On the Security of the SPEKE Password-Authenticated Key Exchange Protocol

Philip MacKenzie[*]

July 19, 2001

Abstract

In the most strict formal definition of security for password-authenticated key exchange, an adversary can test at most one password per impersonation attempt. We propose a slightly relaxed definition which restricts an adversary to testing at most a constant number of passwords per impersonation attempt. This definition seems useful, since there is currently a popular password-authenticated key exchange protocol called SRP that seems resistant to off-line dictionary attack, yet does allow an adversary to test two passwords per impersonation attempt. In this paper we prove (in the random oracle model) that a certain instantiation of the SPEKE protocol that uses hashed passwords instead of non-hashed passwords is a secure password-authenticated key exchange protocol (using our relaxed definition) based on a new assumption, the *Decision Inverted-Additive Diffie-Hellman* assumption. Since this is a new security assumption, we investigate its security and relation to other assumptions; specifically we prove a lower bound for breaking this new assumption in the generic model, and we show that the computational version of this new assumption is equivalent to the Computational Diffie-Hellman assumption.

1 Introduction

The password-authenticated key exchange problem can be informally stated as follows. Two entities who only share a password wish to set up a secure session over an insecure network. That is, they wish to authenticate each other and agree on a large session key to be used for protecting their subsequent communication. If one of the entities is a user and the other is a server, then one can see that this is a problem in the area of *remote user access*.

Many previous solutions have either required additional cryptographic infrastructure, like a certificate authority and certified (or at least known) public keys, or have been vulnerable to certain types of attacks, like replay attacks or off-line dictionary attacks. However, newer protocols have been developed that have promised increased security, e.g., [7, 8, 18, 17, 31, 20, 21, 22, 32, 27]. Some of these have been broken [25, 26], but three (and some variants thereof) have actually been formally proven secure: SNAPI [23], PAK [13], and EKE2 [3] (based on EKE [7]). The first two are proven in the random oracle model [5], while the last is proven secure in the random oracle and ideal cipher model. The first is proven secure based on the RSA assumption, while the last two are proven secure based on the Computational Diffie-Hellman assumption.

In this paper we show that a certain instantiation of the SPEKE protocol [20] is secure in the random oracle model based on a new assumption, the hardness of the *Decision Inverted-Additive*

[*]Bell Laboratories, Lucent Technologies philmac@lucent.com

Diffie-Hellman problem. Informally, given a generator g of an appropriate group of size q, the problem is to distinguish the distributions $(g^{1/x}, g^{1/y}, g^{1/(x+y)})$ and $(g^{1/x}, g^{1/y}, g^{1/z})$, where x, y, z are randomly drawn from Z_q. We prove a lower bound for solving this problem in the generic model that asymptotically matches the lower bound for solving Decision Diffie-Hellman in the generic model. We also show that the computational version of this problem is equivalent (with respect to polynomial-time black-box reduction) to the computational version of the Diffie-Hellman problem. We note that prior to the result in this paper, there was no proof of security for any instantiation of the SPEKE protocol, and the (heuristic) security of the protocol was believed to be based on the standard Diffie-Hellman problem. We believe that it is quite surprising that its provable security seems to require a slightly different assumption.

In addition to basing the security of SPEKE on this new assumption, we also have to slightly relax the definition of security. In SNAPI, PAK, and EKE2, it is shown that in a single impersonation attempt (think: login attempt), an adversary can gain information about at most one possible password. In SPEKE, however, we are only able to show that in a single impersonation attemption, an adversary can gain information about at most two possible passwords. This is certainly not as strong a statement, but it is probably sufficient for most realistic systems, since the security is reduced by only a small constant factor.

We should emphasize that we do not actually know how to test two passwords using one impersonation attempt. This seems to be an artifact of the proof technique. However, there is at least one other password-authenticated key exchange protocol (namely, SRP [32]) that is heuristically secure against off-line dictionary attacks, yet allows the testing of two passwords in one impersonation attempt.[1] Thus this relaxed definition of security seems like a useful and important notion for analyzing real password-authenticated key exchange protocols.

1.1 Models for Secure Authentication and Key Exchange

Bellare and Rogaway [4] present the first formal model of security for entity authentication and key exchange, for the symmetric two party case. In [6] they extend it to the three party case. Blake-Wilson et.al. [9] further extend the model to cover the asymmetric setting. Independently, MacKenzie and Swaminathan [24] and Bellare et al. [3] present extensions to the model to allow for password authentication. Halevi and Krawczyk [19] and Boyarsky [12] present models which include both passwords and asymmetric keys (since both of those papers deal with protocols that are password-based, but rely on server public keys).

Bellare, Canetti, and Krawczyk [2] present a different model for security of entity authentication and key exchange, based on the multi-party simulatability tradition [1]. Shoup [30] refines and extends their model. Boyko et al. [13] extend the model of [30] to include password authentication.

2 Model

For our proofs, we use the model for password-authenticated key exchange defined in Boyko et al. [13]. We assume the adversary totally controls the network, a la [4].

Briefly, this model is defined using an ideal key exchange system, and a real system in which the protocol participants and adversaries work. The ideal system will be secure by definition, and

[1]Using the notation in [32], for passwords P_1 and P_2 and salt s, an adversary that impersonates an SRP server may calculate $v_1 = g^{H(s,P_1)}$ and $v_2 = g^{H(s,P_2)}$ and send $B = v_1 + v_2$ in the second message of the SRP protocol, instead of $B = v_1 + g^b$ for a random b. Then the adversary can test for password P_1 by assuming $b = H(s, P_2)$ and test for password P_2 by assuming $b = H(s, P_1)$. This attack was discovered by Bleichenbacher [10].

the idea is to show that anything an adversary can do to our protocol in the real system can also be done in the ideal system, and thus it would follow that the protocol is secure in the real system.

2.1 Ideal system

We assume there is a set of (honest) *users*, indexed $i = 1, 2, \ldots$. Each user i may have several *instances* $j = 1, 2, \ldots$. Then (i, j) refers to a given *user instance*. A user instance (i, j) is told the identity of its partner, i.e., the user it is supposed to connect to (or receive a connection from). An instance is also told its *role* in the session, i.e., whether it is going to *open* itself for connection, or whether it is going to *connect* to another instance.

There is also an *adversary* that may perform certain operations, and a *ring master* that handles these operations by generating certain random variables and enforcing certain global consistency constraints. Some operations result in a record being placed in a *transcript*.

The ring master keeps track of session keys $\{K_{ij}\}$ that are set up among user instances (as will be explained below, the key of an instance is set when that instance starts a session). In addition, the ring master has access to a random bit string R of some agreed-upon length (this string is not revealed to the adversary). We will refer to R as *the environment*. The purpose of the environment is to model information shared by users in higher-level protocols.

We will denote a password shared between users A and B as $\pi[A, B]$.

The adversary may perform the following operations: (1) *initialize user* operation with a new user number i and a new identifier ID_i as parameters; (2) *set password* with a new user number i, a new identifier ID', and a password π as parameters (modeling the adversary creating his own account); (3) *initialize user instance* with parameters including a user instance (i, j), its role, and a user identifier denoting the partner with whom it wants to connect; (4) *terminate user instance* with a user instance (i, j) as a parameter; (5) *test instance password* with a user instance (i, j) and a password guess π as parameters (see restrictions below); (6) *start session* with a user instance (i, j) as a parameter (modeling the user instance successfully connecting to its partner and establishing a random session key; (7) *application* with a function f as parameter, and returning the function f applied to the environment and any session keys that have been established (modeling leakage of session key information in a real protocol through the use of the key in, for example, encryptions of messages); (8) *implementation*, with a comment as parameter (modeling real world queries that are not needed in the ideal world).

For an adversary $\mathcal{A}^*$, $IdealWorld(\mathcal{A}^*)$ is the random variable denoting the transcript of the adversary's operations.

In the model of [13], the adversary is restricted in terms of the *test instance password* query. Specifically, it is only allowed to ask the query once per instance, thus modeling the adversary guessing a password and attempting to authenticate herself. In our model, we allow the adversary to ask the query *twice* per instance. This models the adversary being able to test two passwords during a single authentication attempt. While this is strictly weaker than the original model of [13], it still provides good practical security and prevents off-line dictionary attacks. As discussed above, it also accurately models the actual ability of an adversary in the SRP protocol [32].

For a detailed description of the syntax and semantics of the above operations, see [13].

2.2 Real System

In the real system, users and user instances are denoted as in the ideal system. User instances are defined as state machines with implicit access to the user's ID, PID, and password (i.e., user instance (i, j) is given access to $\pi[ID_i, PID_{ij}]$). User instances also have access to private random inputs (i.e., they may be randomized). A user instance starts in some initial state, and may transform its state only when it receives a message. At that point it updates its state, generates a response message, and reports its status, either *continue*, *accept*, or *reject*, with the following meanings:

- *continue*: the user instance is prepared to receive another message.
- *accept*: the user instance (say (i, j)) is finished and has generated a session key K_{ij}.
- *reject*: the user instance is finished, but has not generated a session key.

The adversary may perform the following types of operations: (1) *initialize user* operation as in the ideal system; (2) *set password* operation as in the ideal system; (3) *initialize user instance* as in the ideal system; (4) *deliver message* with an input message m and a user instance (i, j) as parameters, and returning the message output from (i, j) upon receiving m; (5) *random oracle* with the random oracle index i and input value x as parameters, and returning the result of applying random oracle H_i to x; (6) *application* as in the ideal system.

For an adversary $\mathcal{A}$, *RealWorld*$(\mathcal{A})$ denotes the transcript of the adversary's operations.

Again, details of these operations can be found in [13].

2.3 Definition of Security

Our definition of security is the same as the one in [30] for key exchange. It requires

1. **completeness**: for any real world adversary that faithfully delivers messages between two user instances with complimentary roles and identities, both user instances accept; and
2. **simulatability**: for every efficient real world adversary $\mathcal{A}$, there exists an efficient ideal world adversary $\mathcal{A}^*$ such that *RealWorld*$(\mathcal{A})$ and *IdealWorld*$(\mathcal{A}^*)$ are computationally indistinguishable.

3 SPEKE Protocol

3.1 Preliminaries

Let κ and ℓ denote our security parameters, where κ is the "main" security parameter and can be thought of as a general security parameter for hash functions and secret keys (say 128 or 160 bits), and $\ell > \kappa$ can be thought of as a security parameter for discrete-log-based public keys (say 1024 or 2048 bits). Let $\{0,1\}^*$ denote the set of finite binary strings and $\{0,1\}^n$ the set of binary strings of length n. A real-valued function $\epsilon(n)$ is *negligible* if for every $c > 0$, there exists $n_c > 0$ such that $\epsilon(n) < 1/n^c$ for all $n > n_c$.

Let q of size at least ℓ and be a prime such that $p = 2q + 1$ is also prime. Let g be a generator Z_p^*, and let $h = g^2$ be a generator of the subgroup of Z_p^* of size q. Call this subgroup $G_{p,q}$. We will often omit " $\bmod p$" from expressions when it is obvious that we are working in Z_p^*.

Let $\mathrm{DH}(X, Y)$ denote the Diffie-Hellman value h^{xy} of $X = h^x$ and $Y = h^y$. Let $\mathrm{DH}_{h'}(X, Y)$ denote the Diffie-Hellman value $(h')^{xy}$ of $X = (h')^x$ and $Y = (h')^y$. Let $\mathrm{IDH}(X, Y)$ denote the Inverted Additive Diffie-Hellman value $h^{(x+y)^{-1}}$ of $X = h^{x^{-1}}$ and $Y = h^{y^{-1}}$. We assume the hardness of the *Decisional Inverted-Additive Diffie-Hellman problem* (DIDH) in $G_{p,q}$, which implies the hardness of the *Computational Inverted-Additive Diffie-Hellman problem* (CIDH) in $G_{p,q}$, which

$$\begin{array}{lcr}
A & & B \\
x \in_R Z_q & & \\
m = ((H_1(A,B,\pi))^2)^x & \xrightarrow{\quad m \quad} & \text{Test } m \overset{?}{\not\equiv} 0, 1, -1 \bmod p \\
 & & y \in_R Z_q \\
 & & \mu = ((H_1(A,B,\pi))^2)^y \\
 & & \sigma = (m^2)^y \\
\sigma = (\mu^2)^x & \xleftarrow{\quad \mu, k \quad} & k = H_{2a}(A,B,m,\mu,\sigma,\pi) \\
\text{Test } k \overset{?}{=} H_{2a}(A,B,m,\mu,\sigma,\pi) & & \\
k' = H_{2b}(A,B,m,\mu,\sigma,\pi) & & \\
K = H_3(A,B,m,\mu,\sigma,\pi) & \xrightarrow{\quad k' \quad} & \text{Test } k' \overset{?}{=} H_{2b}(A,B,m,\mu,\sigma,\pi) \\
 & & K = H_3(A,B,m,\mu,\sigma,\pi)
\end{array}$$

Figure 1: The SPEKE protocol, with $\pi = \pi[A, B]$. The resulting session key is K. If a "Test" returns false, the protocol is aborted.

in turn implies the hardness of the *Computational Diffie-Hellman problem* (CDH) in $G_{p,q}$ (as shown in Section 5.2).

One formulation of CDH is that given h, X, Y in $G_{p,q}$, where $X = h^x$ and $Y = h^y$ are chosen randomly, compute $Z = \mathrm{DH}(X, Y)$ with non-negligible probability.

One formulation of CIDH is that given h, X, Y in $G_{p,q}$, where $X = h^{x^{-1}}$ and $Y = h^{y^{-1}}$ are chosen randomly, compute $Z = \mathrm{IDH}(X, Y)$ with non-negligible probability.

One formulation of DIDH is that given h, X, Y, Z in $G_{p,q}$, where $X = h^{x^{-1}}$ and $Y = h^{y^{-1}}$ are chosen randomly, and Z is either $\mathrm{IDH}(X, Y)$ or random, each with half probability, determine if $Z = \mathrm{IDH}(X, Y)$. Breaking DIDH implies a constructing a polynomial-time adversary that distinguishes $Z = \mathrm{IDH}(X, Y)$ from a random Z with non-negligible advantage over a random guess.

3.2 The Protocol

Define hash functions $H_{2a}, H_{2b}, H_3 : \{0,1\}^* \to \{0,1\}^\kappa$ and $H_1 : \{0,1\}^* \to \{0,1\}^\eta$ (where $\eta \geq \ell + \kappa$). We will assume that H_1, H_{2a}, H_{2b}, and H_3 are independent random functions. Note that while H_1 is described as returning a bit string, we will operate on its output as a number modulo p.

In Figure 1 we give a fully specified instantiation of the SPEKE protocol based on the description in Jablon [21]. For the "proof of knowledge" of the session key, we use the simple technique from PAK [13] of sending a hash value of the Diffie-Hellman secret shared value. For the function of the password to be used as the base of the Diffie-Hellman exponentiations, we use a "full-domain" hash of the password.

Theorem 1. *The SPEKE protocol (as instantiated in this paper) is a secure password-authenticated key exchange protocol in the explicit-authentication model under the Decision Inverted-Additive Diffie-Hellman assumption.*

The proof is given in Section 4.

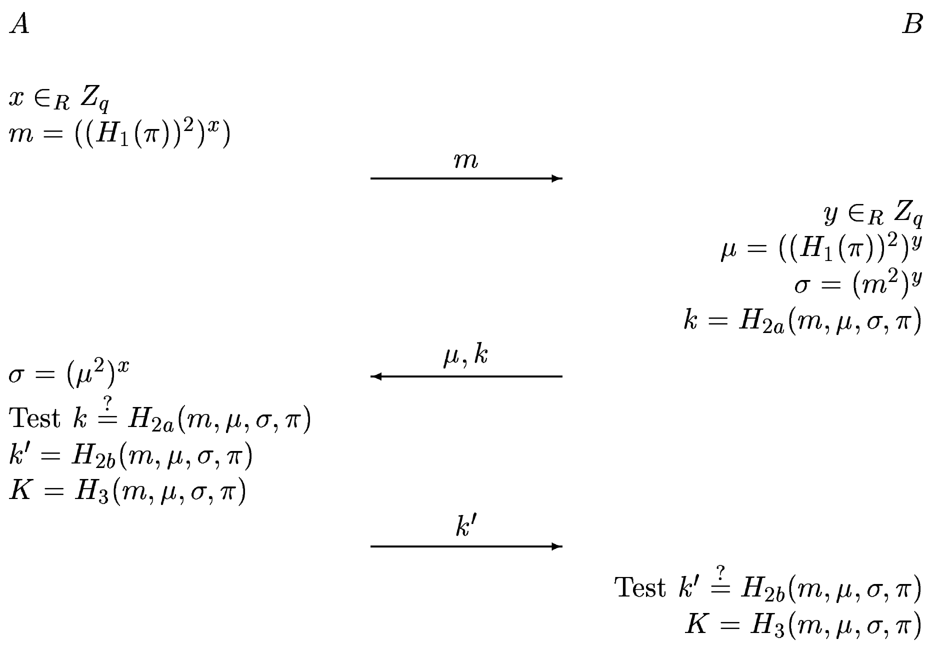
A
B
$x \in_R Z_q$
$m = ((H_1(\pi))^2)^x)$
m
$y \in_R Z_q$
$\mu = ((H_1(\pi))^2)^y$
$\sigma = (m^2)^y$
$k = H_{2a}(m, \mu, \sigma, \pi)$
μ, k
$\sigma = (\mu^2)^x$
Test $k \stackrel{?}{=} H_{2a}(m, \mu, \sigma, \pi)$
$k' = H_{2b}(m, \mu, \sigma, \pi)$
$K = H_3(m, \mu, \sigma, \pi)$
k'
Test $k' \stackrel{?}{=} H_{2b}(m, \mu, \sigma, \pi)$
$K = H_3(m, \mu, \sigma, \pi)$

Figure 2: The Simplified SPEKE protocol.

4 Security of the SPEKE Protocol

The completeness requirement follows directly by inspection. Here we prove that the simulatability requirement holds. The basic technique is essentially that of Shoup [30]. The idea is to create an ideal world adversary $\mathcal{A}^*$ by running the real world adversary $\mathcal{A}$ against a simulated real system, which is built on top of the underlying ideal system. In particular, $\mathcal{A}^*$ (i.e., the simulator combined with $\mathcal{A}$) will behave in the ideal world just like $\mathcal{A}$ behaves in the real world, except that idealized session keys will be used in the real world simulation instead of the actual session keys computed in the real system.

Thus our proof consists of constructing a simulator (that is built on top of an ideal system) for a real system so that the transcript of an adversary attacking the simulator is computationally indistinguishable from the transcript of an adversary attacking the real system.

Let T be the running time of the adversary. (We will also use this as a bound on the number of operations the adversary performs in the real system.) T must be polynomial in the security parameter κ. W.o.p. stands for "with overwhelming probability," i.e., with probability at least $1 - \epsilon$, for some ϵ which is negligible in the security parameter κ.

4.1 The Simulator

The general idea of our simulator is to try to detect guesses on the password (by examining the adversary's random oracle queries) and turn them into *test instance password* queries. If the simulator does not notice a password guess, then it either sets up a connection between two instances (if all the messages between them have been correctly relayed), or rejects (otherwise).

The main difficulty in constructing the simulator is that we need to simulate the protocol without knowing the actual passwords. We solve this problem as follows: It may be seen that the password only appears in the protocol as an argument to random oracle queries. Now, whenever the actual protocol would use the result of a random oracle query whose arguments involve the password, we will simply substitute a random value for the oracle's response. We can think of this as an "implicit" oracle call, i.e., one where we know the value returned, even though we don't (as of yet, at least) know the arguments. In handling the adversary's explicit random oracle queries, as well as those protocol operations that use random oracles, we need to make sure that we don't use inconsistent values for the result of a random oracle on a certain input. In particular, we need to be able to detect if an adversary's query to a random oracle might match a prior implicit oracle call. We will say that an oracle query is *shadowed* by an implicit oracle query if the two queries would be equal for some feasible (i.e., not yet ruled out) value of the password.

In the process of describing the simulator, we will show that the transcript of the simulation in the ideal world is computationally indistinguishable from the transcript of the actual adversary in the real world, step by step, unless is is possible to construct an algorithm to break the DIDH problem.

Let us now proceed with the technical details. Say an *initiator* is an instance that sends the first message in the protocol, and a *responder* is an instance that sends the second message in the protocol. We will always write (i, j) for the user instance that is an initiator, and (i', j') for the user instance that is a responder. Let A (resp. B) be the ID of the current initiator (resp. responder), i.e., either ID_i or $PID_{i'j'}$ (resp. PID_{ij} or $ID_{i'}$), depending on the context. Let $\pi^* = \pi[A, B]$. An *acceptable partner* for an instance (i, j) (resp. (i', j')) is any instance (i', j') (resp. (i, j)) with $PID_{i'j'} = ID_i$ and $PID_{ij} = ID_{i'}$ (resp. $PID_{ij} = ID_{i'}$ and $PID_{i'j'} = ID_i$). We say that two instances had a *matching conversation* if all the messages sent by one were received by the other (preserving order) and vice versa. (This definition is as in [4], except that we don't care about the timings, i.e., we don't rule out the possibility of a message being received before it is sent. However, we will see that the probability of this is negligible in our protocol. Also note that, as opposed to [4], matching conversations are not used in our definition of security, but only as a notion to better illustrate our proof.) When we say that an instance in the open role has had a matching conversation with an instance in the connect role, this does not make any requirements on the last message (i.e., the last message is not required to have been delivered properly). However, in order for an instance in the connect role to have a matching conversation with an instance in the open role, all the messages need to be delivered properly.

We now describe the actions of the simulator for each possible operation of the adversary in the real protocol. An *initialize user instance* or an *application* operation is simply passed on to the ideal system. A *set password* operation is also passed through to the ideal system (and the password is recorded by the simulator). A *deliver message* operation is dealt with depending on the state of the user instance involved. This state includes the role of the user instance, and the previous messages to and from that user instance. A *random oracle* operation is answered depending on which random oracle is queried. The responses to *deliver message* and *random oracle* operations are specified below.

We name the actions of user instances on *deliver message* operations as follows:

A0 Initiator instance action to start the protocol (i.e., an A0 action for an initiator instance is generating and sending m).

B1 Responder instance action upon receiving the first message (i.e., m).

A2 Initiator instance action upon receiving the message from the responder (i.e., (μ, k)).

B3 Responder instance action upon receiving the second message from the initiator (i.e., k').

For example, B1(m) denotes an adversary's *deliver message* operation on some responder instance, with *InMsg* being m.

We now describe some general rules for handling *deliver message* operations: We discard all improper messages to user instances, just as would be done in the real system. These messages may be improper because, for instance, they are not formatted correctly, or the user identities do not match. If the partner ID of a user instance is not set to an identity of an initialized user, then the original protocol is followed, and the *expose* connection assignment is used for a *start session* operation (this is possible, since the simulator has seen the necessary password in the *set password* operation).

Here are some general rules for handling *random oracle* operations: The simulator keeps a record of all random oracle query-response pairs (of course, this is only done for explicit oracle queries). If a query made to a random oracle matches a previous query to the same random oracle, the stored response is returned. If a random oracle is given user identities A and B as arguments, and either (or both) of A and B is not the identity of a valid user, then the query is answered with a random string (as in the real system).

Detailed descriptions of how the simulator responds to *deliver message* and *random oracle* operations (that do not fall under the above rules) follow:

1. $H_1(A, B, \pi)$

   Generate $\alpha[A, B, \pi] \in_R Z_{2q}$ and store it. Generate $\beta \in_R Z_{\lfloor 2^\eta/p \rfloor}$, and return $(g^{\alpha[A,B,\pi]} \bmod p) + \beta p$. Note that this will be indistinguishable from a random bit string of length η, since $g^{\alpha[A,B,\pi]} \bmod p$ is a random element from Z_p^* and $\frac{2^\eta \bmod p}{2^\eta}$ is negligible.

2. $H_{2a}(A, B, m, \mu, \sigma, \pi)$

   Case 1 Some prior B1 query has recorded a tuple of the form $(i', j', m, \mu, \mu', k_{i'j'})$, with $A = PID_{i'j'}$ and $B = ID_{i'}$, for some values of μ' and $k_{i'j'}$. This implies that a B1(m) query was made to (i', j') and returned $(\mu, k_{i'j'})$, but no A0 query to an acceptable partner returned m—see the B1 query description below. (In other words, this H_{2a} query might be shadowed by the implicit H_{2a} query from case 2 of the B1 action.)

   W.o.p., there will be at most one such tuple, since μ values stored in these tuples are random and independent. If $H_1(A, B, \pi)$ has been asked, and $(m^2)^{\mu'/\alpha[A,B,\pi]} = \sigma$, then:

   (a) If there has been a successful guess on $\{A, B\}$ and π was the password in that guess, call this H_{2a} query a *successful H_{2a} query on (i', j')*.

   (b) If there hasn't been a successful guess on $\{A, B\}$, and there never was an unsuccessful guess on $\{A, B\}$ for π:

      i. If at most one *test instance password* operation has been performed on (i', j'), perform another *test instance password* operation with arguments (i', j') and π. If the test is successful, we also call this query a *successful H_{2a} query on (i', j')*.

      ii. If at least two *test instance password* operations have previously been performed on (i', j'), then abort. We will call this event *an H_{2a} failure.*

   Finally, if this query is a successful H_{2a} query on (i', j') for some (i', j'), then return $k_{i'j'}$. Otherwise, return $k \in_R \{0, 1\}^\kappa$.

   If no H_{2a} failure occurs as a result of this H_{2a} query, then the simulation will be indistinguishable from the real world, as follows: In the real world the k value sent by (i', j') is $H_{2a}(A, B, m, \mu, (\mathrm{DH}_{(H_1(A,B,\pi^*))^2}(\mu, m^2)), \pi^*)$. Therefore, if this H_{2a} query is successful, it will return a value consistent with the adversary's view. On the other hand, if this query is not successful, then generating a random response is indistinguishable from the adversary's view in the real world.

The following claim shows that w.o.p. $\mathcal{A}$ will not ask an H_{2a} query that causes an H_{2a} failure. (The H_{2b} case in the claim is necessary to handle B3 operations.) This is the only claim that relies on the hardness of DIDH.

Claim 1. *Let μ be returned by a B1(m) query to (i', j'). Let $A = PID_{i'j'}$ and $B = ID_{i'}$. Then w.o.p. it will not happen that the three oracle queries*

$$H_s(A, B, m, \mu, \mathrm{DH}_{(H_1(A,B,\pi_1))^2}(\mu, m^2), \pi_1),$$

and

$$H_t(A, B, m, \mu, \mathrm{DH}_{(H_1(A,B,\pi_2))^2}(\mu, m^2), \pi_2)$$

$$H_u(A, B, m, \mu, \mathrm{DH}_{(H_1(A,B,\pi_3))^2}(\mu, m^2), \pi_3)$$

will be asked, with $s, t, u \in \{2a, 2b\}$, unless either $\pi_1 = \pi_2$, or by the time of the second query there has already been a successful guess on $\{A, B\}$.

Note that in this claim we speak both about queries made by the adversary, and explicit queries made within the simulator (e.g., in case 2 of the A2 operation).

The proof appears in Appendix 4.2.

Note that an H_{2a} query can result in a *test instance password* query on (i', j') only if a tuple $(i', j', m, \mu, \mu', k_{i'j'})$ has previously been recorded. It will become apparent below (see the B1 query description) that, w.o.p., this could only happen if (i', j') has not had a matching conversation with an acceptable partner. We have just proven one part of the following claim:

Claim 2. *If a* test instance password *query is made on a responder instance (i', j'), then (i', j') has not had a matching conversation with an acceptable partner.*

The other part is shown in case 3b of B3 (which is the only other place where a *test instance password* could be made on a responder instance).

Case 2 No tuple of the form $(i', j', m, \mu, y, k_{i'j'})$ has been recorded with $A = PID_{i'j'}$ and $B = ID_{i'}$.

Return $k \in_R \{0,1\}^\kappa$. The only way this response could be distinguishable from the real system is if this query would be shadowed by an implicit H_{2a} query from case 1 of some B1 action, i.e., if m and μ are from a matching conversation of two valid user instances, and $\sigma = \mathrm{DH}_{(H_1(A,B,\pi))^2}(\mu, m^2)$. However, by the following claim, w.o.p. this will not occur.

Claim 3. *Let m be returned by an A0 query to (i, j) with $A = ID_i$ and $B = PID_{ij}$, and μ be returned by a subsequent B1(m) query to (i', j') with $B = ID_{i'}$ and $A = PID_{i'j'}$. Then, w.o.p., the there will never be (neither before nor after the A0 and B1 queries) an oracle query $(A, B, m, \mu, \mathrm{DH}_{(H_1(A,B,\pi))^2}(\mu, m^2), \pi)$ to H_{2a}, H_{2b}, or H_3, for any π.*

The proof of the claim appears in Appendix 4.2. Note that we only consider B1(m) queries subsequent to an A0 query that returns m, since the probability of their occurrence prior to an A0 query that returns m is negligible (by randomness of m).

3. $H_{2b}(A, B, m, \mu, \sigma, \pi)$

   Return $k' \in_R \{0,1\}^\kappa$. This is indistinguishable from the real system by the following argument: The only implicit H_{2b} query that could shadow this one is the query from case 1 of an A2 action. However, that shadowing is, w.o.p., impossible by Claim 3. (Note that the H_{2b} query from case 2 of an A2 action is explicit, i.e., all of its arguments are known, and so a query-response pair would be stored for it.)

4. $H_3(A, B, m, \mu, \sigma, \pi)$

   Return $K \in_R \{0,1\}^\kappa$. As for H_{2b} queries, indistinguishability easily follows by Claim 3.

5. A0 query to (i, j)

   Generate and store $m' \in_R Z_q$, and send $m = (g^2)^{m'}$. Clearly, m is uniformly drawn from $G_{p,q}$, just as in the real system.

6. B1(m) query to (i', j')

   Test if $m \not\equiv 0, 1, -1 \bmod p$. If the test fails, then reject. Otherwise, generate $\mu' \in_R Z_q$ and $k_{i'j'} \in_R \{0,1\}^\kappa$. Send $\mu = (g^2)^{\mu'}$ and $k = k_{i'j'}$. Now consider two cases:

   Case 1 The value of m has been sent by an acceptable partner.
   We can think of $k_{i'j'}$ as the result of an implicit query

   $$H_{2a}(A, B, m, \mu, \mathrm{DH}_{(H_1(A,B,\pi^*))^2}(\mu, m^2), \pi^*).$$

   Since μ is a freshly chosen random number, this implicit query, w.o.p., will not shadow any prior H_{2a} queries (whether explicit or implicit). Consequently, $k_{i'j'}$ will be indistinguishable from the value sent in the real system.

   Case 2 The value of m has not been sent by an acceptable partner.
   In this case, record the tuple $(i', j', m, \mu, \mu', k_{i'j'})$. It is important to note (for Claim 2, that if this tuple is recorded, then, w.o.p., (i', j') will never be considered to have a matching conversation with an acceptable partner, due to the fact that values of m are generated randomly by initiator instances. (Recall that in our definition of "matching conversation" we are not concerned with the timing of sends and receives.)
   As in case 1, we can view $k_{i'j'}$ as the result of an implicit query

   $$H_{2a}(A, B, m, \mu, \mathrm{DH}_{(H_1(A,B,\pi^*))^2}(\mu, m^2), \pi^*).$$

   The value of $k_{i'j'}$ will be indistinguishable from the one sent in the real system for the same reason as before.

7. A2(μ, k) query to (i, j)

   Case 1 (i, j) has had a matching conversation with an acceptable partner (i', j').
   Generate $k'_{ij} \in_R \{0,1\}^\kappa$, send $k' = k'_{ij}$, set status to Accept, and perform a *start session* operation with the connection assignment *open for connection from* (i', j'). This corresponds to the following implicit oracle queries:

   $$\begin{aligned} k'_{ij} &= H_{2b}(A, B, m, \mu, \mathrm{DH}_{(H_1(A,B,\pi^*))^2}(\mu^2, m), \pi^*), \text{ and} \\ K_{ij} &= H_3(A, B, m, \mu, \mathrm{DH}_{(H_1(A,B,\pi^*))^2}(\mu^2, m), \pi^*). \end{aligned}$$

where K_{ij} is the random session key assigned by the ring master at the time of the *start session* operation.

By Claim 3, these implicit queries couldn't shadow any past or future explicit queries. It is also easy to see that these implicit queries will not shadow any other implicit queries, unless there is a collision of m values between two initiator instances (and that only occurs with negligible probability).

The *open* connection assignment will be legal, since the only place that the simulator could perform a *test instance password* operation on (i, j) would be in case 2 of the A2 query (see there). Also, w.o.p., we will never have two initiator instances (i_1, j_1) and (i_2, j_2) that are open for connection from the same responder instance (i', j'), since that would imply that (i_1, j_1) and (i_2, j_2) generated the same m value (otherwise, they couldn't both have had matching conversations with (i', j')).

Case 2 (i, j) has not had a matching conversation with an acceptable partner.

Look for a value of π for which an $H_1(A, B, \pi)$ query has been made, and another query $H_{2a}(A, B, m, \mu, (\mu^2)^{m'/\alpha[A,B,\pi]}, \pi)$ has been made and returned k. (W.o.p. there will be at most one such value of π, due to the randomness of H_{2a}.) If no such π is found, then reject. This is indistinguishable from the real system, since w.o.p. the k received would be incorrect in the real system (i.e., the correct $k = H_{2a}(A, B, m, \mu, \mathrm{DH}_{(H_1(A,B,\pi^*))^2}(\mu^2, m), \pi^*)$ would be independent of the adversary's view).

Otherwise (if such a π is found), perform a *test instance password* operation with arguments (i, j) and π (note that this is the only place where we could perform this operation for an initiator instance, and no session has been started yet, so the operation is legal). If the guess is not successful, then reject (this is indistinguishable from the real system, by an argument similar to the one above). If the guess is successful, then:

(a) Set $k' = H_{2b}(A, B, m, \mu, (\mu^2)^{m'/\alpha[A,B,\pi]}, \pi)$.

(b) Send k'.

(c) Accept.

(d) Set $K = H_3(A, B, m, \mu, (\mu^2)^{m'/\alpha[A,B,\pi]}, \pi)$.

(e) Expose using session key K (this is allowed, since there was a successful guess on the password).

Note that the values of k' and K are computed through explicit oracle queries (we can think of these as subroutine calls within the simulator), and the queries and responses are recorded, as usual. It is clear that the values of k' and K produced in this case are the same as would be computed in the real system.

8. B3(k') query to (i', j')

Case 1 (i', j') has had a matching conversation with an acceptable partner (i, j).

Set status to Accept and perform a *start session* operation with a *connect to* (i, j) connection assignment. This is legal because

(a) w.o.p., the instance (i, j) will still be open for connection by the randomness of μ values sent by responder instances (so that, w.o.p., for each initiator (i, j), there will be at most one (i', j') with which it has had a matching conversation, and so at most one instance will try to connect to (i, j)), and

(b) by Claim 2, there could not have been a *test instance password* query on (i', j'), since (i', j') has had a matching conversation with an acceptable partner.

Case 2 The current value m has been sent by some acceptable partner (i, j), and μ and k have been received by (i, j), but the value of k' that was received has not been sent by (i, j).

Reject. This is indistinguishable from the real system since k' is invalid.

Case 3 Neither Case 1 nor Case 2 holds.

Look for a value of π such that an $H_1(A, B, \pi)$ query has been asked, and a query $H_{2b}(A, B, m, \mu, (m^2)^{\mu'/\alpha[A,B,\pi]}, \pi)$ has been asked and returned k'. (W.o.p. there will be at most one such value of π, due to the randomness of H_{2b}.) If no such π is found, then reject. This is indistinguishable from the real system, since w.o.p. the k' received would be incorrect in the real system (i.e., the correct k' would be independent of the adversary's view). (The only way k' could be correct, other than through a random guess, is if it is the result of an implicit H_{2b} query in case 1 of A2. However, it is easy to see that if that was the case, we couldn't get to case 3 of B3.)

Otherwise (if such a π is found), check if there was a successful guess on $\{A, B\}$.

Case 3a There was a successful guess on $\{A, B\}$. If that guess was not π, then reject.

Case 3b There was no successful guess on $\{A, B\}$. If there already was an unsuccessful guess on $\{A, B\}$ for π, then reject. Otherwise, perform a *test instance password* operation with arguments (i', j') and π. (Note that in this case (i', j') has not had a matching conversation with an acceptable partner. This completes the proof of Claim 2.) If the guess is not successful, then reject.

We can easily see by Claim 1 (using $s = t = 2a$ and $u = 2b$), that, w.o.p., this procedure will not make a *test instance password* query on (i', j') if two have already been made before.

If we haven't rejected yet, then

(a) set status to Accept,

(b) set $K = H_3(A, B, m, \mu, (m^2)^{\mu'/\alpha[A,B,\pi]}, \pi)$, and

(c) expose using session key K (this is allowed, since there was a successful guess on the password).

Note that the value of K is computed through an explicit oracle query. It is clear that the value of K produced in this case are the same as would be computed in the real system.

4.2 Proofs of Claims

Proof of Claim 1. We will call an oracle query of form $H_s(A, B, m, \mu, \sigma, \pi)$, for $s \in \{2a, 2b\}$, "bad" if $\sigma = \mathrm{DH}_{(H_1(A,B,\pi))^2}(\mu, m^2)$.

Suppose that with some nonnegligible probability ϵ there will be some responder instance $(\hat{i}', \hat{j}')$ (with $B = ID_{\hat{i}'}$ and $A = PID_{\hat{i}'\hat{j}'}$) such that the following "bad event" occurs:

1. query B1$(\hat{m})$ is made to $(\hat{i}', \hat{j}')$ and returns $\hat{\mu}$, and

2. at least three "bad" queries are made with $(A, B, \hat{m}, \hat{\mu})$ and distinct values of π, before there is a successful guess on $\{A, B\}$.

We will then show how to construct a distinguisher D for the DIDH problem.

The idea of the distinguisher D is as follows: We start with a triple (X, Y, Z) as input (recall that this can be thought of as $((g^2)^{x^{-1}}, (g^2)^{y^{-1}}, (g^2)^{z^{-1}})$), for which we need to determine whether or not $Z = \mathrm{IDH}(X,Y)$. We will run the adversary against a *simulation* of the original SPEKE simulator. In the simulation, we will use X, Y, and Z in place of some random values. If D runs indistinguishably from the simulator up to the bad event, then we will be able to use the logs to determine whether or not $Z = \mathrm{IDH}(X,Y)$.

More specifically, we will "incorporate" X, Y, and Z into three of the H_1 responses (randomly). A bad query will then correspond to a guess at an m value raised to the inverse of the exponent of X, Y, or Z. For example, assume we respond to the query $H_1(A,B,\pi)$ with $X = (g^2)^{x^{-1}}$. Then the adversary sends a value m to an instance of B, and receives $\mu = (g^2)^{\mu'} = (X^2)^{x\mu'/2}$. Then if π were the correct password, the correct σ value would be

$$\sigma = \mathrm{DH}_{(H_1(A,B,\pi))^2}(\mu, m^2) = (m^2)^{x\mu'/2}.$$

If we are able to determine a guess at σ by the adversary, then by stripping away known values, we could determine a guess at m^x, that is, an arbitrary value chosen by the adversary raised to the inverse of the exponent of X. By determining three guesses on σ for three different passwords, we could then determine the guesses m^x, m^y, and m^z. A simple multiplication would then determine if $z = x + y$, and provide us with an answer for the $DIDH$ problem.

In our distinguisher, we also have to prevent the adversary from using arbitrary σ values that cause the above equation to hold, so we return random powers of X, Y, and Z from the H_1 queries, and strip off the randomness in our computation of m^x, m^y, and m^z. Thus arbitrary σ values would only satisfy the equation with negligible probability. Also, in our distinguisher, the output from the H_1 queries is not necessarily in $G_{p,q}$, but this problem is handled similarly to the normal simulation of H_1 queries.

Note that the reduction argument in this proof is very different from the reduction argument in the proof of security for PAK [13], as evidenced by the fact that it is necessary to obtain three correct guesses by the adversary to solve the underlying DIDH problem, as opposed to two correct guesses to solve the underlying DDH problem in the proof of security for PAK.

Now let us give the details. Our distinguisher D for input (X,Y,Z) runs as follows:

1. Select $d_1, d_2, d_3 \in_R \{1, \ldots, T\}$ If any of d_1, d_2, d_3 are equal, abort.
2. Initialize three lists BAD_1, BAD_2, BAD_3 (initially empty).
3. We will be running the simulator mostly in the normal manner, but playing the ring master also. (That is, we will choose passwords for user pairs, and answer any test password queries.) We make the following changes:
   (a) d_1th, d_2th, or d_3th $H_1(A,B,\pi)$ query: Choose $\beta_1, \beta_2, \beta_3 \in_R Z_{\lfloor 2^\eta/p \rfloor}$, $r_1, r_2, r_3 \in_R Z_q$, and $b_1, b_2, b_3 \in_R \{0,1\}$. For the d_1th $H_1()$ query, answer $X^{r_1} g^{b_1 q} + \beta_1 p$. For the d_2th $H_1()$ query, answer $Y^{r_2} g^{b_2 q} + \beta_2 p$. For the d_3th $H_1()$ query, answer $Z^{r_3} g^{b_3 q} + \beta_3 p$. Note that these could occur in any order depending on the relative order of d_1, d_2 and d_3. If they do not correspond to the same set of users (say A and B), then abort. Say π_1, π_2, and π_3 are the corresponding passwords in the H_1 queries. If any correspond to π^*, abort.
   (b) $\mathrm{A2}(\mu, k)$ query to initiator instance (i,j), where $ID_i = A$ and $PID_{ij} = B$: Behave as in the original simulator, except if we get into case 2, then look for query

$$H_{2a}(A, B, m, \mu, (\mu^2)^{m'/\alpha[A,B,\pi^*]}, \pi^*),$$

   i.e., for any $\pi \neq \pi^*$, simply reject.

Note that we know π^*, and that $(H_1(A,B,\pi^*)) = g^{\alpha[A,B,\pi^*]}$. Thus, the original simulator will work correctly in this case. It is safe to ignore oracle queries with $\pi \neq \pi^*$, since those wouldn't lead to an accept (as the *test instance password* query would fail).

(c) B3(k') query to responder instance (i',j'), where $PID_{i'j'} = A$ and $ID_{i'} = B$: If there is no matching conversation with an acceptable partner, then respond with reject. Otherwise, behave as in the original simulator.

If the "bad event" is about to occur and d_1, d_2, d_3 have been guessed correctly, then this response is appropriate: If there was no matching conversation with an acceptable partner and this query was supposed to result in an accept, then there would be a successful guess on $\{A,B\}$ before the third "bad" oracle query, and that would contradict the definition of the "bad event." On the other hand, if the "bad event" has already occurred, then we don't care whether or not the response was correct (as will be seen below, it is sufficient for us to have the "bad event" at any point in our execution, and we don't need to know the point at which it actually occurred).

(d) (A,B,m,μ,σ,π_i) query to H_{2a}, H_{2b}, or H_3 where a tuple $(i',j',m,\mu,\mu',k_{i'j'})$ was stored in a $B1(m)$ query:

Recall that a tuple will only be stored if $m \not\equiv 0,1,-1 \bmod p$. If $m^2 \equiv X \bmod p$, or $m^2 \equiv Y \bmod p$, or $m^2 \equiv Z \bmod p$, then abort. Otherwise, if $\sigma^2 \not\equiv 0 \bmod p$ and $\sigma^2 \not\equiv 1 \bmod p$, put $(m, (\sigma^2)^{r_i/\mu'})$ in list BAD$_i$. (The value $(\sigma^2)^{r_i/\mu'}$ is basically the adversary's guess of $(m^2)^x$, $(m^2)^y$ or $(m^2)^z$, depending on i. If this guess is correct, then the query is "bad" according to the definition above.) Finally, respond with a random k.

If the bad event is about to occur, then this response will be indistinguishable from the original simulator, since no successful guess on $\{A,B\}$ can occur before the bad event.

At the end of the simulation, check if for any m there are stored values $(m,S_1) \in BAD_1$, $(m,S_2) \in BAD_2$, and $(m,S_3) \in BAD_3$, such that $S_1S_2 = S_3$. If so, answer "True IDH," else answer "Random."

For the first half of the analysis, suppose Z is random. Note that for each σ corresponding to an element placed in BAD_3, $\sigma^2 \in G_{p,q}$ (since $\sigma \not\equiv 0 \bmod p$, $p = 2q+1$ and $\gcd(q,2) = 1$). Also note that $\sigma^2 \not\equiv 1 \bmod p$. Thus, each $(\sigma^2)^{r_3}$ is a random element of $G_{p,q}$. Consequently, since Z is random, the probability that we answer "True IDH" would be at most T^3/q by the union bound,

Now suppose $Z = \text{IDH}(X,Y)$. The probability of aborting because m^2 is equivalent to X, Y, or Z is at most $\frac{3T}{q}$. Now note that since $m \not\equiv 0,1,-1 \bmod p$ for any stored tuple, then $(m^2)^x \not\equiv 0,1 \bmod p$, $(m^2)^y \not\equiv 0,1 \bmod p$, and $(m^2)^z \not\equiv 0,1 \bmod p$. Therefore, no σ with σ^2 equal to 0 or 1 will correspond to a correct guess. Now, if the adversary makes three "bad" queries for the pair of users (A,B) with π_1, π_2 and π_3, then the distinguisher will correctly answer "True IDH" (unless $m^2 \equiv X,Y,Z \bmod p$ for some H_{2a} query). The probability of the "bad event" is ϵ, and the probability of guessing d_1,d_2,d_3 correctly is $\frac{1}{T^3}$.

Now, the probability that the distinguisher D guesses correctly is at least

$$\begin{aligned}
\Pr(D \text{ is correct}) &= \Pr(D \text{ guesses ``IDH True''} | \text{IDH instance}) \Pr(\text{IDH instance}) \\
&\quad + \Pr(D \text{ guesses ``Random''} | \text{Random instance}) \Pr(\text{Random instance}) \\
&\geq \left(\frac{\epsilon}{T^3} - \frac{3T}{q}\right)\left(\frac{1}{2}\right) + \left(1 - \frac{T^3}{q}\right)\left(\frac{1}{2}\right).
\end{aligned}$$

Thus the probability that D is correct is at least $\frac{1}{2} + \frac{\epsilon}{2T^3} - \frac{T^3}{2q} - \frac{3T}{2q}$, which is non-negligibly more than $\frac{1}{2}$. □

Proof of Claim 3. If such a query is indeed made with some non-negligible probability ϵ, then we will construct an algorithm D to solve CDH. Let (X, Y) be the challenge CDH instance.

The idea of the construction is similar to the one in the previous proof. The main difference is that now we "incorporate" X and Y into the m and μ values, respectively, sent in a matching conversation.

The algorithm runs as follows:

1. Generate random d, d', d'' between 1 and T.

2. We will be running the simulator in the normal manner, but playing the ring master also. (That is, we will choose passwords for user pairs, and answer any test password queries.) Run the simulator until the dth A0 query. Say this query is to (i, j). Let $A = ID_i$, $B = PID_{ij}$, and $\pi^* = \pi[A, B]$. Reply to this A0 query with $m = X$.

3. Continue with the simulation, but with the following changes:

   (a) d'th B1(m) query to instance (i', j') where $m = X$, $ID_{i'} = B$, and $PID_{i'j'} = A$:
   Set $\mu = Y$. Compute $k = k_{i'j'}$ as in the original simulator and return (μ, k).

   (b) H_{2a}, H_{2b} or H_3 query $(A, B, X, Y, \sigma, \pi)$ where a query $H_1(A, B, \pi)$ has already been made:

   We guess that the d''th query is the first one in which the adversary gives the correct σ. Thus, on the d''th query, stop the algorithm and guess $Z = \sigma^{\alpha[A,B,\pi]/2}$. Otherwise, return a random number from $\{0, 1\}^\kappa$.

   (c) A2(μ, k) query to (i, j) not part of a matching conversation with an acceptable partner:
   Reject. If the simulator should have accepted, the adversary w.o.p. would have already queried H_{2a} with the correct Diffie-Hellman value.

   (d) B3(k') query to the instance (i', j') (i.e., with $m = X$ and $\mu = Y$), which is not part of a matching conversation:
   Reject. If the simulator should have accepted, the adversary w.o.p. would have already queried H_{2b} with the correct Diffie-Hellman value.

4. If the simulation ends without outputting a value for Z, then output "Failure."

Note that if the adversary does make a bad query, we have probability at least $1/T^3$ of guessing the correct initiator and responder instances, along with the first correct random oracle query with the correct σ value.

Thus the probability that the algorithm D computes DH(X, Y) correctly is at least $\Omega(\frac{\epsilon}{T^3})$, which is non-negligible. □

5 Security properties of DIDH and CIDH

Here we give some results on the hardness of the DIDH and CIDH problems.

5.1 Lower bound for DIDH in the generic model

Shoup [29] proves lower bounds for the Discrete Log, CDH, and DDH problems in the generic model. The same techniques may be used to prove a lower bound for DIDH in the generic model.

First we define some terms as in [29]. An *encoding function* on the additive group Z_q is an injective map $\sigma : Z_q \to \{0,1\}^n$ for some integer $n > 0$. A *generic algorithm* $\mathcal{A}$ for Z_q is a probabilistic algorithm that takes as input an *encoding list* $(\sigma(x_1), \dots, \sigma(x_k))$ where σ is an encoding function and $x_i \in Z_q$ for $i \in \{1, \dots, k\}$. During its execution, the algorithm may query an oracle by giving it two indices i, j into the encoding list and a sign bit. The oracle returns the encoding $\sigma(x_i \pm x_j)$ according to the sign bit. This new encoding is then added to the encoding list. Eventually, the algorithm terminates and produces a certain output. The output is denoted by $\mathcal{A}(\sigma; x_1, \dots, x_k)$.

Now we make a simple observation about $IDH(X,Y)$.

Observation 1. *Given X, Y in $G_{p,q}$ with $X = h^{x^{-1}}$ and $Y = h^{y^{-1}}$, let $a = x^{-1}$ and $b = y^{-1}$. Then $IDH(X,Y) = h^{(x+y)^{-1}} = h^{ab/(a+b)}$.*

Theorem 2. *Let $q \geq 5$ be a prime and $S \subset \{0,1\}^*$ a set of at least q binary strings. Let $\mathcal{A}$ be a generic algorithm for Z_q that makes at most m oracle queries. Let $a, b, c \in Z_q \setminus \{0\}$ with $a + b \neq 0$ be chosen at random, let $\sigma : Z_q \to S$ be a random encoding function, and let s be a random bit. Set $w_0 = ab/(a+b)$ and $w_1 = c$. Then*

$$\left| \Pr(\mathcal{A}(\sigma; 1, a, b, w_s, w_{1-s}) = s) - \frac{1}{2} \right| < \frac{2m^2}{q-4},$$

where the probability is over the random choice of $a, b, c \in Z_q \setminus \{0\}$ with $a + b \neq 0$, the random encoding σ, and the random bits used by the algorithm.

The following proof sketch follows closely the proof sketch of the lower bound for DDH in Boneh [11], which describes the proof of the lower bound for DDH in Shoup [29].

Proof. (Sketch) Consider the m oracle queries made by the algorithm. After each query, the algorithm obtains a new encoding $\sigma(x_i)$ for some $x_i \in Z_q$. Note that one can write x_i as $F_i(a, b, w_s, w_{1-s})$ for some linear function F_i that can be deduced from the oracle queries up to that point. Now, as in Shoup, consider a game in which for all i such that for all $j < i$, $F_i \neq F_j$, the oracle returns a new random encoding σ_i. (Note that if $x_i = x_j$, the actual oracle would return the same encoding, since then $\sigma(x_i) = \sigma(x_j)$.) After the algorithm terminates, it outputs a bit s'. If $s' = s$, or if $x_i = x_j$ for some i, j such that $F_i \neq F_j$, then the algorithm wins. Since the algorithm gets no information from the oracle queries, it wins in this game with probability 1/2 plus the probability that $x_i = x_j$ for some i, j such that $F_i \neq F_j$. Note that the behavior of this game differs from the behavior of the algorithm interacting with the actual oracle only if the algorithm wins, so the probability that the algorithm outputs the correct answer when interacting with the actual oracle is bounded by the probability that the algorithm wins in this game.

Now we bound the probability that $x_i = x_j$ for some i, j such that $F_i \neq F_j$. If this is true, then either $F_i(a, b, ab/(a+b), c) = F_j(a, b, ab/(a+b), c)$ or $F_i(a, b, c, ab/(a+b)) = F_j(a, b, c, ab/(a+b))$. Let E be this event. To bound $\Pr(E)$, first note that there are at most $\binom{m}{2}$ pairs i, j for which $F_i \neq F_j$. For each such pair i, j, the number of solutions to $F_i(x, y, xy/(x+y), z) = F_j(x, y, xy/(x+y), z)$ or $F_i(x, y, z, xy/(x+y)) = F_j(x, y, z, xy/(x+y))$ can be bounded as follows. Let $G_0(x, y, z) = (x+y)[F_i(x, y, xy/(x+y), z) - F_j(x, y, xy/(x+y), z)]$ and $G_1(x, y, z) = (x+y)[F_i(x, y, z, xy/(x+y)) - F_j(x, y, z, xy/(x+y))]$. Both are polynomials of degree 2, and the total number of zeros of G_0 and G_1 gives us a bound on the total number of solutions.

From Lemma 1 of Shoup [29] (see [28]), the probability that a random $(x, y, z) \in Z_q$ is a zero of G_0 is bounded by $2/q$. Thus there are at most $2q^2$ zeros of G_0. Similarly, there are at most

$2q^2$ zeros of G_1, and thus there are at most $4q^2$ total solutions. There are $(q-1)^3 - (q-1)^2$ triples $(x,y,z) \in Z_q \setminus \{0\}$ with $x+y \neq 0$, so the total probability that a random $(x,y,z) \in Z_q \setminus \{0\}$ with $x+y \neq 0$ is a solution of either $F_i(x,y,xy/(x+y),z) = F_j(x,y,xy/(x+y),z)$ or $F_i(x,y,z,xy/(x+y)) = F_j(x,y,z,xy/(x+y))$ is at most

$$\Pr(E) \leq \binom{m}{2} \cdot \frac{4q^2}{(q-1)^3 - (q-1)^2} \leq \frac{2m^2q^2}{q^3 - 4q^2 + 5q - 2} \leq \frac{2m^2q^2}{q^3 - 4q^2} = \frac{2m^2}{q-4}.$$

The theorem follows. □

5.2 Equivalence of CDH and CIDH

It is easy to see that the hardness of DIDH implies the hardness of CIDH. In this section we show the equivalence of the CDH and CIDH assumptions. This result is due to Bleichenbacher [10]. We assume h is a generator of a group of size q.

First we show that given a black box for CDH, we can solve CIDH. Observe that given h^a, using a black box for CDH we can compute $\{h^{a^2}, h^{a^4}, \ldots, h^{a^{\lfloor \log_2 q \rfloor}}\}$, and using at most $\log_2 q$ more applications of CDH compute $h^{a^{-1}} = h^{a^{q-1}}$. Thus we can *invert exponents* using $O(\log_2 q)$ calls to CDH. Now, let $(X,Y) = (h^{x^{-1}}, h^{y^{-1}})$ be a CIDH challenge. We simply use the exponent-inverting algorithm to compute h^x and h^y, multiply them to get h^{x+y}, and use the exponent-inverting algorithm again to compute $h^{(x+y)^{-1}}$, which is $\mathrm{IDH}(X,Y)$.

Now using Observation 1 from Section 5.1, we show that given a black box for CIDH, we can solve CDH. Given a CDH challenge $(X,Y) = (h^x, h^y)$, using the black box for CIDH, compute $IDH(h^{2+x+y}, h^{2-x-y})$ and $IDH(h^{2+x-y}, h^{2-x+y})$. The first result is $h^{(4-x^2-y^2-2xy)/4}$ and the second result is $h^{(4-x^2-y^2+2xy)/4}$. Dividing the second by the first leaves h^{xy}, which is $\mathrm{DH}(X,Y)$.

References

[1] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991.

[2] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 419–428, Dallas, Texas, 23–26 May 1998.

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In EUROCRYPT2000 [16], pages 139–155.

[4] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology—CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 232–249. Springer-Verlag, 22–26 Aug. 1993.

[5] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In CCS'93 [14], pages 62–73.

[6] M. Bellare and P. Rogaway. Provably secure session key distribution—the three party case. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 57–66, Las Vegas, Nevada, 29 May–1 June 1995.

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

[8] S. M. Bellovin and M. Merritt. Augumented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In CCS'93 [14], pages 244–250.

[9] S. Blake-Wilson, D. Johnson, and A. Menezes. Key agreement protocols and their security analysis. In *Sixth IMA Intl. Conf. on Cryptography and Coding*, 1997.

[10] D. Bleichenbacher, 2000. Personal Communication.

[11] D. Boneh. The decision Diffie-Hellman problem. In *Proceedings of the Third Algorithmic Number Theory Symposium*, volume 1423 of *Lecture Notes in Computer Science*, pages 48–63. Springer-Verlag, 1998.

[12] M. Boyarsky. Public-key cryptography and password protocols: The multi-user case. In CCS'99 [15], pages 63–72.

[13] V. Boyko, P. MacKenzie, and S. Patel. Provably-secure password authentication and key exchange using Diffie-Hellman. In EUROCRYPT2000 [16], pages 156–171.

[14] *Proceedings of the First Annual ACM Conference on Computer and Communications Security*, 1993.

[15] *Proceedings of the Sixth Annual ACM Conference on Computer and Communications Security*, 1999.

[16] *Advances in Cryptology—EUROCRYPT '2000*, Lecture Notes in Computer Science. Springer-Verlag, 14–18 May 2000.

[17] L. Gong. Optimal authentication protocols resistant to password guessing attacks. In *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, pages 24–29, 1995.

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

[19] S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. In *Proceedings of the Fifth Annual ACM Conference on Computer and Communications Security*, pages 122–131, 1998.

[20] D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communication Review, ACM SIGCOMM*, 26(5):5–20, 1996.

[21] D. Jablon. Extended password key exchange protocols immune to dictionary attack. In *WET-ICE'97 Workshop on Enterprise Security*, 1997.

[22] S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Proceedings of the Workshop on Security Protocols*, 1997.

[23] P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. In T. Okamoto, editor, *Advances in Cryptology—ASIACRYPT 2000*, Lecture Notes in Computer Science, pages 599–613. Springer-Verlag, Dec. 2000.

[24] P. MacKenzie and R. Swaminathan. Secure network authentication with password information. unpublished, `http://www.bell-labs.com/user/philmac/research/snapi.ps.gz`.

[25] S. Patel. Number theoretic attacks on secure password schemes. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 236–247, 1997.

[26] S. Patel, 1999. Personal communication.

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

[28] J. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

[29] V. Shoup. Lower bounds for discrete logarithms and related problems. In *Advances in Cryptology—EUROCRYPT 97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer-Verlag, 1997.

[30] V. Shoup. On formal models for secure key exchange. In CCS'99 [15], page to appear.

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

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