$. A verification key contained in an oracle-generated $\mathsf{msg}_1$ is called an *oracle-generated verification key.*

**Experiment $P_0'$**: In experiment $P_0'$, the simulator interacts with the adversary as before except that the adversary does *not* succeed, and the experiment is aborted, if any of the following occur:

1. At any point, an oracle-generated verification key is used more than once. In particular, the experiment is aborted if an oracle-generated $\mathsf{msg}_1$ is ever repeated.
2. At any point during the experiment, an oracle-generated $\mathsf{msg}_2$ is repeated.
3. At any point, the adversary forges a new, valid message/signature pair with respect to any oracle-generated verification key.
4. At any point during the experiment, a collision occurs in the hash function $H$ (regardless of whether this is due to a direct action of the adversary, or whether this occurs during the course of the simulator's response to an oracle query).

It is immediate that event 2 occurs with negligible probability. It is also straightforward to show that events 1, 3, and 4 occur with negligible probability assuming the security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ as a one-time signature scheme and the security of $\mathsf{CRHF}$ as a collision-resistant hash family. For completeness, we briefly sketch the proof that events 1 and 3 occur with only negligible probability (based on security of $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$); the proof that event 4 occurs with negligible probability (when $\mathsf{CRHF}$ is collision-resistant) is even easier.

Consider the following adversary $\mathcal{F}$ who is given verification key $\mathsf{VK}$ generated by $\mathsf{Gen}$ and access to a signing oracle. Let $p(k)$ be a polynomial upper bound on the number of oracle queries made by $\mathcal{A}$. This adversary $\mathcal{F}$ chooses a random value $i \leftarrow \{1, \ldots, p(k)\}$ and then simulates experiment $P_0$ for $\mathcal{A}$ in the natural way: the entire experiment is simulated by $\mathcal{F}$ internally except for the following:

- The $i$th oracle query of $\mathcal{A}$ that involves running $\mathsf{Gen}$ (note this may be either an $\mathsf{Execute}$ query or a $\mathsf{Send}_0$ query) is answered using verification key $\mathsf{VK}$.
- If/when a signature is needed for the instance corresponding to the oracle query just described, $\mathcal{F}$ obtains this signature from its oracle.

It is easy to see that this results in a perfect simulation of $P_0$ for $\mathcal{A}$.

We claim that if either of events 1 or 3 occur then $\mathcal{F}$ can forge a signature with noticeable probability. If event 1 occurs then with probability at least $2/p(k)$ it is the verification key $\mathsf{VK}$ that is used more than once. In this case $\mathcal{F}$ has generated (on its own) a secret key corresponding to $\mathsf{VK}$ and so can forge a signature on any message of its choice. If event 3 occurs then with probability at least $1/p(k)$ adversary $\mathcal{A}$ has forged a signature with respect to $\mathsf{VK}$, in which case $\mathcal{F}$ can output this as its own forgery. But if $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is a secure one-time signature scheme then $\mathcal{F}$ cannot output a forgery except with negligible probability. This completes the proof that events 1 or 3 occur with negligible probability.

Putting everything together, we see that $\left|\mathsf{Adv}_{\mathcal{A},P_0}(k) - \mathsf{Adv}_{\mathcal{A},P_0'}(k)\right|$ is negligible.

**Experiment** $P_1$: In experiment $P_1$, the simulator interacts with the adversary as in $P_0'$ except that the adversary's queries to the $\mathsf{Execute}$ oracle are handled differently: for each $\mathsf{Execute}$ query the values $C$ and $I$ are computed as $C := h^{r_1} \cdot g_1^{N+1}$ and $I := h^{r_2} \cdot g_1^{N+1}$; note that $g_1^{N+1}$ is not a valid password. Furthermore, the session keys are computed as

$$\mathsf{sk}^i_{Client} := \mathsf{sk}^j_{Server} := A^{x_2} B^{y_2} (C/pw_{Client})^{z_2} D^{w_2} F^{x_1} G^{y_1} (I/pw_{Client})^{z_1} J^{w_1}. \tag{1}$$

The following bounds the effect this transformation can have on the adversary's advantage.

**Claim 1** *Under the DDH assumption,* $|\mathsf{Adv}_{\mathcal{A},P_0'}(k) - \mathsf{Adv}_{\mathcal{A},P_1}(k)|$ *is negligible.*

The proof of the claim relies on the semantic security of the (modified) Cramer-Shoup encryption scheme (which is implied by its security against chosen-ciphertext attacks).[8] We show that the simulator can use $\mathcal{A}$ as a subroutine to distinguish encryptions of the correct client passwords from encryptions of $g_1^{N+1}$. The simulator is given a public key $pk = \langle \mathbb{G}, g_1, g_2, h, c, d, H\rangle$ for an instance of the Cramer-Shoup scheme and may repeatedly query an encryption oracle $\mathcal{E}_{pk,\tilde{b}}(\cdot,\cdot,\cdot)$ where $\tilde{b}$ is a randomly-chosen bit (unknown to the simulator). The advantage of the simulator in attacking the encryption scheme is the absolute value of the difference between the probability the simulator outputs 1 when $\tilde{b} = 1$ and the probability the simulator outputs 1 when $\tilde{b} = 0$.

The simulator begins by running the following modified initialization protocol:

```
$\mathsf{Initialize}'(1^k, \mathbb{G}, g_1, g_2, h, c, d, H)$ —
    $(\mathsf{Client}, \mathsf{Server}) \leftarrow \mathsf{UserGen}(1^k)$
    for each $C \in \mathsf{Client}$
        $pw'_C \leftarrow \{1, \ldots, N\}$
        $pw_C := g_1^{pw'_C}$
        for each $S \in \mathsf{Server}$
            $pw_{S,C} := pw_C$
    return $\mathsf{Client}, \mathsf{Server}, \mathbb{G}, g_1, g_2, h, c, d, H$
```

The simulator responds to $\mathsf{Send}$, $\mathsf{Reveal}$, and $\mathsf{Test}$ queries as in experiments $P_0'$ and $P_1$. However, it responds to $\mathsf{Execute}$ queries as shown in Figure 5. In words: each time the simulator responds to an $\mathsf{Execute}$ query it constructs the values $A, B, C, D$ and $F, G, I, J$ using its encryption oracle by querying this oracle using the appropriate label, and the correct password and $g_1^{N+1}$ as its two "messages". When $\mathcal{A}$ terminates, the simulator outputs 1 if and only if $\mathcal{A}$ succeeds.

It is quite easy to see that when $\tilde{b} = 0$ the actions of the $\mathsf{Execute}$ oracle are exactly as in experiment $P_0'$, while if $\tilde{b} = 1$ the actions of the $\mathsf{Execute}$ oracle are exactly as in experiment $P_1$. (In comparing Figures 3 and 5, it will be helpful to recall that when responding to an $\mathsf{Execute}$ query, $pw_{Client}$ is always equal to $pw_{Server,Client}$ and furthermore $\mathsf{sk}^i_{Client}$ is always equal to $\mathsf{sk}^j_{Server}$.) In particular, when $\tilde{b} = 0$ then $C = h^{r_1} \cdot pw_{Client}$ and $I = h^{r_2} \cdot pw_{Client}$ (for $r_1 = \log_{g_1} A$ and $r_2 = \log_{g_1} F$ unknown to the simulator), while if $\tilde{b} = 1$ then $C = h^{r_1} \cdot g_1^{N+1}$ and $I = h^{r_2} \cdot g_1^{N+1}$. One can also verify that when $\tilde{b} = 0$ the

[8]Earlier versions of this work [33] show how to obtain a tighter concrete security reduction in this step by relying directly on the DDH assumption. We have chosen to present a simpler proof here.

```
$\mathsf{Execute}(\mathit{Client}, i, \mathit{Server}, j)$ —
    if $\Big(\mathit{Client} \notin \mathsf{Client}$ or $\mathit{Server} \notin \mathsf{Server}$ or $\mathsf{used}^i_{\mathit{Client}}$ or $\mathsf{used}^j_{\mathit{Server}}\Big)$
        return $\perp$
    $\mathsf{used}^i_{\mathit{Client}} := \text{TRUE}$; $\mathsf{used}^j_{\mathit{Server}} := \text{TRUE}$
    $\mathit{PIDs} := \mathit{Client}|\mathit{Server}$;  $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathsf{Gen}(1^k)$
    $x_1, x_2, y_1, y_2, z_1, z_2, w_1, w_2 \leftarrow \mathbb{Z}_q$
    $\text{LABEL}_1 := \mathit{PIDs}|\mathsf{VK}$;  $\langle A|B|C|D\rangle \leftarrow \mathcal{E}_{pk,\tilde{b}}(\text{LABEL}_1, pw_{\mathit{Client}}, g_1^{N+1})$
    $\mathsf{msg}_1 := \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$
    $\mathsf{status}^i_{\mathit{Client},1} := \langle \text{NULL}, \mathit{Server}, \text{FALSE}, \text{FALSE}\rangle$

    $\alpha := H(\mathit{PIDs}|\mathsf{VK}|A|B|C)$;  $E := g_1^{x_2} g_2^{y_2} h^{z_2} (cd^\alpha)^{w_2}$
    $\text{LABEL}_2 := \mathsf{msg}_1|\mathit{Server}|E$;  $\langle F|G|I|J\rangle \leftarrow \mathcal{E}_{pk,\tilde{b}}(\text{LABEL}_2, pw_{\mathit{Client}}, g_1^{N+1})$
    $\mathsf{msg}_2 := \langle \mathit{Server}|E|F|G|I|J\rangle$
    $\mathsf{status}^j_{\mathit{Server},1} := \langle \text{NULL}, \mathit{Client}, \text{FALSE}, \text{FALSE}\rangle$

    $\beta := H(\mathsf{msg}_1|\mathit{Server}|E|F|G|I)$
    $K := g_1^{x_1} g_2^{y_1} h^{z_1} (cd^\beta)^{w_1}$;  $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{msg}_1|\mathsf{msg}_2|K)$;
    $\mathsf{msg}_3 := \langle K|\mathsf{Sig}\rangle$

    $\mathsf{sid}^i_{\mathit{Client}} := \mathsf{sid}^j_{\mathit{Server}} := \langle \mathsf{msg}_1|\mathsf{msg}_2|\mathsf{msg}_3\rangle$
    $\mathsf{pid}^i_{\mathit{Client}} := \mathit{Server}$;  $\mathsf{pid}^j_{\mathit{Server}} := \mathit{Client}$
    $\mathsf{acc}^i_{\mathit{Client}} := \mathsf{term}^i_{\mathit{Client}} := \mathsf{acc}^j_{\mathit{Server}} := \mathsf{term}^j_{\mathit{Server}} := \text{TRUE}$
    $\mathsf{sk}^i_{\mathit{Client}} := \mathsf{sk}^j_{\mathit{Server}} := A^{x_2} B^{y_2} (C/pw_{\mathit{Client}})^{z_2} D^{w_2} F^{x_1} G^{y_1} (I/pw_{\mathit{Client}})^{z_1} J^{w_1}$
    return $\mathsf{msg}_1, \mathsf{msg}_2, \mathsf{msg}_3$,
            $\mathsf{status}^i_{\mathit{Client},1}, \mathsf{status}^j_{\mathit{Server},1}, \mathsf{status}^i_{\mathit{Client}}, \mathsf{status}^j_{\mathit{Server}}$
```

Figure 5: The modified Execute oracle for the proof of Claim 1.

session keys are computed as in experiment $P_0'$, while if $\tilde{b} = 1$ the session keys are computed as in experiment $P_1$.

The simulator's advantage in attacking the Cramer-Shoup scheme is therefore

$$\left|\Pr_{\mathcal{A},P_0'}[\mathsf{Succ}] - \Pr_{\mathcal{A},P_1}[\mathsf{Succ}]\right|.$$

The claim follows since this is negligible under the DDH assumption. □

**Experiment** $P_2$: In experiment $P_2$, the simulator interacts with the adversary as in $P_1$ except that during queries $\mathsf{Execute}(\mathit{Client}, i, \mathit{Server}, j)$ the session key $\mathsf{sk}^i_{\mathit{Client}}$ is chosen uniformly at random from $\mathbb{G}$; session key $\mathsf{sk}^j_{\mathit{Server}}$ is set equal to $\mathsf{sk}^i_{\mathit{Client}}$.

**Claim 2** $\mathsf{Adv}_{\mathcal{A},P_1}(k) = \mathsf{Adv}_{\mathcal{A},P_2}(k)$.

We will show that the distribution on the view of the adversary is identical in experiments $P_1$ and $P_2$. For any particular Execute query made by the adversary in experiment $P_1$, we may write $C = h^{r_1'} \cdot pw_{\mathit{Client}}$ with $r_1' \neq r_1$ (recall from Figure 3 that $r_1 \overset{\text{def}}{=} \log_{g_1} A = \log_{g_2} B$). Now, for any $\mu, \nu \in \mathbb{G}$ and fixing the random choices for the remainder of experiment $P_1$, the probability over choice of $x_2, y_2, z_2, w_2$ that $E = \mu$ and $\mathsf{sk}^i_{\mathit{Client}} = \nu$ is exactly the probability that

$$\log_{g_1} \mu \quad = \quad x_2 + y_2 \cdot \log_{g_1} g_2 + z_2 \cdot \log_{g_1} h + w_2 \cdot \log_{g_1} (cd^{\alpha}) \tag{2}$$

and

$$\begin{aligned} &\log_{g_1} \nu - \log_{g_1} (F^{x_1} G^{y_1} (I/pw_{Client})^{z_1} J^{w_1}) = \\ &\qquad x_2 \cdot r_1 + y_2 \cdot r_1 \log_{g_1} g_2 + z_2 \cdot r'_1 \log_{g_1} h + w_2 \cdot r_1 \log_{g_1} (cd^{\alpha}), \end{aligned} \tag{3}$$

where we use the fact that $g_1$ is a generator. Viewing Equations (2) and (3) as equations over $\mathbb{Z}_q$ in the variables $x_2, y_2, z_2, w_2$, we see that they are linearly independent and not identically zero since $r'_1 \neq r_1$ (here, we use the fact that $h$ is a generator and hence $\log_{g_1} h \neq 0$). Thus, the desired probability is $1/q^2$. In other words, the value of $\mathsf{sk}^i_{Client}$ is independent of the value of $E$ and hence independent of the remainder of experiment $P_1$. Thus, the adversary's view in experiment $P_1$ is distributed identically to the adversary's view in experiment $P_2$. The claim follows. □

In experiment $P_2$, the adversary's probability of correctly guessing the bit $b$ used by the Test oracle is exactly 1/2 if the Test query is made to a fresh instance that was activated using an Execute query. This is so because session keys for such instances in $P_2$ are chosen at random from $\mathbb{G}$, and hence there is no way to distinguish whether the Test oracle outputs a random session key or the "actual" session key (which is just a random element, anyway). Hence the remainder of the proof concentrates on instances that are invoked via Send queries (which is the more difficult case to consider).

**Experiment** $P_3$: In experiment $P_3$, the simulator first runs the following modified initialization procedure and stores the values $\kappa, \chi_1, \chi_2, \xi_1, \xi_2$ for future use.

```
Initialize(1^k) —
    $\mathbb{G} \leftarrow \mathcal{G}(1^k); g_1, g_2 \leftarrow \mathbb{G}$
    $\kappa \leftarrow \mathbb{Z}_q^*$
    $(\chi_1, \chi_2), (\xi_1, \xi_2) \leftarrow \{(x, y) \in \mathbb{Z}_q \times \mathbb{Z}_q \mid g_1^x g_2^y \neq 1\}$
    $h := g_1^{\kappa};\ \ c := g_1^{\chi_1} g_2^{\chi_2};\ \ d := g_1^{\xi_1} g_2^{\xi_2}$
    $H \leftarrow \mathsf{CRHF}(1^k)$
    $(\mathsf{Client}, \mathsf{Server}) \leftarrow \mathsf{UserGen}(1^k)$
    for each $C \in \mathsf{Client}$
        $pw'_C \leftarrow \{1, \ldots, N\}$
        $pw_C := g_1^{pw'_C}$
        for each $S \in \mathsf{Server}$
            $pw_{S,C} := pw_C$
    return $\mathsf{Client}, \mathsf{Server}, \mathbb{G}, g_1, g_2, h, c, d, H$
```

The simulator's handling of Send oracle queries in $P_3$ will also change, but before describing this we introduce some terminology. For a query $\mathsf{Send}_1(\mathit{Server}, j, \mathsf{msg}_1)$ where $\mathsf{msg}_1$ is adversarially-generated, let $\mathsf{msg}_1 = \langle \mathit{Client} | \mathsf{VK} | A | B | C | D \rangle$ and $\alpha = H(\mathit{PIDs} | \mathsf{VK} | A | B | C)$. If either $A^{\chi_1 + \alpha\xi_1} B^{\chi_2 + \alpha\xi_2} \neq D$ or $C/pw_{Client} \neq A^{\kappa}$, then $\mathsf{msg}_1$ is said to be *invalid*. Otherwise, $\mathsf{msg}_1$ is said to be *valid*. Similarly, for a query $\mathsf{Send}_2(\mathit{Client}, i, \mathsf{msg}_2)$ where $\mathsf{msg}_2$ is adversarially-generated, let $\mathsf{msg}_2 = \langle \mathit{Server} | E | F | G | I | J \rangle$ and $\beta$ be as defined in Figure 4. If either $F^{\chi_1+\beta\xi_1}G^{\chi_2+\beta\xi_2} \neq J$ or $I/pw_{Client} \neq F^{\kappa}$ then $\mathsf{msg}_2$ is said to be *invalid.* Otherwise, $\mathsf{msg}_2$ is said to be *valid.* Informally, valid messages are encryptions of the correct password while invalid messages are not. Note that the simulator can efficiently determine whether any given adversarially-generated message is valid because it knows $\kappa, \chi_1, \chi_2, \xi_1, \xi_2$.

Given this terminology, we continue with our description of experiment $P_3$. When the adversary makes oracle query $\mathsf{Send}_2(\mathit{Client}, i, \mathsf{msg}_2)$, the simulator examines $\mathsf{msg}_2$. If $\mathsf{msg}_2$ is adversarially-generated and valid, the query is answered as in $P_2$ except that $\mathsf{sk}^i_{Client}$ is assigned[9] the special value $\nabla$. In any other case (i.e., $\mathsf{msg}_2$ is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in experiment $P_2$.

When the adversary makes oracle query $\mathsf{Send}_3(\mathit{Server}, j, \mathsf{msg}_3)$, the simulator examines $\mathsf{msg}_1 = \mathsf{first\text{-}msg\text{-}in}$ for instance $\Pi^j_{Server}$.[10] If $\mathsf{msg}_1$ is adversarially-generated and valid, the query is answered as in experiment $P_2$ except that $\mathsf{sk}^j_{Server}$ is assigned the special value $\nabla$. In any other case (i.e., $\mathsf{msg}_1$ is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in experiment $P_2$.

Finally, the definition of the adversary's success in $P_3$ is changed. If the adversary ever queries $\mathsf{Reveal}(U, i)$ or $\mathsf{Test}(U, i)$ where $\mathsf{sk}^i_U = \nabla$, the simulator halts and the adversary succeeds. Otherwise, the adversary's success is determined as in experiment $P_2$.

**Claim 3** $\mathsf{Adv}_{\mathcal{A},P_2}(k) \leq \mathsf{Adv}_{\mathcal{A},P_3}(k)$.

The distribution of the public parameters is identical in experiments $P_2$ and $P_3$. Furthermore, the distributions on the adversary's view in experiments $P_2$ and $P_3$ are identical up to the point when the adversary queries $\mathsf{Reveal}(U, i)$ or $\mathsf{Test}(U, i)$ with $\mathsf{sk}^i_U = \nabla$; if such a query is never made, the distributions on the view are identical. Since the adversary succeeds if such a query ever occurs, the claim follows. $\square$

**Experiment** $P_4$: In experiment $P_4$, we again modify the simulator's response to a $\mathsf{Send}_3$ query. Upon receiving query $\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$, the simulator examines $\mathsf{msg}_1 = \mathsf{first\text{-}msg\text{-}in}$ for instance $\Pi^j_{Server}$. If $\mathsf{msg}_1$ is oracle-generated and a value is to be assigned to $\mathsf{sk}^j_{Server}$, the simulator finds the unique pair *Client*, $i$ such that $\mathsf{sid}^i_{Client} = \mathsf{sid}^j_{Server}$ (we comment on the existence of a unique such pair below) and sets $\mathsf{sk}^j_{Server} := \mathsf{sk}^i_{Client}$. In any other case, the query is answered as in experiment $P_3$.

That there exists a unique pair *Client*, $i$ with $\mathsf{sid}^i_{Client} = \mathsf{sid}^j_{Server}$ in the situation above follows from the fact that, assuming $\mathsf{sk}^j_{Server}$ is to be assigned a value — and so in particular the simulation is not aborted — the following hold: (1) there is a unique pair *Client*, $i$ such that instance $\Pi^i_{Client}$ uses $\mathsf{VK}$, where $\mathsf{VK}$ is the verification key contained in $\mathsf{msg}_1$ (this is a consequence of $\mathsf{msg}_1$ being oracle-generated, and the fact that the simulator aborts if an oracle-generated verification key is ever used twice); furthermore, (2) this pair satisfies $\mathsf{sid}^i_{Client} = \mathsf{sid}^j_{Server}$ (this is a consequence of the fact that the simulator aborts if a signature forgery occurs).

Since $\mathsf{sid}^j_{Server} = \mathsf{sid}^i_{Client}$ implies $\mathsf{sk}^j_{Server} = \mathsf{sk}^i_{Client}$ in $P_3$, the following is immediate:

[9]Here and in the remainder of the proof, it is understood that when we say a certain variable is assigned some value this is conditioned on that variable being assigned a value at all (and, e.g., the oracle query does not return $\perp$ for some other reason, nor is the experiment aborted).

[10]If the adversary had not previously made a $\mathsf{Send}_1$ query for this instance, the simulator simply responds with $\perp$ as in previous experiments. We will not explicitly mention this in the rest of the proof.

**Claim 4** $\mathsf{Adv}_{\mathcal{A},P_4}(k) = \mathsf{Adv}_{\mathcal{A},P_3}(k)$.

**Experiment** $P_5$: In experiment $P_5$, we again modify the simulator's response to a $\mathsf{Send}_3$ query. Upon receiving query $\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$, the simulator examines $\mathsf{msg}_1 = \mathsf{first\text{-}msg\text{-}in}$ for the same instance $\Pi^j_{\mathit{Server}}$. If $\mathsf{msg}_1$ is adversarially-generated and invalid, the session key $\mathsf{sk}^j_{\mathit{Server}}$ is assigned a value randomly chosen from $\mathbb{G}$. In all other cases, the query is answered as in experiment $P_4$.

**Claim 5** $\mathsf{Adv}_{\mathcal{A},P_5}(k) = \mathsf{Adv}_{\mathcal{A},P_4}(k)$.

We prove the claim by showing that the distributions on the adversary's view in the two experiments are identical. For a given query $\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$ where $\mathsf{msg}_1 = \mathsf{first\text{-}msg\text{-}in} = \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$ is adversarially-generated and invalid, let $\mathsf{msg}_2 = \mathsf{first\text{-}msg\text{-}out} = \langle \mathit{Server}|E|F|G|I|J\rangle$ and $\alpha = H(\mathit{PIDs}|\mathsf{VK}|A|B|C)$. Since $\mathsf{msg}_1$ is invalid, either $A^{\chi_1+\alpha\xi_1}B^{\chi_2+\alpha\xi_2} \neq D$ or else $C/pw_{\mathit{Client}} \neq A^{\kappa}$ (or possibly both). For any $\mu, \nu \in \mathbb{G}$ and fixing the randomness used in the rest of experiment $P_4$, the probability over choice of $x, y, z, w$ that $E = \mu$ and $\mathsf{sk}^j_{\mathit{Server}} = \nu$ is exactly the probability that

$$\log_{g_1} \mu = x + y \cdot \log_{g_1} g_2 + z \cdot \log_{g_1} h + w \cdot \log_{g_1}(cd^{\alpha}) \tag{4}$$

and

$$\log_{g_1} \nu - r \log_{g_1} K = x \cdot \log_{g_1} A + y \cdot \log_{g_1} B + z \cdot \log_{g_1}(C/pw_{\mathit{Client}}) + w \cdot \log_{g_1} D, \tag{5}$$

using the fact that $g_1$ is a generator. Now consider two cases depending on the value of $\log_{g_1} A$. If $\log_{g_1} A = 0$, then since $\mathsf{msg}_1$ is invalid at least one of the values $\log_{g_1} B$, $\log_{g_1}(C/pw_{\mathit{Client}})$, or $\log_{g_1} D$ is not equal to 0. One can then immediately verify that Equations (4) and (5), viewed as equations over $\mathbb{Z}_q$ in the variables $x, y, z, w$, are linearly independent and not identically zero. If $\log_{g_1} A \neq 0$, it can be similarly verified that Equations (4) and (5) are linearly independent and not identically zero. In either case, then, the desired probability is $1/q^2$. Thus, the value of $\mathsf{sk}^j_{\mathit{Server}}$ is independent of the value of $E$ and hence independent of the remainder of the experiment. $\square$

Before describing the next experiment, let us first summarize where things stand in experiment $P_5$. In responding to a query $\mathsf{Send}_3(S, j, \mathsf{msg}_3)$ in that experiment, the simulator does not need to use the value $r$ (stored as part of the internal state for instance $\Pi^j_S$) to compute the value of the session key. In particular, if $\mathsf{msg}_1$ was the initial message sent to this instance the simulator proceeds as follows:

- If $\mathsf{msg}_1$ is adversarially-generated and valid, $\mathsf{sk}^j_S$ is set equal to $\nabla$ as described in experiment $P_3$.
- If $\mathsf{msg}_1$ is oracle-generated, $\mathsf{sk}^j_S$ is set equal to the session key for the partnered client instance as described in experiment $P_4$.
- If $\mathsf{msg}_1$ is adversarially-generated and invalid, $\mathsf{sk}^j_S$ is chosen at random as described in experiment $P_5$.

We will rely on the above when we prove the subsequent claim.

**Experiment** $P_6$: In experiment $P_6$, we modify the simulator's response to $\mathsf{Send}_1$ queries. Now, $I$ is computed as $h^r g_1^{N+1}$, where the dictionary of legal passwords is $\{1, \ldots, N\}$; note that $g_1^{N+1}$ is not a valid password since $N < q$.

**Claim 6** *Under the DDH assumption,* $|\mathsf{Adv}_{\mathcal{A},P_5}(k) - \mathsf{Adv}_{\mathcal{A},P_6}(k)|$ *is negligible.*

A proof of the claim relies on the security of the (modified) Cramer-Shoup encryption scheme against adaptive chosen-ciphertext attacks; cf. Section 2.4. We show that the simulator can use $\mathcal{A}$ as a subroutine in order to distinguish encryptions of the correct client password(s) from encryptions of $g_1^{N+1}$. The simulator is given a public key $pk = \langle \mathbb{G}, g_1, g_2, h, c, d, H \rangle$ for an instance of the Cramer-Shoup encryption scheme and may repeatedly query an encryption oracle $\mathcal{E}_{pk,\tilde{b}}(\cdot,\cdot,\cdot)$ where $\tilde{b}$ is a randomly-chosen bit (unknown to the simulator). The simulator may also repeatedly query a decryption oracle $\mathcal{D}_{sk}(\cdot,\cdot)$ subject to the restriction stated in Definition 6. By definition, the advantage of the simulator in attacking the encryption scheme is the absolute value of the difference between the probability the simulator outputs 1 when $\tilde{b} = 1$ and the probability the simulator outputs 1 when $\tilde{b} = 0$.

The simulator begins by running the following modified initialization protocol:

```
$\mathsf{Initialize}'(1^k, \mathbb{G}, g_1, g_2, h, c, d, H)$ —
    $(\mathsf{Client}, \mathsf{Server}) \leftarrow \mathsf{UserGen}(1^k)$
    for each $C \in \mathsf{Client}$
        $pw'_C \leftarrow \{1, \ldots, N\}$
        $pw_C := g_1^{pw'_C}$
        for each $S \in \mathsf{Server}$
            $pw_{S,C} := pw_C$
    return $\mathsf{Client}, \mathsf{Server}, \mathbb{G}, g_1, g_2, h, c, d, H$
```

The simulator responds to $\mathsf{Send}_0$, $\mathsf{Execute}$, $\mathsf{Reveal}$, and $\mathsf{Test}$ oracle queries as in experiments $P_5$ and $P_6$. The simulator responds to $\mathsf{Send}_1$, $\mathsf{Send}_2$, and $\mathsf{Send}_3$ queries as shown in Figure 6. In words:

- In responding to a $\mathsf{Send}_1$ query the simulator no longer computes the values $F, G, I, J$ on its own. Instead, it queries its encryption oracle using (1) the appropriate label, and (2) the correct password and $g_1^{N+1}$ as its two "messages." It then uses the ciphertext $\langle F|G|I|J \rangle$ returned by this oracle to construct $\mathsf{msg\text{-}out}$.

- In responding to a $\mathsf{Send}_3$ query, the simulator first checks whether $\mathsf{msg}_1 = \mathsf{first\text{-}msg\text{-}in}$ is oracle- or adversarially-generated. If $\mathsf{msg}_1$ is oracle-generated, the session key is computed by finding the partnered client instance (as discussed above). If $\mathsf{msg}_1$ is adversarially-generated, the simulator determines whether $\mathsf{msg}_1$ is valid or invalid using its decryption oracle and then sets the session key appropriately.

- In responding to a $\mathsf{Send}_2$ query, the simulator uses its decryption oracle (as needed) to determine whether an adversarially-generated incoming message is valid or invalid.

Note that the simulator never need submit to its decryption oracle an "illegal" query (i.e., a ciphertext that it previously received from its encryption oracle) since the simulator never

```
$\mathsf{Send}_1(\mathit{Server}, j, \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle)$ —
    if $\left(\mathit{Server} \notin \mathsf{Server}$ or $\mathsf{used}^j_{\mathit{Server}}\right)$ return $\perp$
    $\mathsf{used}^j_{\mathit{Server}} := \text{TRUE}$;  $\mathit{PIDs} := \mathit{Client}|\mathit{Server}$
    if $A, B, C, D \notin \mathbb{G}$ or $\mathit{Client} \notin \mathsf{Client}$
        $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$;  return $\mathsf{status}^j_{\mathit{Server}}$
    $x, y, z, w \leftarrow \mathbb{Z}_q$;  $\alpha := H(\mathit{PIDs}|\mathsf{VK}|A|B|C)$
    $E := g_1^x g_2^y h^z (cd^\alpha)^w$ ; $\text{LABEL} := \mathsf{msg\text{-}in}|\mathit{Server}|E$
    $\langle F|G|I|J\rangle \leftarrow \mathcal{E}_{pk,\tilde{b}}(\text{LABEL}, pw_{\mathit{Server},\mathit{Client}}, g_1^{N+1})$
    $\mathsf{msg\text{-}out} := \langle \mathit{Server}|E|F|G|I|J\rangle$
    $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \mathit{Client}, \text{FALSE}, \text{FALSE}\rangle$
    $\mathsf{state}^j_{\mathit{Server}} := \langle \mathsf{msg\text{-}in}, x, y, z, w, \mathsf{msg\text{-}out}, pw_{\mathit{Server},\mathit{Client}}\rangle$
    return $\mathsf{msg\text{-}out}, \mathsf{status}^j_{\mathit{Server}}$

$\mathsf{Send}_2(\mathit{Client}, i, \langle \mathit{Server}|E|F|G|I|J\rangle)$ —
    if $(\mathit{Client} \notin \mathsf{Client}$ or $\neg\mathsf{used}^i_{\mathit{Client}}$ or $\mathsf{term}^i_{\mathit{Client}})$ return $\perp$
    if $\mathit{Server} \neq \mathsf{pid}^i_{\mathit{Client}}$ or $E, F, G, I, J \notin \mathbb{G}$
        $\mathsf{status}^i_{\mathit{Client}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$;  return $\mathsf{status}^i_{\mathit{Client}}$
    $\langle \mathsf{SK}, r, \mathsf{first\text{-}msg\text{-}out}\rangle := \mathsf{state}^i_{\mathit{Client}}$
    $x, y, z, w \leftarrow \mathbb{Z}_q$;  $\beta := H(\mathsf{first\text{-}msg\text{-}out}|\mathit{Server}|E|F|G|I)$
    $K := g_1^x g_2^y h^z (cd^\beta)^w$;  $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\mathsf{first\text{-}msg\text{-}out}|\mathsf{msg\text{-}in}|K)$;
    $\mathsf{msg\text{-}out} := \langle K|\mathsf{Sig}\rangle$
    $\mathsf{sid}^i_{\mathit{Client}} := \langle \mathsf{first\text{-}msg\text{-}out}|\mathsf{msg\text{-}in}|\mathsf{msg\text{-}out}\rangle$
    $\mathsf{acc}^i_{\mathit{Client}} := \mathsf{term}^i_{\mathit{Client}} := \text{TRUE}$
    $\text{LABEL} := \mathsf{first\text{-}msg\text{-}out}|\mathit{Server}|E$
    if $\mathsf{msg\text{-}in}$ is adversarially-generated and $\mathcal{D}_{sk}(\text{LABEL}, \langle F|G|I|J\rangle) = pw_{\mathit{Client}}$
        $\mathsf{sk}^i_{\mathit{Client}} := \nabla$
    else
        $\mathsf{sk}^i_{\mathit{Client}} := E^r F^x G^y (I/pw_{\mathit{Client}})^z J^w$
    return $\mathsf{msg\text{-}out}, \mathsf{status}^i_{\mathit{Client}}$

$\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$ —
    if $\left(\mathit{Server} \notin \mathsf{Server}$ or $\neg\mathsf{used}^j_{\mathit{Server}}$ or $\mathsf{term}^j_{\mathit{Server}}\right)$ return $\perp$
    $\langle \mathsf{first\text{-}msg\text{-}in}, x, y, z, w, \mathsf{first\text{-}msg\text{-}out}, pw\rangle := \mathsf{state}^j_{\mathit{Server}}$
    $\langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle := \mathsf{first\text{-}msg\text{-}in}$
    if $K \notin \mathbb{G}$ or $\mathsf{Vrfy}_{\mathsf{VK}}(\mathsf{first\text{-}msg\text{-}in}|\mathsf{first\text{-}msg\text{-}out}|K, \mathsf{Sig}) \neq 1$
        $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$;  return $\mathsf{status}^j_{\mathit{Server}}$
    $\mathsf{sid}^j_{\mathit{Server}} := \langle \mathsf{first\text{-}msg\text{-}in}|\mathsf{first\text{-}msg\text{-}out}|\mathsf{msg\text{-}in}\rangle$
    $\mathsf{acc}^j_{\mathit{Server}} := \mathsf{term}^j_{\mathit{Server}} := \text{TRUE}$
    if $\mathsf{first\text{-}msg\text{-}in}$ is oracle-generated
        find the unique $i$ such that $\mathsf{sid}^i_{\mathit{Client}} = \mathsf{sid}^j_{\mathit{Server}}$
        $\mathsf{sk}^j_{\mathit{Server}} := \mathsf{sk}^i_{\mathit{Client}}$
    else
        $\text{LABEL} := \mathit{Client}|\mathit{Server}|\mathsf{VK}$
        $pw' := \mathcal{D}_{sk}(\text{LABEL}, \langle A|B|C|D\rangle)$
        if $pw' = pw$ then $\mathsf{sk}^j_{\mathit{Server}} := \nabla$
        if $pw' \neq pw$ then $\mathsf{sk}^j_{\mathit{Server}} \leftarrow \mathbb{G}$
    return $\mathsf{status}^j_{\mathit{Server}}$
```

Figure 6: The modified Send oracles for the proof of Claim 6.

needs to check whether oracle-generated messages are valid or invalid. Finally, the simulator outputs 1 if and only if $\mathcal{A}$ succeeds.

Examination of Figure 6 shows that the $\mathsf{Send}_2$ and $\mathsf{Send}_3$ oracles behave exactly the same as in both experiments $P_5$ and $P_6$. On the other hand, when $\tilde{b} = 0$ the actions of the $\mathsf{Send}_1$ oracle are as in experiment $P_5$, while if $\tilde{b} = 1$ then the actions of the $\mathsf{Send}_1$ oracle are as in experiment $P_6$. The simulator's advantage in attacking the Cramer-Shoup encryption scheme is therefore exactly $|\Pr_{\mathcal{A},P_5}[\mathsf{Succ}] - \Pr_{\mathcal{A},P_6}[\mathsf{Succ}]|$. The claim follows since this advantage must be negligible under the DDH assumption, since the Cramer-Shoup encryption scheme is secure given this assumption (cf. Section 2.4). $\square$

**Experiment** $P_7$: In experiment $P_7$, queries to the $\mathsf{Send}_2$ oracle are handled differently. In response to a query $\mathsf{Send}_2(\mathit{Client}, i, \mathsf{msg}_2)$, if $\mathsf{msg}_2$ is either oracle-generated or adversarially-generated but invalid, the session key $\mathsf{sk}^i_{\mathit{Client}}$ is assigned a value chosen randomly from $\mathbb{G}$.

**Claim 7** $\mathsf{Adv}_{\mathcal{A},P_7}(k) = \mathsf{Adv}_{\mathcal{A},P_6}(k)$.

We prove the claim by showing that the distributions on the adversary's view are identical in the two experiments. Consider a particular query $\mathsf{Send}_2(\mathit{Client}, i, \mathsf{msg}_2)$ in $P_6$. Regardless of whether $\mathsf{msg}_2$ is oracle-generated or adversarially-generated but invalid, it is the case that either $F^{\chi_1+\beta\xi_1}G^{\chi_2+\beta\xi_2} \neq J$ or $I/pw_{\mathit{Client}} \neq F^{\kappa}$. A proof similar to that used in proving Claim 5 then shows that $\mathsf{sk}^i_{\mathit{Client}}$ in experiment $P_6$ is uniformly distributed in $\mathbb{G}$, independent of the rest of the experiment. $\square$

**Experiment** $P_8$: In experiment $P_8$, we modify the simulator's response to $\mathsf{Send}_0$ queries. Now, $C$ is computed as $C := h^r g_1^{N+1}$; note that $g_1^{N+1}$ is not a valid password. (This modification parallels the change made in going from experiment $P_5$ to experiment $P_6$.)

**Claim 8** *Under the DDH assumption,* $|\mathsf{Adv}_{\mathcal{A},P_8}(k) - \mathsf{Adv}_{\mathcal{A},P_7}(k)|$ *is negligible.*

The proof exactly follows that of Claim 6. In particular, note that in responding to a query $\mathsf{Send}_2(C, i, \mathsf{msg}_2)$ in experiment $P_7$, the simulator never requires the value $r$ (stored as part of the internal state for instance $\Pi^i_C$) to compute a session key: if $\mathsf{msg}_2$ is oracle-generated, the session key is assigned a randomly-chosen value; if $\mathsf{msg}_2$ is adversarially-generated and invalid (which can be verified using the decryption oracle), the session key is also assigned a randomly-chosen value; finally, if $\mathsf{msg}_2$ is adversarially-generated and valid (which can again be verified using the decryption oracle), the session key is assigned $\nabla$. Oracle queries to $\mathsf{Send}_3$ can also be answered by the simulator, using its decryption oracle as necessary. Finally, there is never a need for the simulator to make an "illegal" query its decryption oracle. The claim follows. $\square$

The adversary's view in experiment $P_8$ is independent of the passwords chosen by the simulator until one of the following occur:

- The adversary queries $\mathsf{Reveal}(\mathit{Client}, i)$ or $\mathsf{Test}(\mathit{Client}, i)$, where the adversary had previously queried $\mathsf{Send}_2(\mathit{Client}, i, \mathsf{msg}_2)$ for adversarially-generated and valid $\mathsf{msg}_2$.

- The adversary queries $\mathsf{Reveal}(\mathit{Server}, j)$ or $\mathsf{Test}(\mathit{Server}, j)$, where the adversary had previously queried $\mathsf{Send}_1(\mathit{Server}, j, \mathsf{msg}_1)$ for adversarially-generated and valid $\mathsf{msg}_1$.

The probability that either of these events occur is at most $Q(k)/N$, where $Q(k)$ is the number of on-line attacks made by $\mathcal{A}$. If neither of the above events occur, then the adversary succeeds only if it can guess the value of $b$ used by the $\mathsf{Test}$ oracle. However, if neither of the above events occur and the adversary queries $\mathsf{Test}(U, i)$ where $\Pi_U^i$ is fresh and $\mathsf{acc}_U^i = \text{TRUE}$, then $\mathsf{sk}_U^i$ is randomly-distributed in $\mathbb{G}$ independent of $\mathcal{A}$'s view; the probability of correctly guessing $b$ in this case is therefore exactly 1/2.

The preceding discussion implies that

$$\Pr_{\mathcal{A},P_8}[\mathsf{Succ}] \leq Q(k)/N + \tfrac{1}{2} \cdot (1 - \tfrac{Q(k)}{N})$$

and thus the adversary's advantage in experiment $P_8$ is at most $Q(k)/N$. The sequence of claims proven above show that

$$\mathsf{Adv}_{\mathcal{A},P_0}(k) \leq \mathsf{Adv}_{\mathcal{A},P_8}(k) + \varepsilon(k)$$

for some negligible function $\varepsilon(\cdot)$ and therefore the adversary's advantage in $P_0$ (i.e., the original protocol) is at most $Q(k)/N$ plus some negligible quantity. This completes the proof of the theorem. ■

## 3.3 Proof of Forward Secrecy

We first demonstrate a potential attack on the protocol (in the stronger adversarial model of Section 2.3) if it is modified in a seemingly-innocuous way. Then, we show a proof that the protocol as formally defined does indeed achieve forward secrecy.

Consider the following change to the protocol as described in the previous section: when responding to a $\mathsf{Send}_1$ query, a server no longer stores $pw_{S,C}$ as part of its state information (cf. Figure 4). Now, when a server responds to a $\mathsf{Send}_3$ query it simply sets $pw = pw_{S,C}$. In other words, instead of using the password stored as part of its (temporary, volatile) state information, the server simply uses the value of the client's password as stored in its long-term memory (e.g., the password file).

We now show that this simple change introduces a vulnerability in the attack model described in Section 2.3. Fix a client *Client*. The attack begins by having the adversary impersonate *Client* to a server $S$ using an arbitrary password, say $pw = g_1$. That is, the adversary generates $\mathsf{VK}$, chooses a random $r_1$, computes $A = g_1^{r_1}$, $B = g_2^{r_1}$, $C = h_{r_1} \cdot g_1$, and $D = (cd^{\alpha})^{r_1}$ (for $\alpha$ computed in the appropriate way), and sends $\langle \mathit{Client} | \mathsf{VK} | A | B | C | D \rangle$ to server $S$. The server will respond with a message $\langle S | E | F | G | I | J \rangle$ (which is computed using password $pw_{S,\mathit{Client}} = pw_{\mathit{Client}}$). Next, the adversary changes the password stored at $S$ for *Client* (i.e., $pw_{S,\mathit{Client}}$) to the value $g_1$ using oracle call $\mathsf{Corrupt}_2(S, \mathit{Client}, g_1)$. Finally, the adversary computes the final message of the protocol just as a legitimate client would by choosing $x_1, y_1, z_1, w_1$, computing $K = g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta})^{w_1}$ (for $\beta$ computed in the appropriate way), and sending $\langle K | \mathsf{Sig} \rangle$. Finally, the adversary obtains $\mathsf{sk}_S$ for this instance of the server using a $\mathsf{Reveal}$ query.

We claim that the adversary can now determine $pw_{\mathit{Client}}$ — the original password of the client — and hence successfully impersonate *Client* to a *different* server $S' \neq S$. To see this, note that when $S$ computes the session key after receiving the final message of the protocol it uses $pw_{S,Client} = g_1$ (which is, we stress, different from the password the server used to compute the second message of the protocol). Thus,

$$\begin{aligned}\mathsf{sk}_S = A^{x_2}B^{y_2}(C/g_1)^{z_2}D^{w_2}K^{r_2} &= g_1^{r_1x_2}g_2^{r_1y_2}h^{r_1z_2}(cd^{\alpha})^{r_1w_2}K^{r_2} \\ &= E^{r_1}K^{r_2}\end{aligned}$$

(where $x_2, y_2, z_2, w_2, r_2$ are values used by the server which are unknown to the adversary). Since the adversary can compute $E^{r_1}$ itself, the adversary can use $\mathsf{sk}_S$ to determine the value $K^{r_2} = \mathsf{sk}_S/E^{r_1}$. Now, using exhaustive search through the password dictionary, the adversary can identify $pw_{Client}$ as that unique value for which:

$$F^{x_1}G^{y_1}(I/pw_{Client})^{z_1}J^{w_1} = K^{r_2}.$$

The protocol as formally specified in the previous section is not vulnerable to this attack since it ensures that a server-instance uses the same password throughout its execution by storing the client's password as part of the state information for that instance. Interestingly, this means that it is possible for two server-instances active at the same time to be using different passwords for a given client (the password being used will be determined by the value of $pw_{S,C}$ at the time the initial $\mathsf{Send}_1$ query for each instance was made).

We now provide a rigorous proof that the protocol as specified achieves forward secrecy.

**Theorem 3** *Under the same assumptions as in Theorem 2, the protocol of Figure 1 achieves forward secrecy.*

**Proof** A significant portion of the proof is similar to the proof of Theorem 2, and so we will focus only on the differences here. As in the previous proof, we imagine a simulator that runs the protocol for any adversary $\mathcal{A}$ (the simulator must now also simulate the $\mathsf{Corrupt}$ oracles for $\mathcal{A}$ but this does not present any problems). When the adversary completes its execution and outputs a bit $b'$, the simulator can tell whether the adversary succeeds by checking whether (1) a single $\mathsf{Test}$ query was made on instance $\Pi_U^i$; (2) $\mathsf{acc}_U^i$ was TRUE at the time of the $\mathsf{Test}$ query; (3) instance $\Pi_U^i$ is fresh (according to the definition of freshness appropriate for this setting; see Section 2.3); and (4) $b' = b$. Success of the adversary is denoted by event $\mathsf{fsSucc}$, and for any game $P$ we let $\mathsf{fsAdv}_{\mathcal{A},P}(k) \overset{\text{def}}{=} 2 \cdot \Pr_{\mathcal{A},P}[\mathsf{fsSucc}] - 1$. We refer to the real execution of the experiment as $P_0$.

**Experiment $P_0'$:** We define experiment $P_0'$ exactly as in the previous proof; following the same reasoning as there, it is clear that $\left|\mathsf{fsAdv}_{\mathcal{A},P_0}(k) - \mathsf{fsAdv}_{\mathcal{A},P_0'}(k)\right|$ is negligible.

**Experiment $P_1$:** In experiment $P_1$, the simulator interacts with the adversary as in $P_0'$ except that the adversary's queries to the $\mathsf{Execute}$ oracle are handled differently. In response to the adversary's oracle query $\mathsf{Execute}(\mathit{Client}, i, \mathit{Server}, j)$, the simulator checks whether $pw_{Client} = pw_{Server,Client}$. If so, the values $C, I$ are computed as $C := h^{r_1} \cdot g_1^{N+1}$ and $I := h^{r_2} \cdot g_1^{N+1}$ (as before, the key point is that $g_1^{N+1}$ is not a valid password), and the session keys are computed as

$$\mathsf{sk}_{Client}^i := \mathsf{sk}_{Server}^j := A^{x_2}B^{y_2}(C/pw_{Client})^{z_2}D^{w_2}F^{x_1}G^{y_1}(I/pw_{Client})^{z_1}J^{w_1}.$$

On the other hand, if $pw_{Client} \neq pw_{Server,Client}$ then the values $C, I$ are computed as $C := h^{r_1} \cdot g_1^{N+1}$ and $I := h^{r_2} \cdot g_1^{N+2}$, and the session keys are computed as

$$\begin{aligned}
\mathsf{sk}^i_{Client} &:= A^{x_2} B^{y_2} (C/pw_{Client})^{z_2} D^{w_2} F^{x_1} G^{y_1} (I/pw_{Client})^{z_1} J^{w_1} \\
\mathsf{sk}^j_{Server} &:= A^{x_2} B^{y_2} (C/pw_{Server,Client})^{z_2} D^{w_2} F^{x_1} G^{y_1} (I/pw_{Server,Client})^{z_1} J^{w_1}.
\end{aligned}$$

(We implicitly assume here that $N + 2 \leq q$ and so neither $g_1^{N+1}$ nor $g_1^{N+2}$ are valid passwords.) Using essentially the same proof as in Claim 1, semantic security of the (modified) Cramer-Shoup encryption scheme implies that $|\mathsf{fsAdv}_{\mathcal{A},P_0'}(k) - \mathsf{fsAdv}_{\mathcal{A},P_1}(k)|$ is negligible.

**Experiment $P_2$:** In experiment $P_2$, the simulator interacts with the adversary as in $P_1$ except that during queries $\mathsf{Execute}(Client, i, Server, j)$ the simulator proceeds as follows: if $pw_{Client} = pw_{Server,Client}$ then $\mathsf{sk}^i_{Client}$ is chosen uniformly at random from $\mathbb{G}$ and $\mathsf{sk}^j_{Server}$ is set equal to $\mathsf{sk}^i_{Client}$. Otherwise, both $\mathsf{sk}^i_{Client}$ and $\mathsf{sk}^j_{Server}$ are chosen independently and uniformly from $\mathbb{G}$.

**Claim 9** $\mathsf{fsAdv}_{\mathcal{A},P_1}(k) = \mathsf{fsAdv}_{\mathcal{A},P_2}(k)$.

The claim follows since the distribution on the view of the adversary is identical in both experiments. The case of $pw_{Client} = pw_{Server,Client}$ exactly follows the proof of Claim 2. So, consider an invocation of the $\mathsf{Execute}$ oracle when $pw_{Client} \neq pw_{Server,Client}$ (for brevity, denote $pw_{Client}$ by $pw_C$ and $pw_{Server,Client}$ by $pw_{S,C}$). We may write

$$C = h^{r_1} \cdot g_1^{N+1} = h^{r_1'} \cdot pw_C = h^{r_1''} \cdot pw_{S,C}$$

for some $r_1', r_1''$, and

$$I = h^{r_2} \cdot g_1^{N+2} = h^{r_2'} \cdot pw_C = h^{r_2''} \cdot pw_{S,C}$$

for some $r_2', r_2''$. Note that none of $r_1, r_1', r_1''$ are equal, and similarly for $r_2, r_2', r_2''$. Furthermore, we have $r_1'' - r_1' = r_2'' - r_2'$ but $r_1' - r_1 \neq r_2' - r_2$.

Now, for any $\mu_1, \mu_2, \nu_1, \nu_2 \in \mathbb{G}$ and fixing the random choices for the remainder of experiment $P_1$, the probability over choice of $x_1, y_1, z_1, w_1, x_2, y_2, z_2, w_2$ that $E = \mu_1$, $K = \mu_2$, $\mathsf{sk}^i_{Client} = \nu_1$, and $\mathsf{sk}^j_{Server} = \nu_2$ is exactly the probability that

$$\begin{aligned}
\log_{g_1} \mu_1 &= x_2 + y_2 \cdot \log_{g_1} g_2 + z_2 \cdot \log_{g_1} h + w_2 \cdot \log_{g_1}(cd^\alpha) && (6)\\
\log_{g_1} \mu_2 &= x_1 + y_1 \cdot \log_{g_1} g_2 + z_1 \cdot \log_{g_1} h + w_1 \cdot \log_{g_1}(cd^\beta) && (7)\\
\log_{g_1} \nu_1 &= x_1 \cdot r_2 + y_1 \cdot r_2 \log_{g_1} g_2 + z_1 \cdot r_2' \log_{g_1} h + w_1 \cdot r_2 \log_{g_1}(cd^\beta) \\
&\quad + x_2 \cdot r_1 + y_2 \cdot r_1 \log_{g_1} g_2 + z_2 \cdot r_1' \log_{g_1} h + w_2 \cdot r_1 \log_{g_1}(cd^\alpha) && (8)\\
\log_{g_1} \nu_2 &= x_1 \cdot r_2 + y_1 \cdot r_2 \log_{g_1} g_2 + z_1 \cdot r_2'' \log_{g_1} h + w_1 \cdot r_2 \log_{g_1}(cd^\beta) \\
&\quad + x_2 \cdot r_1 + y_2 \cdot r_1 \log_{g_1} g_2 + z_2 \cdot r_1'' \log_{g_1} h + w_2 \cdot r_1 \log_{g_1}(cd^\alpha). && (9)
\end{aligned}$$

It may be verified that Equations (6)–(9) are linearly independent and not identically zero, and so the values $E, K, \mathsf{sk}^i_{Client}$, and $\mathsf{sk}^j_{Server}$ are independently and uniformly distributed, independent of the rest of the experiment. The claim follows. □

Before continuing, we introduce some new terminology. First, we say that instance $\Pi^i_U$ is *associated with* $U'$ if either $U = U'$ or $\mathsf{pid}^i_U = U'$. More interestingly, we call a query $\mathsf{Send}_0(C,*,*)$ or $\mathsf{Send}_2(C,*,*)$ *corrupted* if the adversary had previously queried $\mathsf{Corrupt}_1(C)$. Similarly, we call a query $\mathsf{Send}_1(S,i,*)$ (with $\mathsf{pid}_S^i = C$) corrupted if the adversary had previously queried either $\mathsf{Corrupt}_1(C)$ or $\mathsf{Corrupt}_2(S,C,*)$. We call a query $\mathsf{Send}_3(S,i,*)$ corrupted exactly when the corresponding query $\mathsf{Send}_1(S,i,*)$ (i.e., the $\mathsf{Send}_1$ query for the same instance) is corrupted. Note that for any corrupted query $\mathsf{Send}_*(U,i,*)$ the adversary "knows" the password being used by $\Pi_U^i$ at the time this query is made, and furthermore the instance $\Pi_U^i$ is not fresh (as per the definition given in Section 2.3).

The experiments we now introduce exactly parallel those introduced in the previous proof, with the main differences being that we only change the simulator's actions in response to non-corrupted $\mathsf{Send}$ queries.

**Experiment $P_3$:** In experiment $P_3$, the simulator runs the modified initialization procedure as in the proof of Theorem 2, with the values $\kappa, \chi_1, \chi_2, \xi_1, \xi_2$ stored as before. In a way analogous to (but slightly different from) the previous proof, we define a $\mathsf{msg}_1$ or $\mathsf{msg}_2$ to be *oracle-generated* if it is output as a result of an $\mathsf{Execute}$ query or in response to a *non-corrupted* $\mathsf{Send}_0$ or $\mathsf{Send}_1$ query. Otherwise a message is called *adversarially-generated.* We also define *valid* and *invalid* exactly as in the previous proof. We stress, however, that to determine whether a message $\mathsf{msg}_1$ submitted to the $\mathsf{Send}_1$ oracle is valid/invalid, the value of $pw_{Server,Client}$ *at the time of the* $\mathsf{Send}_1$ *query* is used (even if this value is changed at some later point in the experiment via a $\mathsf{Corrupt}_2$ query).

We now continue with our description of experiment $P_3$. When the adversary makes a non-corrupted oracle query $\mathsf{Send}_2(\mathit{Client}, i, \mathsf{msg}_2)$ and $\mathsf{msg}_2$ is adversarially-generated and valid, the query is answered as in experiment $P_2$ but the simulator stores $\nabla$ as the "session key" for this instance. In any other case (i.e., $\mathsf{msg}_2$ is oracle-generated, or adversarially-generated but invalid), the query is exactly answered as in experiment $P_2$.

When the adversary makes non-corrupted oracle query $\mathsf{Send}_3(\mathit{Server}, j, \mathsf{msg}_3)$, the simulator examines $\mathsf{msg}_1 = \mathsf{first\text{-}msg\text{-}in}$ for instance $\Pi_{Server}^j$. If $\mathsf{msg}_1$ is adversarially-generated and valid, the query is answered as in experiment $P_2$ but the simulator stores $\nabla$ as the "session key" for this instance. In any other case (i.e., $\mathsf{msg}_1$ is oracle-generated, or adversarially-generated but invalid), the query is answered exactly as in experiment $P_2$.

Finally, the definition of the adversary's success is changed. If the adversary queries $\mathsf{Test}(U,i)$ or $\mathsf{Reveal}(U,i)$ and $\mathsf{sk}_U^i = \nabla$, the simulator halts and the adversary succeeds. The adversary also succeeds if it queries $\mathsf{Corrupt}_1(C)$ and there exists any instance associated with $C$ which has session key $\nabla$. Otherwise, the adversary succeeds as in experiment $P_2$.

Since there are more ways for the adversary to succeed in $P_3$ and the experiment is otherwise identical to $P_2$ from the adversary's point of view, it follows immediately (as in Claim 3) that $\mathsf{fsAdv}_{\mathcal{A},P_2}(k) \le \mathsf{fsAdv}_{\mathcal{A},P_3}(k)$.

**Experiment $P_4$:** In experiment $P_4$, we again modify the simulator's response to a non-corrupted $\mathsf{Send}_3$ query. Upon receiving non-corrupted query $\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$, the simulator examines $\mathsf{msg}_1 = \mathsf{first\text{-}msg\text{-}in}$ for instance $\Pi_{Server}^j$. If $\mathsf{msg}_1$ is oracle-generated and a value is to be assigned to $\mathsf{sk}_{Server}^j$, the simulator finds the unique pair $\mathit{Client}, i$ such that $\mathsf{sid}_{Client}^i = \mathsf{sid}_{Server}^j$ (that a unique such pair exists follows from a similar argument as in the previous proof) and sets $\mathsf{sk}_{Server}^j = \mathsf{sk}_{Client}^i$.

The modification described above is only made for non-corrupted $\mathsf{Send}_3$ queries, and so in particular this means that the adversary did not query $\mathsf{Corrupt}_2(\mathit{Server}, \mathit{Client}, *)$ before initializing instance $\Pi^j_{Server}$ (using a $\mathsf{Send}_1$ query). It follows that instances $\Pi^j_{Server}$, $\Pi^i_{Client}$ use the same password. But in experiment $P_3$, we always have $\mathsf{sk}^j_{Server} = \mathsf{sk}^i_{Client}$ whenever $\mathsf{sid}^j_{Server} = \mathsf{sid}^i_{Client}$ and instances $\Pi^j_{Server}$, $\Pi^i_{Client}$ use the same password. We therefore have $\mathsf{fsAdv}_{\mathcal{A},P_4}(k) = \mathsf{fsAdv}_{\mathcal{A},P_3}(k)$.

**Experiment $P_5$:** In experiment $P_5$, we again modify the simulator's response to a non-corrupted $\mathsf{Send}_3$ query. Upon receiving non-corrupted query $\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$, the simulator examines $\mathsf{msg}_1 = \mathsf{first\text{-}msg\text{-}in}$ for instance $\Pi^j_{Server}$. If $\mathsf{msg}_1$ is adversarially-generated and invalid, the session key $\mathsf{sk}^j_{Server}$ is assigned a value randomly chosen from $\mathbb{G}$. In all other cases, the query is answered as in experiment $P_4$. Exactly as in Claim 5, it is the case that $\mathsf{fsAdv}_{\mathcal{A},P_5}(k) = \mathsf{fsAdv}_{\mathcal{A},P_4}(k)$ (note in particular that the proof of Claim 5 holds even if the adversary learns the client's password at some later point).

**Experiment $P_6$:** In experiment $P_6$, we modify the simulator's response to non-corrupted $\mathsf{Send}_1$ queries. Now, $I$ is computed as $I := h^r g_1^{N+1}$; again, note that $g_1^{N+1}$ is not a valid password. As in the proof of Claim 6, security of the (modified) Cramer-Shoup encryption scheme implies that $|\mathsf{fsAdv}_{\mathcal{A},P_5}(k) - \mathsf{fsAdv}_{\mathcal{A},P_6}(k)|$ is negligible under the DDH assumption.

**Experiment $P_7$:** In experiment $P_7$, we will modify the simulator's response to non-corrupted $\mathsf{Send}_2$ queries. Now, in response to a non-corrupted query $\mathsf{Send}_2(\mathit{Client}, i, \mathsf{msg}_2)$, the session key $\mathsf{sk}^i_{Client}$ is assigned a value chosen randomly from $\mathbb{G}$ if either: (1) $\mathsf{msg}_2$ is adversarially-generated and invalid; or (2) $\mathsf{msg}_2$ is oracle-generated (recall that a $\mathsf{msg}_2$ output in response to a *corrupted* $\mathsf{Send}_1$ oracle query is *not* considered oracle-generated). In any other case, the session key is assigned a value as in experiment $P_7$. Exactly as in the previous proof (cf. Claim 7), we have $\mathsf{fsAdv}_{\mathcal{A},P_7}(k) = \mathsf{fsAdv}_{\mathcal{A},P_6}(k)$.

**Experiment $P_8$:** In experiment $P_8$, we modify the simulators response to non-corrupted $\mathsf{Send}_0$ queries. Now, $C$ is computed as $C := h^r g_1^{N+1}$. As in the proof of Claim 8, security of the (modified) Cramer-Shoup encryption scheme implies that $|\mathsf{fsAdv}_{\mathcal{A},P_8}(k) - \mathsf{fsAdv}_{\mathcal{A},P_7}(k)|$ is negligible.

In experiment $P_8$, the adversary can succeed in one of two ways: either it correctly guesses the value of the bit used by the $\mathsf{Test}$ oracle, or it succeeds as specified in experiment $P_3$ (namely, if it queries $\mathsf{Test}(U, i)$ or $\mathsf{Reveal}(U, i)$ with $\mathsf{sk}^i_U = \nabla$, or if it queries $\mathsf{Corrupt}_1(C)$ and there exists any instance associated with $C$ which has session key $\nabla$). The probability that it succeeds in the second manner is at most $Q(k)/N$, where $Q(k)$ is the number of on-line attacks made by $\mathcal{A}$ (cf. the definition of on-line attacks in Section 2.3). Otherwise, if the adversary queries $\mathsf{Test}(U, i)$ for a fresh instance $\Pi^i_U$ such that $\mathsf{acc}^i_U = \text{TRUE}$, then $\mathsf{sk}^i_U$ is uniformly-distributed in $\mathbb{G}$ independent of the adversary's view (since, in particular, $\Pi^i_U$ is fresh only if the adversary's $\mathsf{Send}$ queries for this instance are non-corrupted). As in the previous proof, it follows that $\mathsf{fsAdv}_{\mathcal{A},P_8}(k) \le Q(k)/N$ and so

$$\mathsf{fsAdv}_{\mathcal{A},P_0}(k) \le Q(k)/N + \varepsilon(k)$$

for some negligible function $\varepsilon(\cdot)$. This concludes the proof of the theorem. ■

**Acknowledgments.** We are grateful to the reviewers for their thoughtful comments that helped improve the presentation.

## References

[1] M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. *30th ACM Symposium on Theory of Computing (STOC)*, ACM, pp. 419–428, 1998.

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

[3] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *Proc. 1st ACM Conference on Computer and Communications Security*, ACM, pp. 62–73, 1993.

[4] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *Adv. in Cryptology — Crypto 1993*, LNCS vol. 773, Springer-Verlag, pp. 232–249, 1994.

[5] M. Bellare and P. Rogaway. Provably-Secure Session Key Distribution: the Three Party Case. *27th ACM Symposium on Theory of Computing (STOC)*, ACM, pp. 57–66, 1995.

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

[7] S.M. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise. *1st ACM Conf. on Computer and Communications Security*, ACM, pp. 244–250, 1993.

[8] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of Two-Party Authentication Protocols. *IEEE Journal on Selected Areas in Communications* 11(5): 679–693, 1993.

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

[10] V. Boyko. On All-or-Nothing Transforms and Password-Authenticated Key Exchange Protocols. PhD Thesis, MIT, 2000.

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

[12] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. *J. ACM* 51(4): 557–594, 2004.

[13] R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally Composable Password-Based Key Exchange. *Adv. in Cryptology — Eurocrypt 2005*, LNCS vol. 3494, Springer-Verlag, pp. 404–421, 2005.

[14] R. Canetti and H. Krawczyk. Key-Exchange Protocols and Their Use for Building Secure Channels. *Adv. in Cryptology — Eurocrypt 2001*, LNCS vol. 2045, Springer-Verlag, pp. 453–474, 2001.

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

[16] R. Cramer and V. Shoup. Design and Analysis of Practical Public-Key Encryption Schemes Secure Against Adaptive Chosen Ciphertext Attack. *SIAM J. Computing* 33(1): 167–226, 2003.

[17] I. Damgård. Collision Free Hash Functions and Public Key Signature Schemes. *Adv. in Cryptology — Eurocrypt 1987*, LNCS vol. 304, Springer-Verlag, pp. 203–216, 1988.

[18] G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith. Efficient and Non-Interactive Non-Malleable Commitment. *Adv. in Cryptology — Eurocrypt 2001*, LNCS vol. 2045, Springer-Verlag, pp. 40–59, 2001.

[19] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory* 22(6): 644–654, 1976.

[20] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes, and Cryptography* 2(2): 107–125, 1992.

[21] M. Di Raimondo and R. Gennaro. Provably Secure Threshold Password-Authenticated Key Exchange. *Adv. in Cryptology — Eurocrypt 2003*, LNCS vol. 2656, Springer-Verlag, pp. 507–523, 2003.

[22] D. Dolev, C. Dwork, and M. Naor. Nonmalleable Cryptography. *SIAM Journal of Computing* 30(2): 391–437, 2000.

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

[24] R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. *ACM Trans. Information and System Security* 9(2): 181–234, 2006.

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

[26] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal of Computing* 17(2): 281–308, 1988.

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

[28] L. Gong. Optimal Authentication Protocols Resistant to Password Guessing Attacks. *Proc. 8th IEEE Computer Security Foundations Workshop*, pp. 24–29, 1995.

[29] S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. *ACM Trans. Information and System Security* 2(3): 230–268, 1999.

[30] D. Jablon. Strong Password-Only Authenticated Key Exchange. *ACM Computer Communications Review* 26(5): 5–20, 1996.

[31] D. Jablon. Extended Password Key Exchange Protocols Immune to Dictionary Attack. *Proc. of WET-ICE '97*, IEEE, pp. 248–255, 1997.

[32] S. Jiang and G. Gong. Password Based Key Exchange with Mutual Authentication. *Selected Areas in Cryptography 2004*, LNCS vol. 3357, Springer-Verlag, pp. 267–279, 2004.

[33] J. Katz. Efficient Cryptographic Protocols Preventing "Man-in-the-Middle" Attacks. PhD Thesis, Columbia University, 2002.

[34] 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.

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

[36] J. Katz, R. Ostrovsky, and M. Yung. Forward Secrecy in Password-Only Key-Exchange Protocols. *Security in Communication Networks: SCN 2002*, LNCS vol. 2576, Springer-Verlag, pp. 29–44, 2002.

[37] C. Kaufman, R. Perlman, and M. Speciner. *Network Security: Private Communication in a Public World, second edition.* Prentice Hall, 2002.

[38] D. Klein. Foiling the Cracker: Survey and Improvements of Password Security. *Proc. USENIX Security Workshop*, 1990.

[39] T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham. Reducing Risks from Poorly-Chosen Keys. *ACM Operating Systems Review* 23(5): 14–18, 1989.

[40] S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. *Security Protocols Workshop '97*, LNCS 1361, Springer-Verlag, pp. 79–90, 1997.

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

[42] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1999.

[43] R. Morris and K. Thompson. Password Security: A Case History. *Comm. ACM* 22(11): 594–597, 1979.

[44] M.-H. Nguyen and S.P. Vadhan. Simpler Session-Key Generation from Short Random Passwords. *Theory of Cryptography*, LNCS vol. 2951, Springer-Verlag, pp. 428–445, 2004.

[45] S. Patel. Number-Theoretic Attacks on Secure Password Schemes. *Proc. IEEE Symposium on Research in Security and Privacy*, IEEE, pp. 236–247, 1997.

[46] V. Shoup. On Formal Models for Secure Key Exchange. Draft, 1999. Available at `http://eprint.iacr.org/1999/012`.

[47] V. Shoup. A Proposal for an ISO Standard for Public-Key Encryption, version 2.1. Draft, 2001. Available at `http://eprint.iacr.org/2001/112`.

[48] E. Spafford. Observing Reusable Password Choices. *Proc. USENIX Security Workshop*, 1992.

[49] M. Steiner, G. Tsudik, and M. Waidner. Refinement and Extension of Encrypted Key Exchange. *ACM Operating Systems Review* 29(3): 22–30, 1995.

[50] T. Wu. The Secure Remote Password Protocol. *Proc. Internet Society Symp. on Network and Distributed System Security*, pp. 97–111, 1998.

[51] T. Wu. A Real-World Analysis of Kerberos Password Security. *Proceedings of the Internet Society Symp. on Network and Distributed System Security*, pp. 13–22, 1999.