Provably Secure **P**assword-**A**uthenticated **K**ey Exchange Using Diffie-Hellman[*]

Victor Boyko[†] Philip MacKenzie[‡] Sarvar Patel[§]

September 8, 2000

Abstract

When designing password-authenticated key exchange protocols (as opposed to key exchange protocols authenticated using cryptographically secure keys), one must not allow any information to be leaked that would allow verification of the password (a weak shared key), since an attacker who obtains this information may be able to run an *off-line dictionary attack* to determine the correct password. Of course, it may be extremely difficult to hide all password information, especially if the attacker may pose as one of the parties in the key exchange. Nevertheless, we present a new protocol called PAK which is the first Diffie-Hellman-based password-authenticated key exchange protocol to provide a formal proof of security (in the random oracle model) against both passive and active adversaries. In addition to the PAK protocol that provides mutual *explicit* authentication, we also show a more efficient protocol called PPK that is provably secure in the *implicit*-authentication model. We then extend PAK to a protocol called PAK-X, in which one side (the client) stores a plaintext version of the password, while the other side (the server) only stores a verifier for the password. We formally prove security of PAK-X, even when the server is compromised. Our formal model for password-authenticated key exchange is new, and may be of independent interest.

Keywords: Password authentication, key exchange, Diffie-Hellman.

1 Introduction

Two entities, who only share a password, and who are communicating over an insecure network, want to authenticate each other and agree on a large session key to be used for protecting their subsequent communication. This is called the *password-authenticated key exchange* problem. If one of the entities is a user and the other is a server, then this can be seen as a problem in the area of *remote user access*. Many solutions for remote user access rely on cryptographically secure keys, and consequently have to deal with issues like key management, public-key infrastructure, or secure hardware. Many solutions that are password-based, like telnet or Kerberos, have problems that range from being totally insecure (telnet sends passwords in the clear) to being susceptible to certain types of attacks (Kerberos is vulnerable to off-line dictionary attacks [40]).

[*] An abridged version of this paper appeared in Eurocrypt 2000

[†] MIT

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

[§] Bell Laboratories, Lucent Technologies sarvar@lucent.com

Over the past decade, many password-authenticated key exchange protocols that promised increased security have been developed, e.g., [8, 9, 22, 21, 37, 25, 26, 29, 39, 34].[1] Some of these have been broken [31, 32], and, in fact, only two very recent ones have been formally proven secure. The SNAPI protocol in [30] is proven secure in the random oracle model,[2] assuming the security of RSA (and also Decision Diffie-Hellman,[3] when perfect forward secrecy is desired). The simple and elegant protocol in [3] is proven as secure as Decision Diffie-Hellman in a model that includes random oracles and ideal block ciphers. (Our work was performed independently of [3]. In fact, the conference version of our results [13] appears in the same proceedings as [3].)

We present a new password-authenticated key exchange protocol called **PAK** (Password-Authenticated Key exchange), which we prove to be as secure as Decision Diffie-Hellman in the random oracle model. Compared to the protocol of [30], PAK (1) does not require the RSA assumption for security, (2) is more efficient in terms of the number of rounds, and (3) is conceptually simpler, with a simpler proof. Compared to the protocol of [3], PAK does not require an ideal block cipher assumption for security, but has a more complicated proof. (We note that the ideal block cipher assumption is used much less often in the literature than the random oracle assumption.) We also show how the security of PAK can be related to the Computational Diffie-Hellman problem, although with weaker security bounds.

In addition to PAK, we also show a more efficient protocol called PPK (Password Protected Key exchange) that is provably secure in the implicit-authentication model. The PPK protocol only requires 2 rounds of communication.

We then extend PAK to a protocol called PAK-X, in which one side (the client) stores a plaintext version of the password, while the other side (the server) only stores a verifier for the password. We formally prove security of PAK-X, even when the server is compromised. Security in this case refers to an attacker not being able to pose as a client after compromising the server; naturally, it would be trivial to pose as the server.

Our formal model for password-authenticated key exchange is new, and may be of independent interest. It is based on the formal model for secure key exchange by Shoup [35] (which follows the work of [2]), enhanced with notions of password authentication security from [23, 30]. This model is based on the multi-party simulatability tradition (e.g. [1]), in which one first defines an ideal system that models, using a trusted center, the service to be performed (in this case, password-authenticated key exchange), and then one proves that the protocol running in the real world is essentially equivalent to that ideal system.

2 Background

2.1 User Authentication

Techniques for user authentication are broadly based on one or more of the following categories: (1) what a user knows, (2) what a user is, or (3) what a user has. Passwords or PINs are example of the first category. Biometric techniques, such as analysis of voice, fingerprints, retinal

[1] We will discuss hybrid protocols (i.e., password-based protocols in which a server public key is also known to the user) in Section 2.2.

[2] The random oracle model was introduced in [5]. Many popular protocols have been proven secure in that model, including Optimal Asymmetric Encryption Padding (OAEP) [6]. It would certainly be desirable to have a security proof using only standard cryptographic assumptions [14], but, so far, no protocol (and in particular, no efficient protocol) is known for the password authentication problem that is provably secure in the standard model.

[3] The hardness of the Decision Diffie-Hellman problem is essentially equivalent to the semantic security of the ElGamal encryption scheme [18]. See Boneh [11] for more information on this problem.

scans, or keystrokes, fit in the second category. Identification tokens, such as smart cards, fit in the third category. Techniques involving biometric devices tend to be cost-prohibitive, while techniques involving smart cards tend to be both expensive and relatively inconvenient for users. The least expensive and most convenient solutions for user authentication have been based on the first category, of "what a user knows," and that is what we will focus on in this work.

In fact, we will focus on the harder problem of *remote user authentication*, in which not only must the basic authentication techniques be secure, but the protocol that communicates the authentication data across the network must also be secure. The need for remote user authentication is greatly increasing, due mainly to the explosive growth of the Internet and other types of networks, such as wireless communication networks. In any of these environments, it is safest to assume that the underlying links or networks are insecure, and we should realistically expect that a powerful adversary would be capable of eavesdropping on legitimate sessions, deleting and inserting messages into those sessions, and even initiating sessions herself.

Now let us consider the question: "What can a user know?" It is common knowledge that users cannot remember long random numbers, hence if the user is required to know a large secret key (either a large symmetric key or a private key corresponding to a public key), then these keys will have to be stored on the user's system. Furthermore, keeping these secret requires an extra security assumption and introduces a new point of weakness. Even if a user is required to know some public but non-generic data, like the server's public key, this must be stored on the user's system and requires an extra assumption that the public key cannot be modified. In either case, (1) there is a significant increase in administration overhead because both secret and public keys have to be generated and securely distributed to the user's system and the server, and (2) this would not allow for users to walk up to a generic station that runs the authentication protocol and be able to perform secure remote authentication to a system that was previously unknown to that station (such as, perhaps, the user's home system).

To solve these problems one may wish to use a trusted third party, either on-line (as in Kerberos) or off-line (i.e., a certification authority). However, the fact that the third party is "trusted" implies another security requirement. Also, the users or servers must at some point interact with the third party before they can communicate remotely, which increases the overhead of the whole system. Naturally, if an organized and comprehensive PKI emerges, this may be less of a problem. Still, password-only protocols seem very inviting because they are based on direct trust between a user and a server, and do not require the user to store long secrets or data on the user's system. They are thus cheaper, more flexible, and less administration-intensive. They also allow for a generic protocol which can be pre-loaded onto users' systems.

2.2 Password-Authentication Protocols

Many existing password authentication protocols, like `telnet` and `ftp`, send the password in the clear and are thus vulnerable to eavesdroppers.[4] Sending the password in the clear can be avoided by using more sophisticated schemes, such as one-time passwords systems (e.g., S/KEY [24]), or simple challenge-response schemes (e.g., CHAP [36]). However, these protocols are susceptible to *off-line dictionary attacks*: Many users choose passwords of relatively low entropy, so it is possible for the adversary to compile a dictionary of possible passwords.[5] Obviously, we can't prevent the adversary from trying all the passwords on-line, but such an attack can be made infeasible by

[4] Hashing the password does not offer more protection.

[5] See, for example, the experiment performed by Wu [40]. Briefly, using an off-line dictionary attack, he was able to discover 2045 passwords in a realm of slightly over twenty-five thousand users by verifying about 100 million candidate passwords.

simply placing a limit on the number of unsuccessful authentication attempts. On the other hand, an off-line search through the dictionary is quite doable. Here is an example of an off-line dictionary attack against a simple challenge-response protocol: The adversary overhears a challenge R and the associated response $f(P, R)$ that involves the password. Now she can go off-line and run through all the passwords P' from a dictionary of likely passwords, comparing the value $f(P', R)$ with $f(P, R)$. If one of the values matches the response, then the true password has been discovered.

A decade ago, Lomas et.al. [28] presented the first protocols which were resistant to these types of off-line dictionary attacks. The protocols assumed that the client had the server's public key and thus were not strictly password-only protocols. Gong et.al. [22] later presented their versions of these types of protocols. Recently, Halevi and Krawczyk [23] presented protocols and formal proofs of security for the same scenario as addressed by [28], i.e., where the client authenticates with a password and the server with a public key. Boyarsky [12] has addressed some problems with the Halevi-Krawczyk protocols in the multi-user scenario.

The EKE protocol [8] was the first password-authenticated key exchange protocol that did not require the user to know the server's public key. The idea of EKE was to use the password to symmetrically encrypt the protocol messages of a standard key exchange (e.g., Diffie-Hellman [17]). Then an attacker making a password guess could decrypt the symmetric encryption, but could not break the asymmetric encryption in the messages, and thus could not verify the guess. Following EKE, many protocols for password-authenticated key exchange were proposed which did not require the user to know the server's public key [9, 22, 21, 37, 25, 26, 29, 39]. Some of these protocols were, in addition, designed to protect against server compromise, so that an attacker that was able to steal data from a server could not later masquerade as a user without having performed a dictionary attack.[6] All of these protocol proposals contained informal arguments for security. However, the fact that some of these protocols were subsequently shown to be insecure [31, 32] should emphasize the importance of formal proofs of security.

2.3 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 [7] they extend it to the three party case. Blake-Wilson et.al. [10] further extend the model to cover the asymmetric setting. Independently, [30] and [3] present extensions to the model to allow for password authentication. Halevi and Krawczyk [23] 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 [35] refines and extends their model. We present a further extension of [35] that includes password authentication.

3 Model

For our proofs, we extend the formal notion of security for key exchange protocols from Shoup [35] to password-authenticated key exchange. We will prove security against a static adversary, i.e., one whose choice of whom to corrupt is independent of its view while attacking the protocol. We assume the adversary totally controls the network, a la [4].

[6] Naturally, given the data from a server, an attacker could perform an off-line dictionary attack, since the server must know something that would allow verification of a user's password.

Security for key exchange in [35] is defined using an ideal system, which describes the service (of key exchange) that is to be provided, and a real system, which describes the world in which the protocol participants and adversaries work. The ideal system should be defined such that an "ideal world adversary" cannot (by definition) break the security. Then, intuitively, a proof of security would show that anything an adversary can do 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.

3.1 Definition of Security

The definition of security for key exchange given in [35] 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 $\mathit{RealWorld}(\mathcal{A})$ and $\mathit{IdealWorld}(\mathcal{A}^*)$ are computationally indistinguishable (here $\mathit{RealWorld}(\cdot)$ and $\mathit{IdealWorld}(\cdot)$ refer to real and ideal world transcripts, respectively, and are defined in Sections 3.3 and 3.2).

We will use this definition for password-authenticated key exchange as well, with no modifications. We can do this because, as will be seen below, our ideal model includes passwords explicitly. If it did not, we would have to somehow explicitly state the probability of distinguishing real world from ideal world transcripts, given how many impersonation attempts the real world adversary made.

We now proceed with a description of the Ideal System for password authentication. The parts that are not directly related to passwords are taken from [35], except for a slight modification to handle mutual authentication.

3.2 Ideal System

The basic assumption of the ideal system is that the key exchange is done by a trusted third party (called *the ring master*): The ring master generates all the keys, and delivers them to the users. The communication between the users and the ring master is presumed to be perfectly secure. Thus, the key exchange in the ideal system is perfectly secure.

Let us now present the details. 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.

Since we deal with password authentication, and because passwords are not cryptographically secure, our system must somehow allow a non-negligible probability of an adversary successfully impersonating an honest user. We do this by including passwords explicitly in our model. We let π denote the function assigning passwords to pairs of users. To simplify notation, we will write $\pi[A, B]$ to mean $\pi[\{A, B\}]$ (i.e., $\pi[A, B]$ is by definition equivalent to $\pi[B, A]$).

The adversary may perform the following types of operations:

initialize user [Transcript: (`"initialize user"`, i, ID_i)]

The adversary assigns identity string ID_i to (new) user i. In addition, a random password $\pi[ID_i, ID_{i'}]$ is chosen by the ring master for each existing user i'. The passwords are not placed in the transcript. This models the out-of-band communication required to set up passwords between users.

set password [Transcript: (`"set password"`, i, ID', π)]

The identity ID' is required to be new, i.e., not assigned to any user. This sets $\pi[ID_i, ID']$ to π and places a record in the transcript.

After ID' has been specified in a *set password* operation, it cannot be used in a subsequent *initialize user* operation.

initialize user instance [Transcript: (`"init. user inst."`, i, j, $role(i, j)$, PID_{ij})]

The adversary assigns a user instance (i, j) a role (one of $\{\text{open}, \text{connect}\}$) and a user PID_{ij} that is supposed to be its partner. If PID_{ij} is not set to an identity of an initialized user, then we require that a *set password* operation has been previously performed for i and PID_{ij} (and hence there can be no future *initialize user* operation with PID_{ij} as the user ID).

terminate user instance [Transcript: (`"terminate user instance"`,i,j)]

The adversary specifies a (previously initialized) user instance (i, j) to terminate.

test instance password

This is called with an instance (i, j) and a password guess π. The adversary queries if $\pi = \pi[ID_i, PID_{ij}]$. If this is true, the query is called a *successful guess on* $\{ID_i, PID_{ij}\}$ (note that a successful guess on $\{A, B\}$ is also a successful guess on $\{B, A\}$).

This query may only be asked once per user instance. The instance has to be initialized and not yet engaged in a session (i.e., no *start session* operation has been performed for that instance). Note that the adversary is allowed to ask a *test instance password* query on an instance that has been terminated.

This query does not leave any records in the transcript.

start session [Transcript: (`"start session"`, i, j)]

The adversary specifies that a session key K_{ij} for user instance (i, j) should be constructed. The adversary specifies which *connection assignment* should be used. There are three possible connection assignments, as shown in Table 1.

Note that the connection assignment is not recorded in the transcript.

application [Transcript: (`"application"`, f, $f(R, \{K_{ij}\})$)]

The adversary is allowed to obtain any information she wishes about the environment and the session keys. (This models leakage of session key information in a real protocol through the use of the key in, for example, encryptions of messages.) The function f is specified by the adversary and is assumed to be efficiently computable.

<table>
<tr>
<td>1. <b>open for connection from</b> (i', j'). This requires that
• <i>role</i>(i, j) is “open,”
• (i', j') has been initialized and has not been terminated,
• <i>role</i>(i', j') is “connect,”
• $PID_{ij} = ID_{i'}$,
• $PID_{i'j'} = ID_i$,
• no other instance is open for connection from (i', j'), and
• no <i>test instance password</i> operation has been performed on (i, j).
The ring master generates K_{ij} randomly. We now say that (i, j) is open for connection from (i', j').</td>
<td>2. <b>connect to</b> (i', j'). This requires that
• <i>role</i>(i, j) is “connect,”
• (i', j') has been initialized and has not been terminated,
• <i>role</i>(i', j') is “open,”
• $PID_{ij} = ID_{i'}$,
• $PID_{i'j'} = ID_i$,
• (i', j') was open for connection from (i, j) after (i, j) was initialized, and
• no <i>test instance password</i> operation has been performed on (i, j).
The ring master sets $K_{ij} = K_{i'j'}$. We now say that (i', j') is no longer open for connection.</td>
</tr>
<tr>
<td colspan="2">3. <b>expose.</b> This requires that either PID_{ij} has not been assigned to an identity of an initialized user, or there has been a successful guess on $\{ID_i, PID_{ij}\}$. The ring master sets K_{ij} to the value specified by the adversary.</td>
</tr>
</table>

Table 1: Valid connection assignments for the *start session* operation

implementation [Transcript: (`"impl"`, *cmnt*)]

The adversary is allowed to put in an "implementation comment" which does not affect anything else in the ideal world. This will be needed for generating ideal world views that are equivalent to real world views, as will be discussed later.

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

Discussion (general key exchange): Because keys exchanged between two honest users are not transmitted during the *start session* operations, it should be clear that key exchange is completely secure in the ideal world. Naturally, *application* operations may reveal keys. This models the use of the keys in a higher-level protocol. What we require from a secure key exchange protocol is that even given some partial information about the keys, the adversary shouldn't be able to do anything in the real world that he couldn't do in the ideal world. In other words, if a higher-level protocol is secure (in some appropriate sense) in an ideal system with keys generated by the ring master, that protocol should also be secure if we use a secure key exchange scheme.

Remarks on mutual authentication: As may be seen from the definition, the guarantees provided to the instances in *open* and *connect* roles are not completely symmetric. Specifically, an instance in a *connect* role is guaranteed that a connection will be established (since its partner is ready to accept the connection). On the other hand, an instance (i, j) in an *open* role has no guarantee that anyone will ever connect to it. All that is required is that there must exist an instance (i', j') of the partner that has not yet been terminated, and the only connection that either (i, j) or (i', j') could ever establish would be to each other. It appears impossible to completely eliminate the asymmetry of the definition without unrealistic assumptions.

The unilateral-authentication model of Shoup [35] is different from the one presented above in the following way: The *open* connection assignment does not specify the instance from which the connection is expected, and any instance of the partner is allowed to connect. On the other hand, the *connect* connection assignment still specifies an instance. It is unclear to us as to why one would need a model that provides authentication to only one of the parties. In any case, as Shoup remarks, authentication is not an important issue in ordinary key exchange. We will see, however, that authentication is more of an issue with passwords. This will become apparent when we consider the password key exchange model with implicit authentication in Section 5.1.

Discussion (password authentication): The major difficulty in designing an ideal system for password-authenticated key exchange is that there may be a non-negligible probability of an adversary guessing a password and impersonating a user. Thus either the definition of security must allow for a non-negligible simulation error, or the ideal system is forced to have some notion of passwords built in.

Our ideal system for password-authenticated key exchange explicitly uses (ring master generated) passwords. This forces any proof of security to have a simulator that has to be not only aware of a password-guessing attempt, but of exactly which password is being guessed. (In the proof for the PAK protocol presented in this thesis, the simulator is able to learn this information.) If this is not possible, then constructing and using a different ideal system would be required.

We did not specify how the ring master chooses passwords for pairs of users. The simplest model would be to have a dictionary $\mathcal{D}$, which is a set of strings, and let all passwords be chosen uniformly and independently from that dictionary. To achieve the strongest notion of security, though, we can give the adversary all the power, and simply let her specify the distribution of the passwords as an argument to the *initialize user* operation (the specification of the distribution would be recorded in the transcript). The passwords of a user could even be dependent on the passwords of other users. We note that our proofs of security do not rely on any specific distribution of passwords, and would thus be correct even in the stronger model.

Why does our ideal system correctly describe the ideal world of password-authenticated key exchange? If two users successfully complete a key exchange, then the adversary cannot obtain the key or the password. This is modeled by the adversary not being allowed any *test instance password* queries for a successful key exchange. On the other hand, the adversary is allowed one test password query for any other key exchange, with successful impersonation only allowed if the adversary actually guesses the password correctly. This corresponds to an on-line impersonation/password-guessing attempt by the adversary. (One may think of this as modeling an adversary who attempts to log in to a server by sending a guessed password.)

We also model the ability for an adversary to set up passwords between any users and himself, using the *set password* query. This can be thought of as letting the adversary set up rogue accounts on any computer she wishes, as long as those rogue accounts have different user IDs from all the valid users.

3.3 Real System with Passwords

We now describe the real system in which we assume a password-authenticated key exchange protocol runs. Again, this is basically from [35], except that we do not concern ourselves with public keys and certification authorities, since all authentication is performed using shared passwords. Note that the same real system is used, no matter what authentication model we choose for our ideal 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:

initialize user [Transcript: (`"initialize user"`, i, ID_i)]
As in the ideal system (with passwords), the adversary assigns identity string ID_i to (new) user i (as before). In addition, a random password $\pi[ID_i, ID_{i'}]$ is chosen by the ring master for each existing user i'. The passwords are not placed in the transcript. This models the out-of-band communication required to set up passwords between users. As mentioned above with respect to the ideal system, the distribution of passwords may be fixed, or may be specified by the adversary as an argument to the operation (in the latter case, a description of the distribution would be recorded in the transcript).

initialize user instance [Transcript: (`"init. user inst."`, $i, j, role(i, j), PID_{ij}$)]
As in the ideal system, the adversary assigns a user instance (i, j) a role (one of {open, connect}) and a user PID_{ij} that is supposed to be the partner of (i, j).

deliver message [Transcript: ($\texttt{"impl"}, \texttt{"message"}, i, j, \mathit{InMsg}, \mathit{OutMsg}, \mathit{status}$)]

The adversary delivers *InMsg* to user instance (i, j). The user instance updates its state, and replies with *OutMsg* and reports *status*. If *status* is "accept", the record ($\texttt{"start session"}, i, j$) is added to the transcript, and if *status* is "reject", the record ($\texttt{"terminate instance"}, i, j$) is added to the transcript.

set password [Transcript: ($\texttt{"set password"}, i, ID', \pi$)]

As in the ideal system, this sets $\pi[ID_i, ID']$ to π and places a record in the transcript.

application [Transcript: ($\texttt{"application"}, f, f(R, \{K_{ij}\})$)]

As in the ideal system, the adversary is allowed to obtain any information she wishes about the environment and the session keys, except that the keys are now actual session keys generated by user instances.

random oracle [Transcript: ($\texttt{"impl"}, \texttt{"random oracle"}, i, x, H_i(x)$)]

The adversary queries random oracle i on a binary string x and receives the result of the random oracle query $H_i(x)$. Note that we do not allow *application* operations to query random oracles H_i. In other words, we do not give higher-level protocols access to the random oracles used by the key exchange scheme. (Although a higher-level protocol could have its own random oracle.) The adversary, however, does have access to all the random oracles.

For an adversary $\mathcal{A}$, *RealWorld*($\mathcal{A}$) denotes the transcript of the adversary's operations. In addition to records made by the operations, the transcript will include the random coins of the adversary in an *implementation* record ($\texttt{"impl"}, \texttt{"coins"}, \mathit{coins}$).

4 Explicit Authentication: The PAK Protocol

4.1 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 κ and p of size ℓ be primes such that $p = rq + 1$ for some value r co-prime to q. Let g be a generator of a 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 g^{xy} of $X = g^x$ and $Y = g^y$. We assume the hardness of the *Decision Diffie-Hellman problem* (DDH) in $G_{p,q}$. One formulation is that given g, X, Y, Z in $G_{p,q}$, where $X = g^x$ and $Y = g^y$ are chosen randomly, and Z is either $\mathrm{DH}(X, Y)$ or random, each with half probability, determine if $Z = \mathrm{DH}(X, Y)$. Breaking DDH implies a constructing a polynomial-time adversary that distinguishes $Z = \mathrm{DH}(X, Y)$ from a random Z with non-negligible advantage over a random guess.

4.2 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

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

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

is described as returning a bit string, we will operate on its output as a number modulo p.

The PAK protocol is given in Figure 1.

Theorem 1. *The PAK protocol is a secure password-authenticated key exchange protocol in the explicit-authentication model.*

The proof is given in Appendix A.

5 Implicit Authentication: The PPK Protocol

We first describe an Ideal System with Implicit Authentication, and then describe the PPK protocol. Note that we still use the Real System from Section 3.3.

5.1 Ideal System with Implicit Authentication

Here we consider protocols in which the parties are *implicitly* authenticated, meaning that if one of the communicating parties is not who it claims to be, it simply won't be able to obtain the session key of the honest party. However, the honest party (which could be playing the role of `"open"` or `"connect"`) would still open the session, but with no one able to actually communicate with on that session.[7]

Thus some of the connections may be "dangling." We will allow two new connection assignments:

dangling open. This requires *role*(i, j) to be "open."

dangling connect. This requires *role*(i, j) to be "connect."

In both cases, the ring master generates K_{ij} randomly.

To use implicit authentication with passwords, we will make the following rules:

[7] In a later version of [35], Shoup also deals with implicit authentication, but in a different way. We feel our solution is more straightforward and intuitive.

- We no longer require that no two instances are open for connection from the same instance (since for implicit authentication we don't care that every accepting instance has a unique partner instance).
- Dangling connection assignments are allowed even for instances on which the *test instance password* query has been performed.
- A *test instance password* query is allowed on an instance that has started a session with a dangling connection assignment. In that case, if the test is successful, then the ring master returns to the adversary the key of the instance (the adversary is "rewarded" with the key, since the usual "reward," i.e., the ability to make an *expose* connection assignment, is not usable once the dangling connection assignment has been made).

We still restrict the number of *test instance password* queries to at most one per instance. The rules relating to other connection assignments do not change.

The reason for the permissiveness in *test instance password* is that an instance with a dangling connection assignment can't be sure that it wasn't talking to the adversary. All that is guaranteed is that the adversary won't be able to get the key of that instance, unless she correctly guesses the password.

In practice, this means that we can't rule out an unsuccessful password guess attempt on an instance until we can confirm that some partner instance has obtained the same key. (If another instance has indeed obtained the same key, that guarantees that a non-dangling connection assignment was made, since otherwise the keys would be independent.) It follows that if we are trying to count the number of unsuccessful login attempts (e.g., so that we can lock the account when some threshold is reached), we can't consider an attempt successful until we get some kind of confirmation that the other side has obtained the same key. We thus see that key confirmation (which, in our model, is equivalent to explicit authentication) is indeed relevant when we use passwords.

5.2 PPK Protocol

If we don't require explicit authentication, we can make a much more efficient protocol. The PPK protocol requires only two rounds of communication. The protocol is given in Figure 2. Here $H_1' : \{0,1\}^* \to \{0,1\}^\eta$ is another random function. PPK has the same basic structure as PAK, except without the "authentication" values k and k'.

Note that in PPK, as opposed to PAK, both the m and μ values need to be "encrypted" with the password (by multiplication with a random oracle value). The reason is that in PPK these values are not in any way authenticated. If, as in PAK, μ wasn't "encrypted," then an adversary could carry out the following attack:

1. Perform a conversation with Alice, impersonating Bob: Alice sends $m = g^x \cdot (H_1(A, B, \pi^*))^r$. Adversary responds with $\mu = g^y$.
2. Adversary asks an *application* query to get Alice's key
$$K = H_3(A, B, m, \mu, (\frac{m}{(H_1(A, B, \pi^*))^r})^y, \pi^*).$$
3. Now the adversary can carry out an off-line dictionary attack, by checking K against
$$H_3(A, B, m, \mu, (\frac{m}{(H_1(A, B, \pi))^r})^y, \pi)$$
for all values of π.

$$\begin{array}{lcr}
A & & B \\
 & & \\
x \in_R Z_q & & \\
m = g^x \cdot (H_1(A,B,\pi))^r & \xrightarrow{\quad m \quad} & \text{Test } m \overset{?}{\not\equiv} 0 \bmod p \\
 & & y \in_R Z_q \\
 & & \mu = g^y \cdot (H_1'(A,B,\pi))^r \\
 & & \sigma = (\frac{m}{(H_1(A,B,\pi))^r})^y \\
\text{Test } \mu \overset{?}{\not\equiv} 0 \bmod p & \xleftarrow{\quad \mu \quad} & K = H_3(A,B,m,\mu,\sigma,\pi) \\
\sigma = (\frac{\mu}{(H_1'(A,B,\pi))^r})^x & & \\
K = H_3(A,B,m,\mu,\sigma,\pi) & &
\end{array}$$

Figure 2: The PPK protocol, with $\pi = \pi[A, B]$. The resulting session key is K.

This attack is not possible against PAK, since Alice would not accept (and would not compute K) without receiving a proper k value, and the adversary can't generate the proper k value without knowing the password.

Theorem 2. *The PPK protocol is a secure password-authenticated key exchange protocol in the implicit-authentication model.*

The completeness requirement follows directly by inspection. The proof of simulatability appears in Appendix B. The basic structure of the proof is the same as for the PAK protocol, except that we now have an analog of Claim 1 for both initiator and responder instances.

6 Resilience to Server Compromise: The PAK-X Protocol

6.1 Ideal System with Passwords: Resilience to Server Compromise

We now define a system in which one party is designated as a server, and which describes the ability of an adversary to obtain information about passwords stored on the server, along with the resultant security. The information stored by the server is called *the verifier* (since it allows for verification of the client's password). The goal in this model is that the adversary shouldn't be able to impersonate the client, even if she obtains the verifier stored by the server (clearly, the adversary will be able to impersonate the server). Note that the verifier can be used for an off-line dictionary attack, since it obviously contains enough information to check the password. We can thus only require that impersonating the client using the verifier should be impossible without a dictionary attack.

The modifications to the model of the ideal system are as follows: One role (open or connect) is designated as the *server* role, while the other is designated as the *client* role. We add the *test password* and *get verifier* operations, and change the *start session* operation. In our ideal system there is no tangible verifier stored by the server. Rather, possession of the verifier (which results from the *get verifier* operation) gives the adversary a "right" to perform an off-line dictionary attack.

The operations are handled as follows:

test password

This query takes two users, say i and i', as arguments, along with a password guess π. If a *get verifier* query has been made on $\{i, i'\}$, then this returns whether $\pi = \pi[ID_i, ID_{i'}]$. If the comparison returns true, this is called a successful guess on $\{ID_i, ID_{i'}\}$. If no *get verifier* has been made on $\{i, i'\}$, then no answer is returned (but see the description of *get verifier* below).

This query does not place a record in the transcript. It can be asked any number of times, as long as the next query after every *test password* is of type *implementation.* (The idea of the last requirement is that a *test password* query has to be caused by a "real-world" operation, which leaves an *implementation* record in the transcript.)

get verifier [Transcript: (`"get verifier"`, i, i')]

Arguments: users i and i'. For each *test password* query on $\{i, i'\}$ that has previously been asked (if any), returns whether or not it was successful. If any one of them actually was successful, then this *get verifier* query is called a successful guess on $\{ID_i, ID_{i'}\}$. Note that the information about the success of failure of *test password* queries is *not* placed in the transcript.

start session [Transcript: (`"start session"`, i, j)]

In addition to the rules specified previously, a connection assignment of *expose* for client instance (i, j) is allowed at any point after a *get verifier* query on users i or i' has been performed, where $ID_{i'} = PID_{ij}$.

The *test password* query does not affect the legality of *open* and *connect* connection assignments.

6.2 Real System: Resilience to Server Compromise

As in the ideal system, one role (open or connect) is designated as the *server* role, while the other is designated as the *client* role. In the real system the server stores an actual bit string, a verifier, to verify a client's password. Thus the protocol has to specify a polynomial-time verifier generation algorithm *VGen* that, given a set of user identities $\{A, B\}$, and a password π, produces a verifier V.

As above for $\pi[A, B]$, we will write $V[A, B]$ to mean $V[\{A, B\}]$, the verifier of user pair $\{A, B\}$. A user instance (i, j) in the server role is given access to $V[ID_i, PID_{ij}]$. A user instance (i, j) in the client role is given access to $\pi[ID_i, PID_{ij}]$.

The changes to the *initialize user* and *set password* operations are as follows:

initialize user [Transcript: (`"initialize user"`, i, ID_i)]

In addition to what is done in the basic real system, $V[ID_i, ID_{i'}] = VGen(\{ID_i, ID_{i'}\},$ $\pi[ID_i, ID_{i'}])$ is computed for each i'.

set password [Transcript: (`"set password"`, i, ID', π)]

In addition to what is done in basic real system, $V[ID_i, ID']$ is set to $VGen(\{ID_i, ID'\}, \pi)$.

We add the *get verifier* operation:

get verifier [Transcript:(`"get verifier"`, i, i'), followed by (`"impl"`, `"verifier"`, i, i', $V[ID_i, ID_{i'}]$)]

The adversary performs this query with i and i' as arguments, with $V[ID_i, ID_{i'}]$ being returned.

$$
\begin{array}{lcr}
A \text{ (client)} & & B \text{ (server)} \\
x \in_R Z_q & & \\
m = g^x \cdot (H_1(A,B,V))^r & \xrightarrow{\quad m \quad} & \text{Test } m \overset{?}{\not\equiv} 0 \bmod p \\
 & & y \in_R Z_q \\
 & & \mu = g^y \\
 & & \sigma = (\frac{m}{(H_1(A,B,V))^r})^y \\
 & & c \in_R \{0,1\}^\kappa,\ a = g^{H'_0(A,B,c)} \\
\sigma = \mu^x & \xleftarrow{\quad \mu, a, k \quad} & k = c \oplus H_{2a}(A,B,m,\mu,a,\sigma, \\
c = k \oplus H_{2a}(A,B,m,\mu,a,\sigma,a^v,V) & & V^{H'_0(A,B,c)}, V) \\
\text{Test } a \overset{?}{=} g^{H'_0(A,B,c)} & & \\
k' = H_{2b}(A,B,m,\mu,\sigma,a,k,c,V) & & \\
K = H_3(A,B,m,\mu,\sigma,c,V) & \xrightarrow{\quad k' \quad} & \text{Test } k' \overset{?}{=} H_{2b}(A,B,m,\mu,\sigma,a,k,c,V) \\
 & & K = H_3(A,B,m,\mu,\sigma,c,V)
\end{array}
$$

Figure 3: The PAK-X protocol, with $\pi = \pi[A,B]$, $v = v[A,B]$, and $V = V[A,B]$. The resulting session key is K.

6.3 PAK-X Protocol

In our protocol, we will designate the *open* role as the client role. We will use A and B to denote the identities of the client and the server, respectively. In addition to the random oracles we have used before, we will use additional functions $H_0 : \{0,1\}^* \to \{0,1\}^{|q|+\kappa}$ and $H'_0 : \{0,1\}^* \to \{0,1\}^{|q|+\kappa}$, which we will assume to be random functions. The verifier generation algorithm is

$$VGen(\{A,B\},\pi) = g^{v[A,B]},$$

where we define $v[A,B] = H_0(\min(A,B), \max(A,B), \pi)$ (we need to order user identities, just so that any pair of users has a unique verifier).

The PAK-X protocol is given in Figure 3.

Theorem 3. *The PAK-X protocol is a secure password-authenticated key exchange protocol in the explicit-authentication model, with resilience to server compromise.*

The completeness requirement follows directly by inspection. The proof of simulatability appears in Appendix C. The basic structure of the proof is the same as for the PAK protocol. A major technical difficulty in the simulation is the case when the adversary has obtained the verifier, and is now acting as the server (in fact, the security of the SNAPI-X protocol [30] has only been shown under the assumption that such a scenario does not occur). The difficulty is that the simulator needs to verify the value of $V^{H'_0(A,B,c)}$ without knowing v. This is achieved by checking all the H'_0 queries until we find a query c that gives $a = g^{H'_0(A,B,c)}$. We can then use this value of c to determine the correct value of $V^{H'_0(A,B,c)}$. (A similar technique was used independently in [20] to obtain an efficient encryption scheme secure against an adaptive chosen-ciphertext attack.)

7 Conclusions and Open Problems

We have presented new formal definitions of security for password-authenticated key exchange protocols, with several variations: explicit authentication, implicit authentication, and resilience to server compromise. For each variation of the model, we have presented a new, efficient, and provably secure construction: the PAK, PPK, and PAK-X protocols. The proofs of security are based on the DDH problem and the random oracle assumption.

The major open problem seems to be the construction of provably secure password-authenticated key exchange protocols without using random oracles. It seems unlikely that random oracles can be easily removed from our schemes, as the proofs rely on the random oracle model quite heavily. One of the major issues in proving the security of a protocol in our security model is the ability of the simulator to extract the adversary's attempted guess of the password (as passwords appear explicitly in the ideal world). In our proofs, this extraction is accomplished by examining the adversary's random oracle queries, which is clearly impossible in a standard security model. It may, however, be possible to construct protocols based on zero-knowledge techniques, where the simulator would extract attempted passwords through rewinding. The problem with this approach is that the best known zero-knowledge protocols in the general concurrent setting take a polynomial number of rounds [33], which is too inefficient for use in practice. In any case, it may be advantageous to consider variations of the model, where the passwords do not need to be extracted, but can be tested in some indirect manner (e.g., by asking the ring master whether the password is equal to the value of some hard-to-compute expression).

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 STOC'98 [38], pages 419–428.

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In EUROCRYPT2000 [19].

[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 [15], pages 62–73.

[6] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology—EUROCRYPT 94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer-Verlag, 1995, 9–12 May 1994.

[7] 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.

[8] 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.

[9] 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 [15], pages 244–250.

[10] 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.

[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 [16], pages 63–72.

[13] V. Boyko, P. MacKenzie, and S. Patel. Provably-secure password authentication and key exchange using Diffie-Hellman. In EUROCRYPT2000 [19].

[14] R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In STOC'98 [38], pages 209–218.

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

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

[17] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22(6):644–654, 1976.

[18] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Trans. Info. Theory*, 31:469–472, 1985.

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

[20] E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In M. Wiener, editor, *Advances in Cryptology—CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer-Verlag, 15–19 Aug. 1999.

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

[22] 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.

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

[24] N. Haller. The s/key one-time password system. *RFC 1760*, 1995.

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

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

[27] J. Kilian, E. Petrank, and C. Rackoff. Lower bounds for zero knowledge on the internet. In *39th Annual Symposium on Foundations of Computer Science*, pages 484–492. IEEE, Nov. 1998.

[28] 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, Dec. 1989. Proceedings of the 12th ACM Symposium on Operating System Principles.

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

[30] P. MacKenzie and R. Swaminathan. Secure network authentication with password information. manuscript.

[31] 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.

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

[33] R. Richardson and J. Kilian. On the concurrent composition of zero-knowledge proofs. In *Advances in Cryptology—EUROCRYPT 99*, volume 1592 of *Lecture Notes in Computer Science*, pages 415–431. Springer-Verlag, 1999.

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

[35] V. Shoup. On formal models for secure key exchange. In CCS'99 [16], page to appear.

[36] W. Simpson. PPP challenge handshake authentication protocol (CHAP). *RFC 1994*, 1996.

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

[38] *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, Dallas, Texas, 23–26 May 1998.

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

[40] T. Wu. A real world analysis of kerberos password security. In *Proceedings of the 1999 Internet Society Network and Distributed System Security Symposium*, 1999.

A Security of the PAK Protocol

The completeness requirement follows directly by inspection. Here we prove that the simulatability requirement holds. The basic technique is essentially that of Shoup [35]. 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.

Let DH(X, Y) denote the Diffie-Hellman value g^{xy} of $X = g^x$ and $Y = g^y$.

A.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.

Indeed, one of the main concerns in the proof is dealing with the possible shadowings: either detecting them when they occur, or showing that they are impossible. The possible shadowings in the PAK simulator are shown in Table 2.

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 Decision Diffie-Hellman 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 $\pi^* = \pi[A, B]$.

	Current					
Earlier	H_{2a}	H_{2b}	H_3	B1	B1$'$	A2
H_{2a}				ϵ	ϵ	
H_{2b}						Claim 3
H_3						Claim 3
B1	Claim 3			ϵ	ϵ	
B1$'$	Explicit			ϵ	ϵ	
A2		Claim 3	Claim 3			ϵ

The names of rows and columns refer to parts of the simulator: handling of the adversary's random oracle queries (H_i) and message responses (e.g., B1). A message-response action without a prime (e.g., B1) denotes that there is a matching conversation with an acceptable partner, while a prime (e.g., B1$'$) denotes the lack of one. Only actions that could possibly cause shadowing are shown in the table.

The columns of the table correspond to an action being currently handled by the simulator. The rows correspond to earlier actions. An entry in the table indicates whether or not while handling the current action we need to be concerned for shadowing caused by an earlier action.

The entries of the table have the following meaning:

blank shadowing is impossible by the structure of the simulator,

ϵ shadowing is possible, but has a negligible probability (statistically),

claim if the adversary can cause such shadowing with nonnegligible probability, then we can construct a DDH distinguisher (as shown in the referenced claim),

explicit shadowing can indeed occur, and is explicitly dealt with in the simulator.

Table 2: Possible shadowings in the PAK simulator.

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 $\alpha[A, B, \pi] \in_R Z_q$ and store it. Then generate $h \in_R Z_p^*$ and $\beta \in_R Z_{\lfloor 2^\eta / p \rfloor}$, and return $(h^q g^{\alpha[A,B,\pi]} \bmod p) + \beta p$. Note that this will be indistinguishable from a random bit string of length η, since $h^q g^{\alpha[A,B,\pi]} \bmod p$ is a random element from Z_p^{*}[8] and $\frac{2^\eta \mathrm{mod} p}{2^\eta}$ is negligible.

   Note that $(H_1(A, B, \pi))^r = (h^q g^{\alpha[A,B,\pi]})^r = h^{qr} g^{r \cdot \alpha[A,B,\pi]} = g^{r \cdot \alpha[A,B,\pi]}$.

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

   Case 1 Some prior B1 query has recorded a tuple of the form $(i', j', m, \mu, y, k_{i'j'})$, with $A = \mathit{PID}_{i'j'}$ and $B = \mathit{ID}_{i'}$, for some values of y 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 $(\frac{m}{(H_1(A,B,\pi))^r})^y = \sigma$, 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 a *test instance password* operation has not previously been performed on (i', j'), perform a *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 a *test instance password* operation has 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 $H_{2a}(A, B, m, \mu, \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,\pi^*))^r}), \pi^*)$. 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.)

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

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

   and

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

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

[8] To see this, note that h^q is a random element from the subgroup of order r in Z_p^* and $g^{\alpha[A,B,\pi]}$ is a random element of the subgroup of order q in Z_p^*.

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 A.2.

Note that an H_{2a} query can result in a *test instance password* query on (i', j') only if a tuple $(i', j', m, \mu, y, k_{i'j'})$ 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 $\sigma = \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,\pi))^r})$. 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 $(A, B, m, \mu, \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,\pi))^r}), \pi)$ to H_{2a}, H_{2b}, or H_3, for any π.*

The proof of the claim appears in Appendix A.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 $w \in_R Z_q$, and send $m = g^w$. Clearly, m is uniformly drawn from $G_{p,q}$, just as in the real system.

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

   Generate $y \in_R Z_q$ and $k_{i'j'} \in_R \{0, 1\}^{\kappa}$. Send $\mu = g^y$ 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

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

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 $(i', j', m, \mu, y, k_{i'j'})$. 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

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

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:

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

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 $H_{2a}(A, B, m, \mu, \mu^{w - r\alpha[A,B,\pi]}, \pi)$ 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 $k = H_{2a}(A, B, m, \mu, \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,\pi^*))^r}), \pi^*)$ 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 $k' = H_{2b}(A, B, m, \mu, \mu^{w-r\alpha[A,B,\pi]}, \pi)$.

(b) Send k'.

(c) Accept.

(d) Set $K = H_3(A, B, m, \mu, \mu^{w-r\alpha[A,B,\pi]}, \pi)$.

(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 $H_{2b}(A, B, m, \mu, (\frac{m}{(H_1(A,B,\pi))^r})^y, \pi)$ 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 $s = 2a$ and $t = 2b$), that, w.o.p., this procedure will not make a *test instance password* query on (i', j') if one has already been made before.

If we haven't rejected yet, then

(a) set status to Accept,

(b) set $K = H_3(A, B, m, \mu, (\frac{m}{(H_1(A,B,\pi))^r})^y, \pi)$, 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.

A.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 $\sigma = \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,\pi))^r})$.

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 two "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 DDH problem.

The idea of the distinguisher D is as follows: We start with a triple (X, Y, Z) as input, for which we need to determine whether or not $Z = \mathrm{DH}(X, Y)$. We will run the adversary against a *simulation* of the original PAK simulator. In the simulation, we will use X and Y 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 $Z = \mathrm{DH}(X, Y)$.

More specifically, we will "incorporate" X into half of the H_1 responses (randomly), and Y into B1 responses (i.e., the μ values). A bad query will then have to contain

$$\sigma = \mathrm{DH}(\mu, \frac{m}{(H_1(A, B, \pi))^r}) = \mathrm{DH}(Yg^z, \frac{m}{X^{b[\pi]r}g^\alpha}),$$

where $b[\pi] \in \{0, 1\}$, z and α are random values known to us, and m comes from the adversary. One bad query does not give us information about $\mathrm{DH}(X, Y)$, since we don't know m. However, two bad queries with different values of $b[\pi]$ will give us enough information. If $b[\pi]$ is randomly chosen for each π, and the adversary asks two bad queries for different choices of the password, then we will get information about $\mathrm{DH}(X, Y)$. The main difficulty in the construction is how to perform the simulation without knowing the discrete logarithms of H_1 and μ values. For that reason, we want to minimize the number of places where X and Y get used, and so we need to make a guess as to where the "bad event" will occur.

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

1. Generate random d between 1 and T.

2. Initialize two lists BAD_0 and BAD_1 (initially empty).

3. 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 simulation until the dth pair of users (A, B) is mentioned. (This may be from an oracle query $H_1(A, B, \cdot)$, or from an *initialize user instance* query with user ID A and partner ID B, or vice-versa.) If we guessed d correctly, this pair will be the identities of the users in the "bad event."

4. Once A and B are set, continue as in the original simulator, except:

   (a) $\mathrm{B1}(m)$ query to instance (i', j') with $ID_{i'} = B$ and $PID_{i'j'} = A$: generate $z_{i'j'} \in_R Z_q$ and $k_{i'j'} \in_R \{0,1\}^\kappa$, set $\mu = Yg^{z_{i'j'}}$, and respond with $(\mu, k_{i'j'})$.

   (b) $H_1(A, B, \pi)$: If $\pi = \pi^*$, then set $b[\pi] = b[\pi^*] = 0$. Otherwise, let $b[\pi] \in_R \{0,1\}$. Respond with $(X^{b[\pi]} \cdot h^q g^{\alpha[A,B,\pi]} \bmod p) + \beta p$, for $\alpha[A, B, \pi] \in_R Z_q$, $h \in_R Z_p^*$, and $\beta \in_R Z_{\lfloor 2^\eta/p \rfloor}$.

   (c) $\mathrm{A2}(\mu, k)$ 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

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

   i.e., ignore any $\pi \neq \pi^*$.

   Note that we know π^*, and that $(H_1(A, B, \pi^*))^r = g^{r \cdot \alpha[A,B,\pi^*]}$, since $b[\pi^*] = 0$. 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).

   (d) $\mathrm{B3}(k')$ 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 has 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 second "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).

   (e) $(A, B, m, \mu, \sigma, \pi)$ query to H_{2a} or H_{2b}:

   If $H_1(A, B, \pi)$ was queried and there is an instance (i', j') with $B = ID_{i'}$ that was queried with $\mathrm{B1}(m)$ and returned $\mu = Yg^{z_{i'j'}}$, then do the following (and otherwise, answer the query as in the original simulator):

   First note that if this is a bad query, then

   $$\sigma = DH(\mu, \frac{m}{(H_1(A, B, \pi))^r}) = DH(Yg^{z_{i'j'}}, \frac{m}{X^{b[\pi]r} g^{r \cdot \alpha[A,B,\pi]}}).$$

Now compute

$$\gamma = \sigma m^{-z_{i'j'}} X^{b[\pi] r z_{i'j'}} Z^{b[\pi] r} Y^{r \alpha[A,B,\pi]} g^{r \cdot \alpha[A,B,\pi] z_{i'j'}}.$$

Put γ on the list $\mathrm{BAD}_{b[\pi]}$. Then 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 whether the lists BAD_0 and BAD_1 intersect (note that this check can be done in time $O(T \log T)$, by sorting the lists together). If yes, then output "True DH," otherwise output "Random."

Note that the values stored in BAD_0 and BAD_1 are simply guesses of $DH(m, Y)$, assuming $Z = DH(X, Y)$. If Z is random, then the probability of an intersection between the lists would be at most T^2/q by the union bound, since Z^r would be a random element of $G_{p,q}$ (note that q and r are relatively prime, and $m \neq 0 \bmod p$ because of the test by B).

On the other hand, suppose $Z = \mathrm{DH}(X, Y)$. If the adversary makes two "bad" queries for the pair of users (A, B), for passwords π_1, π_2 with $b[\pi_1] \neq b[\pi_2]$, then the distinguisher will correctly answer "True DH" (since each "bad" query will result in $\gamma = \mathrm{DH}(m, Y)$). The probability of the "bad event" is ϵ. The probability of guessing d correctly is $\frac{1}{T}$. The probability of $b[\pi_1] \neq b[\pi_2]$ for $\pi_1 \neq \pi_2$ is $\frac{1}{2}$. All of these events are independent.

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

$$\begin{aligned} \Pr(D \text{ is correct}) &= \Pr(D \text{ guesses "DH True"} | \text{DH instance}) \Pr(\text{DH instance}) \\ &\quad + \Pr(D \text{ guesses "Random"} | \text{Random instance}) \Pr(\text{Random instance}) \\ &\geq \left(\frac{\epsilon}{2T}\right)\left(\frac{1}{2}\right) + \left(1 - \frac{T^2}{q}\right)\left(\frac{1}{2}\right). \end{aligned}$$

Thus the probability that D is correct is at least $\frac{1}{2} + \frac{\epsilon}{4T} - \frac{T^2}{2q}$, 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 a distinguisher D for DDH. Let (X, Y, Z) be the challenge DDH 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 distinguisher runs as follows:

1. Generate random d between 1 and T, and random $b \in \{0, 1\}$.

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) $\mathrm{B1}(m)$ query to instance (i', j') where $m = X$, $ID_{i'} = B$, and $PID_{i'j'} = A$:
   Generate a random $z_{i'j'} \in Z_q$, and set $\mu = Y g^{z_{i'j'}}$. Compute $k = k_{i'j'}$ as in the original simulator and return (μ, k).

(b) Any H_{2a}, H_{2b} or H_3 query $(A, B, X, \mu, \sigma, \pi)$ where $\mu = Yg^{z_{i'j'}}$ for some (i', j') and $\sigma = ZX^{z_{i'j'}}/\mu^{r\alpha[A,B,\pi]}$: Stop the distinguisher and guess "True DH."

(c) A2(μ, k) query to (i, j) not part of a matching conversation with an acceptable partner: Check that an $H_1(A, B, \pi^*)$ query has been asked and a query $H_{2a}(A, B, m, \mu, \sigma, \pi^*)$ has been asked and returned k (w.o.p., there will be at most one such H_{2a} query, by the randomness of H_{2a} responses). If not, then reject.

Otherwise, if $b = 0$, reject, and if $b = 1$,

i. Set $k'_{ij} = H_{2b}(A, B, m, \mu, \sigma, \pi^*)$.
ii. Send $k' = k'_{ij}$.
iii. Accept.
iv. Set $K_{ij} = H_3(A, B, m, \mu, \sigma, \pi^*)$.
v. Expose using $K = K_{ij}$.

(d) B3(k') query to instance (i', j') where $m = X$, 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 "True DH," then output "Random."

Note that if the adversary does make a bad query, we have probability at least $1/T$ of guessing the correct initiator user instance, and probability at least $1/2$ of answering the A2 query to that user instance correctly (thus allowing the DDH distinguisher to continue in a way that is indistinguishable from the regular simulator).

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

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

where the term T/q comes from the probability of the adversary "guessing" a random Z correctly on a random oracle query. Thus, the probability that D guesses correctly is at least $\frac{1}{2} + \frac{\epsilon}{4T} - \frac{T}{2q}$, which is non-negligibly more than $\frac{1}{2}$. □

Remark on the use of Computational Diffie-Hellman: It is easy to see that the proofs could be modified to base security on the Computational Diffie-Hellman problem (i.e., the hardness of computing $Z = \text{DH}(X, Y)$ for random X and Y). Basically, instead of checking whether the random oracle queries match a given form, we would just guess which of the oracle queries are "right," and thereby extract Z. However, this approach would only give $O(\frac{\epsilon}{T^3})$ success probability in Claim 1, since we would need to guess the location of both of the bad oracle queries.

B Security of the PPK Protocol

B.1 The Simulator

The proof of simulatability of PPK is similar in structure to that of PAK. We name the actions of user instances on *deliver message* operations as follows:

	Current			
Earlier	H_3	B1	B1′	A2′
H_3		ϵ	ϵ	
B1	Claim 8	ϵ	ϵ	
B1′	Explicit (case 1 of H_3)	ϵ	ϵ	
A2′	Explicit (case 2 of H_3)	ϵ	ϵ	ϵ

Table 3: Possible shadowings in the PPK simulator. This table is to be interpreted in the same way as Table 2 (see page 20), except for one technicality: For clarity of presentation, the "B1" entry in the table covers cases 1 and 2 of the B1 action, while the "B1′" entry covers case 3, even though in case 2 there is no matching conversation (and thus it should, strictly speaking, be included in "B1′").

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).

The possible shadowings in the PPK simulator are shown in Table 3.

We will let (i, j), (i', j'), A, B, and π^* have the same meaning as in the proof of PAK (see the beginning of Appendix A.1). The notions of *acceptable partner* and *matching conversation* will also retain their meaning.

The simulator will follow the same general rules as described in Appendix A.1. Responses to *deliver message* and *random oracle* operations that do not fall under those rules are done as follows:

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

   Same as for PAK: Generate $\alpha[A, B, \pi] \in_R Z_q$ and store it. Then generate $h \in_R Z_p^*$ and $\beta \in_R Z_{\lfloor 2^\eta/p \rfloor}$, and return $(h^q g^{\alpha[A,B,\pi]} \bmod p) + \beta p$. Note that this will be indistinguishable from a random bit string of length η, since $h^q g^{\alpha[A,B,\pi]} \bmod p$ is a random element from Z_p^*[9] and $\frac{2^\eta \bmod p}{2^\eta}$ is negligible.

2. $H_1'(A, B, \pi)$

   Same as for PAK: Generate $\alpha'[A, B, \pi] \in_R Z_q$ and store it. Then generate $h \in_R Z_p^*$ and $\beta' \in_R Z_{\lfloor 2^\eta/p \rfloor}$, and return $(h^q g^{\alpha'[A,B,\pi]} \bmod p) + \beta' p$. Note that this will be indistinguishable from a random bit string of length η, since $h^q g^{\alpha'[A,B,\pi]} \bmod p$ is a random element from Z_p^*[10] and $\frac{2^\eta \bmod p}{2^\eta}$ is negligible.

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

[9] To see this, note that h^q is a random element from the subgroup of order r in Z_p^* and $g^{\alpha[A,B,\pi]}$ is a random element of the subgroup of order q in Z_p^*.

[10] To see this, note that h^q is a random element from the subgroup of order r in Z_p^* and $g^{\alpha'[A,B,\pi]}$ is a random element of the subgroup of order q in Z_p^*.

Case 1 Some prior B1 query has recorded a tuple of the form (i', j', m, μ, z), with $A = PID_{i'j'}$ and $B = ID_{i'}$, for some value of z. This implies that a B1(m) query was made to (i', j') and returned μ, but no A0 query to an acceptable partner returned m—see the B1 query description below. (In other words, this H_3 query might be shadowed by the implicit H_3 query from case 3 of the B1 action.) Instance (i', j') must have already opened a session with the *dangling open* connection assignment (see case 3 of B1).

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

(a) If a *test instance password* operation has not previously been performed on (i', j'), perform a *test instance password* operation with arguments (i', j') and π. If the test is successful, the ring master will give us the key $K_{i'j'}$ of instance (i', j') (since (i', j') has a dangling connection assignment). Return $K_{i'j'}$.

(b) If a *test instance password* operation with π has already been performed on (i', j') and was successful, then let $K_{i'j'}$ be the key obtained from the ring master in that operation. Return $K_{i'j'}$.

(c) If an unsuccessful *test instance password* operation has already been performed on (i', j') with a password $\pi' \neq \pi$, then abort. We will call this event a *responder failure* (note that (i', j') is a responder instance).

If we have neither failed nor returned anything yet, then return $k \in_R \{0, 1\}^\kappa$.

If no responder failure occurs as a result of this H_3 query, then the simulation will be indistinguishable from the real world, as follows: In the real world the key generated by (i', j') is

$$H_3\left(A, B, m, \mu, \mathrm{DH}\left(\frac{\mu}{(H_1'(A, B, \pi^*))^r}, \frac{m}{(H_1(A, B, \pi^*))^r}\right), \pi^*\right).$$

Therefore, if this H_3 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 cause a responder failure.

Claim 4. *W.o.p., no responder failure will occur.*

The proof appears in Appendix B.2. Note that case 1 of H_3 is the only place in the simulation where a responder failure can occur.

As can be seen, an H_3 query is the only place in the simulation that a *test instance password* query can be made on a responder instance. We can therefore state the following claim (similarly to PAK):

Claim 5. *If a* test instance password *query is made on a responder instance (i', j'), then the m value received by (i', j') was not sent by an acceptable partner.*

The reason is that an H_3 query can result in a *test instance password* query on (i', j') only if a tuple (i', j', m, μ, z) has previously been recorded. It will become apparent below (see the B1 query description) that, w.o.p., this could only happen if the m value received by (i', j') was not sent by an acceptable partner.

Case 2 Some prior A2 query has recorded a tuple of the form (i, j, m, μ, w), with $A = ID_i$ and $B = PID_{ij}$, for some value of w. This implies that (i, j) sent m and received μ back, but did not have a matching conversation with an acceptable partner—see the A2 query description below. (In other words, this H_3 query might be shadowed by the implicit H_3 query from case 2a or 2b of the A2 action.) Instance (i, j) must have already opened a session with the *dangling connect* connection assignment (see cases 2a and 2b of A2).

(Note that this case cannot overlap with the previous one, i.e., w.o.p., we won't have both a tuple (i', j', m, μ, z) and tuple (i, j, m, μ, w) recorded, with matching values of A and B. That reason is that if the values of m and μ in the tuples match up, then (i, j) and (i', j') will have had a matching conversation. Also, the two instances are acceptable partners. However, an initiator instance that has had a matching conversation with an acceptable partner will not record a tuple—it will be in case 1 of $A2$, while an initiator tuple can only be recorded in case 2 of $A2$.)

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

(a) If a *test instance password* operation has not previously been performed on (i, j), perform a *test instance password* operation with arguments (i, j) and π. If the test is successful, the ring master will give us the key K_{ij} of instance (i, j) (since (i, j) has a dangling connection assignment). Return K_{ij}.

(b) If a *test instance password* operation with π has already been performed on (i', j') and was successful, then let $K_{i'j'}$ be the key obtained from the ring master in that operation. Return $K_{i'j'}$.

(c) If an unsuccessful *test instance password* operation has already been performed on (i, j), then abort. We will call this event an *initiator failure* (note that (i, j) is an initiator instance; in addition, see case 2c of A2 for another place where an initiator failure can occur).

If we have neither failed nor returned anything yet, then return $k \in_R \{0, 1\}^\kappa$.

If no initiator failure occurs as a result of this H_3 query, then the simulation will be indistinguishable from the real world, similarly to case 1.

The following claim shows that w.o.p. $\mathcal{A}$ will not cause an initiator failure.

Claim 6. *W.o.p., no initiator failure will occur (whether in case 2 of H_3 or in case 2c of A2).*

The proof appears in Appendix B.2.

Note that the *test instance password* query on (i, j) will only be made here if a tuple (i, j, m, μ, w) has previously been recorded. That, in turn, implies w.o.p. that (i, j) did not have a matching conversation with an acceptable partner (see the note at the end of case 2b of A2). We have just proven one part of the following claim:

Claim 7. *If a* test instance password *query is made on an originator instance (i, j), then (i, j) has not had a matching conversation with an acceptable partner.*

The other part is shown in case 2a of A2 (which is the only other place where a *test instance password* could be made on an originator instance).

Case 3 Otherwise.

Return $k \in_R \{0, 1\}^\kappa$.

Let's show that this response is valid, i.e., that this H_3 query can't be shadowed by any prior implicit queries. Implicit H_3 queries are made in cases 1, 2, and 3 of B1, and cases 2a and 2b of A2. It is easy to see that any H_3 call that might shadow an implicit query from case 3 of B1 would not be dealt with here (rather, it would be dealt with in case 1 of H_3). Similarly, we don't need to be concerned for implicit queries from cases 2a and 2b of A2, since they are only relevant to case 2 of H_3. Thus, we only need to show that there is no shadowing from cases 1 and 2 of B1. These are dealt with by the following:

Claim 8. *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 a query $H_3(A, B, m, \mu, \mathrm{DH}(\frac{\mu}{(H_1'(A,B,\pi))^r}, \frac{m}{(H_1(A,B,\pi))^r}), \pi)$, for any π.*

The proof of the claim appears in Appendix B.2.

4. A0 query to (i, j)

   Same as for PAK: Generate and store $w \in_R Z_q$, and send $m = g^w$. Clearly, m is uniformly drawn from $G_{p,q}$, just as in the real system.

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

   Generate and store $z \in_R Z_q$, and send $\mu = g^z$ (which clearly has the correct distribution). Now consider three cases:

   Case 1 The value of m has been sent by an acceptable partner (i, j), and no A2 action has yet been performed for (i, j). (Note that by the randomness of m values, there will, w.o.p., be at most one such (i, j).)

   Set status to Accept, and perform a *start session* operation with the connection assignment *open for connection from* (i, j). This corresponds to the implicit oracle query

$$K_{i'j'} = H_3\left(A, B, m, \mu, \mathrm{DH}\left(\frac{m}{(H_1(A,B,\pi^*))^r}, \frac{\mu}{(H_1'(A,B,\pi^*))^r}\right), \pi^*\right),$$

   where $K_{i'j'}$ is the random session key assigned by the ring master at the time of the *start session* operation. Since μ is a freshly chosen random number, this implicit query, w.o.p., will not shadow any prior H_3 queries (whether explicit or implicit). Consequently, $K_{i'j'}$ will be indistinguishable from the value used in the real system.

   The connection assignment is legal because, by Claim 5, there could not have been a *test instance password* query on (i', j'), since (i', j') has received m from an acceptable partner.

   Case 2 The value of m has been sent by an acceptable partner (i, j), and an A2(μ') action has been performed for (i, j), for some μ'.

   Set status to Accept, and perform a *start session* operation with the connection assignment *dangling open*. This corresponds to the implicit oracle query

$$K_{i'j'} = H_3\left(A, B, m, \mu, \mathrm{DH}\left(\frac{m}{(H_1(A,B,\pi^*))^r}, \frac{\mu}{(H_1'(A,B,\pi^*))^r}\right), \pi^*\right),$$

   where $K_{i'j'}$ is the random session key assigned by the ring master at the time of the *start session* operation. As in case 1, it is easy to see that the values of μ and $K_{i'j'}$ will be indistinguishable from those used in the real system.

Case 3 The value of m has not been sent by an acceptable partner.

Record the tuple (i', j', m, μ, z). Then set status to Accept, and perform a *start session* operation with the connection assignment *dangling open*. This corresponds to the implicit oracle query

$$K_{i'j'} = H_3\left(A, B, m, \mu, \mathrm{DH}\left(\frac{m}{(H_1(A, B, \pi^*))^r}, \frac{\mu}{(H_1'(A, B, \pi^*))^r}\right), \pi^*\right),$$

and is valid for the same reason as in case 2.

It is important to note (for Claim 5), that if a tuple is recorded, then, w.o.p., the m value received by (i', j') neither was nor every will be sent by an acceptable partner, due to the fact that values of m are generated randomly by initiator instances.

6. A2(μ) query to (i, j)

Consider two cases:

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.

Let's show that this connection assignment is legal. Since (i, j) had a matching conversation with (i', j'), it follows that (i', j') has received the m sent by (i, j), and (i, j) received the μ sent by (i', j'). Since μ was a fresh random number generated by (i', j'), it follows that w.o.p. (i, j) received μ after (i', j') sent it. It is now easy to see that the B2(m) action for (i', j') was, w.o.p., in case 1. Thus, (i', j') was open for connection from (i, j). W.o.p., (i', j') became open for connection from (i, j) after (i, j) sent m, since m was a fresh random number.

In addition, by Claim 7, no *test instance password* could have been performed on (i, j), since (i, j) has had a matching conversation with an acceptable partner.

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

Now consider two cases:

Case 2a There exists exactly one π such that queries $H_1(A, B, \pi)$ and $H_1'(A, B, \pi)$ have been asked, and query $H_3(A, B, m, \mu, (\frac{\mu}{(H_1'(A,B,\pi))^r})^{w - r\alpha[A,B,\pi]}, \pi)$ has been asked and returned K.

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 7.) This is legal for the following reason: The only other place where a *test instance password* could be asked on an originator instance is in case 2 of H_3. However, that case can only lead to a test if a tuple $(i, j, \dots)$ has been recorded. Such a tuple can only be recorded by an A2 action. Now, clearly no tuple has been recorded thus far by this A2 action, and also no A2 action could have occurred on (i, j) before (we don't allow more than one A2 action to be performed on any particular instance).

(a) If the test is successful, then set status to Accept and expose using key K. This is acceptable for the same reasons as above.

(b) Otherwise, record the tuple (i, j, m, μ, w), set status to Accept, and perform a *start session* operation with the connection assignment *dangling connect*.

This corresponds to the implicit oracle query

$$K_{ij} = H_3\left(A, B, m, \mu, \mathrm{DH}\left(\frac{m}{(H_1(A,B,\pi^*))^r}, \frac{\mu}{(H_1'(A,B,\pi^*))^r}\right), \pi^*\right),$$

where K_{ij} is the random session key assigned by the ring master at the time of the *start session* operation. Clearly, this implicit query will not, w.o.p., shadow any prior explicit queries, or else we would have exposed. W.o.p., it won't shadow another implicit query from an A2 action, by the randomness of m values. Also, it won't shadow an implicit query from a B1 action of any instance (i', j'), since that would imply that (i, j) had a matching conversation with an acceptable partner (as A, B, m, and μ would have to match), and so we would not be in case 2 of A2.

Case 2b No such π exists.

Record the tuple (i, j, m, μ, w). Then set status to Accept, and perform a *start session* operation with the connection assignment *dangling connect.* This corresponds to the implicit oracle query

$$K_{ij} = H_3\left(A, B, m, \mu, \mathrm{DH}\left(\frac{m}{(H_1(A,B,\pi^*))^r}, \frac{\mu}{(H_1'(A,B,\pi^*))^r}\right), \pi^*\right),$$

and is valid for the same reason as above at the end of case 2a.

It is important to note (for Claim 7), that if a tuple is recorded (either in case 2a or in case 2b), 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 μ are generated randomly by responder instances. (Recall that in our definition of "matching conversation" we are not concerned with the timing of sends and receives.)

Case 2c There is more than one such π.

Abort with an *initiator failure.* W.o.p., this case will not occur, by Claim 6.

B.2 Proofs of Claims

Proof of Claim 4. We will call an oracle query of form $H_3(A, B, m, \mu, \sigma, \pi)$ "bad" if

$$\sigma = \mathrm{DH}(\frac{\mu}{(H_1'(A,B,\pi))^r}, \frac{m}{(H_1(A,B,\pi))^r}).$$

Suppose that with some nonnegligible probability ϵ there will be a responder failure. This implies that 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 two "bad" queries are made with $(A, B, \hat{m}, \hat{\mu})$ and distinct values of π, before there is either a successful *test instance password* on $(\hat{i}', \hat{j}')$ or an initiator failure. (Note that the simulator aborts when it encounters an initiator failure. Thus, if a responder failure occurs, that implies that no initiator failure has occurred so far.)

We will then show how to construct a distinguisher D for the DDH problem, with input (X, Y, Z).

The idea of the construction is similar to the one in the proof of Claim 1. Again, we will "incorporate" X into half of the H_1 responses (randomly), and Y into B1 responses (i.e., the μ

values). The main difference is in the definition of the bad event: In Claim 1, the bad event could not occur after a successful *test instance password* on $\{A, B\}$. In this claim, the bad event is only guaranteed not to occur after a successful *test instance password* on the *specific instance* involved. (The reason for the difference has to do with the special handling of *test instance password* for dangling connection assignments.) Since the bad event is thus less restricted, D needs to guess the actual instance where the bad event will occur, as opposed to just guessing the pair of users. This reduces the success probability of D (as compared to Claim 1) by a factor of $\frac{1}{T}$.

Our distinguisher D for input (X, Y, Z) runs as follows:

1. Generate random d_1 and d_2 between 1 and T.

2. Initialize two lists BAD_0 and BAD_1 (initially empty).

3. 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 simulation until the d_1th pair of users (A, B) is mentioned. (This may be from an oracle query $H_1(A, B, \cdot)$, or from an *initialize user instance* query with user ID A and partner ID B, or vice-versa.) If we guessed d_1 correctly, this pair will be the identities of the users in the "bad event."

4. Once A and B are set, continue as in the original simulator, except:

   (a) $\mathrm{B1}(m)$ query to the d_2th responder instance $(\hat{i}', \hat{j}')$ with $ID_{\hat{i}'} = B$ and $PID_{\hat{i}'\hat{j}'} = A$: respond with $\hat{\mu} = Y$. Let $\hat{m} = m$.

   If we guessed d_2 correctly, this will be the responder instance in the "bad event." Note that it is OK to set $\hat{\mu} = Y$, since Y is assumed to be uniformly distributed.

   (b) $H_1(A, B, \pi)$: If $\pi = \pi^*$, then set $b[\pi] = b[\pi^*] = 0$. Otherwise, let $b[\pi] \in_R \{0, 1\}$. Respond with $(X^{b[\pi]} \cdot h^q g^{\alpha[A,B,\pi]} \bmod p) + \beta p$, for $h \in_R Z_p^*$, $\alpha[A, B, \pi] \in_R Z_{p-1}$ and $\beta \in_R Z_{\lfloor 2^\eta / p \rfloor}$.

   (c) $\mathrm{A2}(\mu)$ 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

   $$H_3(A, B, m, \mu, (\frac{\mu}{(H_1'(A, B, \pi^*))^r})^{w - r\alpha[A,B,\pi^*]}, \pi^*),$$

   i.e., ignore any $\pi \neq \pi^*$.

   Note that we know π^*, and that $(H_1(A, B, \pi^*))^r = g^{r \cdot \alpha[A,B,\pi^*]}$, since $b[\pi^*] = 0$. 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).

   It is easy to see that the only way this behavior might be distinguishable from the original simulator is if this query was supposed to cause an abort (i.e., an initiator failure). However, this is not a problem: If the bad event has already occurred, then we are no longer interested in preserving indistinguishability. On the other hand, if the bad event is still about to occur, then this query couldn't cause an initiator failure (since the bad event, by definition, can't occur after an initiator failure).

   (d) $H_3(A, B, m, \mu, \sigma, \pi)$ query, with a tuple (i, j, m, μ, w) recorded, $A = ID_i$, $B = PID_{ij}$: If $\pi = \pi^*$, then behave as in the original simulator (this will be correct, since $b[\pi^*] = 0$). Otherwise, return $k \in_R \{0, 1\}^\kappa$.

   This behavior is indistinguishable from the original simulator for the same reasons as mentioned in the A2 query.

(e) $H_3(A, B, \hat{m}, \hat{\mu}, \sigma, \pi)$ query (note that this case will not, w.o.p., overlap with the preceding one for reasons similar to those mentioned in case 2 of H_3 in the description of the original simulator, i.e., that it would imply a matching conversation with an acceptable partner, and thus the tuple $(i, j, \dots)$ wouldn't have been recorded):
If $H_1(A, B, \pi)$ and $H_1'(A, B, \pi)$ were queried, then do the following (and otherwise, answer the query as in the original simulator):
First note that if this is a bad query, then

$$\begin{aligned}\sigma &= DH\left(\frac{\hat{m}}{(H_1(A,B,\pi))^r}, \frac{\hat{\mu}}{(H_1'(A,B,\pi))^r}\right)\\ &= DH\left(\frac{\hat{m}}{X^{b[\pi]r}g^{r\cdot\alpha[A,B,\pi]}}, \frac{Y}{g^{r\cdot\alpha'[A,B,\pi]}}\right).\end{aligned}$$

Now compute

$$\gamma = \sigma \cdot Z^{b[\pi]r} \cdot Y^{r\alpha[A,B,\pi]} \cdot (\hat{m} \cdot X^{-b[\pi]r} \cdot g^{-r\alpha[A,B,\pi]})^{r\alpha'[A,B,\pi]}.$$

Put γ on the list $\mathrm{BAD}_{b[\pi]}$. Then 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 *test instance password* on $(\hat{i}', \hat{j}')$ can occur before the bad event.

At the end of the simulation, check whether the lists BAD_0 and BAD_1 intersect (note that this check can be done in time $O(T \log T)$, by sorting the lists together). If yes, then output "True DH," otherwise output "Random."

Note that the values stored in BAD_0 and BAD_1 are simply guesses of $DH(\hat{m}, Y)$, assuming $Z = DH(X, Y)$. If Z is random, then the probability of an intersection between the lists would be at most T^2/q by the union bound, since Z^r would be a random element of $G_{p,q}$ (note that q and r are relatively prime, and $m \neq 0 \bmod p$ because of the test by B).

On the other hand, suppose $Z = \mathrm{DH}(X, Y)$. If the adversary makes two "bad" queries for the pair of users (A, B), for passwords π_1, π_2 with $b[\pi_1] \neq b[\pi_2]$, then the distinguisher will correctly answer "True DH" (since each "bad" query will result in $\gamma = \mathrm{DH}(\hat{m}, Y)$). The probability of the "bad event" is ϵ. The probability of guessing d_1 and d_2 correctly is $\frac{1}{T^2}$. The probability of $b[\pi_1] \neq b[\pi_2]$ for $\pi_1 \neq \pi_2$ is $\frac{1}{2}$. All of these events are independent.

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

$$\begin{aligned}\Pr(D \text{ is correct}) &= \Pr(D \text{ guesses "DH True"} | \text{DH instance}) \Pr(\text{DH instance})\\ &\quad + \Pr(D \text{ guesses "Random"} | \text{Random instance}) \Pr(\text{Random instance})\\ &\geq \left(\frac{\epsilon}{2T^2}\right)\left(\frac{1}{2}\right) + \left(1 - \frac{T^2}{q}\right)\left(\frac{1}{2}\right).\end{aligned}$$

Thus the probability that D is correct is at least $\frac{1}{2} + \frac{\epsilon}{4T^2} - \frac{T^2}{2q}$, which is non-negligibly more than $\frac{1}{2}$. □

Proof of Claim 6. We will call an oracle query of form $H_3(A, B, m, \mu, \sigma, \pi)$ "bad" if

$$\sigma = \mathrm{DH}(\frac{\mu}{(H_1'(A,B,\pi))^r}, \frac{m}{(H_1(A,B,\pi))^r})$$

.

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

1. query A0 is made to $(\hat{i},\hat{j})$ and returns $\hat{m}$,
2. at least two "bad" queries are made with $(A, B, \hat{m}, \hat{\mu})$ and distinct values of π, for some $\hat{\mu}$, before there is either a successful *test instance password* on $(\hat{i},\hat{j})$ or a responder failure, and
3. at some point in the execution, the adversary asks the query A2$(\hat{\mu})$ to $(\hat{i},\hat{j})$, and it is processed in case 2 of A2 (this could be either before, in between, or after the bad queries).

We will then show how to construct a distinguisher D for the DDH problem, with input (X, Y, Z).

The construction of D is similar to that of Claim 4, except that now the roles of A and B are switched around. Most notably, H_1 and H'_1 are switched around. Thus, we will "incorporate" X into half of the H'_1 responses (randomly), and Y into an A0 response (i.e., m value). One peculiarity of the construction for this claim is that the value of $\hat{\mu}$ might not become known until after the "bad" queries are made to H_3 (see the definition of the bad event above). For that reason the lists BAD_0 and BAD_1 are now indexed with values of μ. At the end, when $\hat{\mu}$ is already set, the lists $\mathrm{BAD}_0[\hat{\mu}]$ and $\mathrm{BAD}_1[\hat{\mu}]$ are compared.

Our distinguisher D for input (X, Y, Z) runs as follows:

1. Generate random d_1 and d_2 between 1 and T.
2. Initialize two arrays of lists $\mathrm{BAD}_0[\cdot]$ and $\mathrm{BAD}_1[\cdot]$ (initially empty).
3. 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 simulation until the d_1th pair of users (A, B) is mentioned. (This may be from an oracle query $H_1(A, B, \cdot)$, or from an *initialize user instance* query with user ID A and partner ID B, or vice-versa.) If we guessed d_1 correctly, this pair will be the identities of the users in the "bad event."
4. Once A and B are set, continue as in the original simulator, except:
   (a) A0 query to the d_2th initiator instance $(\hat{i},\hat{j})$ with $ID_{\hat{i}} = B$ and $PID_{\hat{i}\hat{j}} = A$: respond with $\hat{m} = Y$.

   If we guessed d_2 correctly, this will be the initiator instance in the "bad event." Note that it is OK to set $\hat{m} = Y$, since Y is assumed to be uniformly distributed.

   (b) $H'_1(A, B, \pi)$: If $\pi = \pi^*$, then set $b[\pi] = b[\pi^*] = 0$. Otherwise, let $b[\pi] \in_R \{0, 1\}$. Respond with $(X^{b[\pi]} \cdot h^q g^{\alpha'[A,B,\pi]} \bmod p) + \beta' p$, for $h \in_R Z_p^*$, $\alpha'[A, B, \pi] \in_R Z_{p-1}$ and $\beta' \in_R Z_{\lfloor 2^\eta / p \rfloor}$.

   (c) A2(μ) on $(\hat{i},\hat{j})$: Set $\hat{\mu} = \mu$. Then do *start session* with the *dangling connect* connection assignment.

   Note that if the bad event is about to occur, then we don't need to be concerned for case 1 of A2. Also, the original simulator would not expose, since that would imply a successful *test instance password* on $(\hat{i},\hat{j})$, and that is also ruled by the bad event. Now, it is easy to see that the only way the above behavior might be distinguishable from the original simulator is if this query was supposed to cause an abort (i.e., an initiator failure). However, this would imply that the bad event has already occurred (i.e., the two bad queries have already been asked), and so we are no longer interested in preserving indistinguishability.

(d) $H_3(A, B, m, \mu, \sigma, \pi)$ query, with a tuple (i', j', m, μ, z) recorded, $A = PID_{i'j'}$, $B = ID_{i'}$: If $\pi = \pi^*$, then behave as in the original simulator (this will be correct, since $b[\pi^*] = 0$). Otherwise, return $k \in_R \{0,1\}^\kappa$.

It is easy to see that the only way this behavior might be distinguishable from the original simulator is if this query was supposed to cause an abort (i.e., a responder failure). However, this is not a problem: If the bad event has already occurred, then we are no longer interested in preserving indistinguishability. On the other hand, if the bad event is still about to occur, then this query couldn't cause a responder failure (since the bad event, by definition, can't occur after a responder failure).

(e) $H_3(A, B, \hat{m}, \mu, \sigma, \pi)$ query (note that this case will not, w.o.p., overlap with the preceding one since it would imply that some responder instance (i', j') has received its m from an acceptable partner, and yet recorded a tuple—that is impossible, since cases 1 and 2 of B1 do not record tuples):

If $H_1(A, B, \pi)$ and $H_1'(A, B, \pi)$ were queried, then do the following (and otherwise, answer the query as in the original simulator):

First note that if this is a bad query, then

$$\sigma = DH\left(\frac{\hat{m}}{(H_1(A, B, \pi))^r}, \frac{\mu}{(H_1'(A, B, \pi))^r}\right)$$
$$= DH\left(\frac{Y}{g^{r \cdot \alpha[A,B,\pi]}}, \frac{\mu}{X^{b[\pi]r} g^{r \cdot \alpha'[A,B,\pi]}}\right).$$

Now compute

$$\gamma = \sigma \cdot Z^{b[\pi]r} \cdot Y^{r\alpha'[A,B,\pi]} \cdot (\mu \cdot X^{b[\pi]r} \cdot g^{r \cdot \alpha'[A,B,\pi]})^{r\alpha[A,B,\pi]}.$$

Put β on the list $\mathrm{BAD}_{b[\pi]}[\mu]$. Then 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 *test instance password* on $(\hat{\imath}, \hat{\jmath})$ can occur before the bad event.

At the end of the simulation, check whether the lists $\mathrm{BAD}_0[\hat{\mu}]$ and $\mathrm{BAD}_1[\hat{\mu}]$ intersect (note that this check can be done in time $O(T \log T)$, by sorting the lists together). If yes, then output "True DH," otherwise (and also if $\hat{\mu}$ has never been set) output "Random."

It is easy to show, similarly to Claim 4, that D is correct with probability at least $\frac{1}{2} + \frac{\epsilon}{4T^2} - \frac{T^2}{2q}$, which is non-negligibly more than $\frac{1}{2}$. □

Proof of Claim 8. If such a query is indeed made with some non-negligible probability ϵ, then we will construct a distinguisher D for DDH. Note that, as in proofs of Claims 4 and 6, we can presume that no failures will occur before the "bad event."

The construction of the distinguisher, with input (X, Y, Z), is similar to that in Claim 3. Thus, we "incorporate" X and Y into the m and μ values, respectively, sent in a matching conversation. The main difference is in the handling of the A2 action that is not part of a matching conversation: In Claim 3, the distinguisher had to guess whether or not to accept, since it couldn't check whether or not the σ value used by the adversary to compute k' is correct. If the distinguisher decided to accept, then it would expose using the key based on the σ value used by the adversary for k'. On the other hand, in PPK, the adversary does not send k'. The σ values can only appear in queries to H_3. Since the adversary could ask many queries to H_3, and we don't know which of them she intended to succeed, we have to pick one by guessing. This reduces the success probability of the distinguisher (as compared to Claim 3) by a factor of $\frac{1}{T}$.

Let us now present the details. The distinguisher, on input (X, Y, Z), runs as follows:

1. Generate random d_1 and d_2 between 1 and T, and random $b \in \{0,1\}$.

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 d_1th A0 query. Say this query is to $(\hat{\imath}, \hat{\jmath})$. Let $A = ID_{\hat{\imath}}$, $B = PID_{\hat{\imath}\hat{\jmath}}$, and $\pi^* = \pi[A, B]$. Reply to this A0 query with $\hat{m} = X$.

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

   (a) B1($\hat{m}$) query to instance (i', j') where $ID_{i'} = B$, and $PID_{i'j'} = A$:
   Generate a random $z_{i'j'} \in Z_q$, and send $\mu = Yg^{z_{i'j'}}$.

   (b) A2(μ) query to $(\hat{\imath}, \hat{\jmath})$ not part of a matching conversation with an acceptable partner:
   The difficulty in this case is that the adversary may ask an H_3 query that should give him the correct key, but we are unable to determine which of the adversary's queries is correct (since we don't know the discrete log of $\hat{m}$). We will resolve the problem by guessing which query is correct: If $b = 0$, our guess is that none of them are correct. If $b = 1$, then our guess is that the d_2th query is correct.

   Proceed as follows: Set $\hat{\mu} = \mu$. Consider all the H_3 queries of form $(A, B, \hat{m}, \hat{\mu}, \cdot, \pi^*)$. If $b = 1$ and there have been d_2 or more of such queries, then expose using the result of the d_2th query. Otherwise, do a *start session* with the *dangling connect* connection assignment.

   Note that we don't need to be concerned about the possibility of an initiator failure, if a bad event is about to occur (as mentioned at the beginning of the proof of this claim). It now easily follows that the above response is indistinguishable from the original simulator, if b and d_2 are guessed correctly.

   (c) $H_3(A, B, X, \mu, \sigma, \pi)$:
   If $\mu = Yg^{z_{i'j'}}$ for some (i', j') and

   $$\sigma = Z \cdot X^{z_{i'j'} - r\alpha'[A,B,\pi]} \cdot g^{r^2 \cdot \alpha[A,B,\pi]\alpha'[A,B,\pi]} / \mu^{r\alpha[A,B,\pi]},$$

   then stop the distinguisher and guess "True DH."

   Otherwise, if $\hat{\mu}$ has been set, $\mu = \hat{\mu}$, $\pi = \pi^*$, $b = 1$, and this is the d_2th query of form $H_3(A, B, \hat{m}, \hat{\mu}, \cdot, \pi^*)$, then respond with the key of $(\hat{\imath}, \hat{\jmath})$ (note that the distinguisher is playing the ring master, and so has access to all the keys). Otherwise, return $k \in_R \{0,1\}^\kappa$.

   Note that we don't have to be concerned with the possibility of an initiator failure, as above. It is then easy to see that the response is indistinguishable from the original simulator, as long as b and d_2 have been guessed correctly.

4. If the simulation ends without outputting "True DH," then output "Random."

Note that if the adversary does make a bad query, we have probability at least $1/T$ of guessing the correct initiator user instance, and probability at least $1/2T$ of answering the A2 and H_3 queries to that user instance correctly (thus allowing the DDH distinguisher to continue in a way that is indistinguishable from the regular simulator).

	Current							
Earlier	*get verifier*	H_0	H_{2a}	H_{2b}	H_3	B1	B1$'$	A2
get verifier		Explicit						
H_0	Explicit							
H_{2a}						ϵ	ϵ	
H_{2b}								Claim 12
H_3								Claim 12
B1			Claim 12			ϵ	ϵ	
B1$'$			Explicit			ϵ	ϵ	
A2				Claim 12	Claim 12			ϵ

Table 4: Possible shadowings in the PAK-X simulator. This table is to be interpreted in the same way as Table 2 (see page 20).

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

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

where the term T/q comes from the probability of the adversary "guessing" a random Z correctly on a random oracle query. Thus, the probability that D guesses correctly is at least $\frac{1}{2} + \frac{\epsilon}{4T^2} - \frac{T}{2q}$, which is non-negligibly more than $\frac{1}{2}$. □

C Security of the PAK-X Protocol

C.1 The Simulator

The proof of simulatability of PAK-X is similar in structure to that of PAK. 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, a, k)).

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

The possible shadowings in the PAK-X simulator are shown in Table 4.

We will let (i, j), (i', j'), A, B, and π^* have the same meaning as in the proof of PAK (see the beginning of Appendix A.1). The notions of *acceptable partner* and *matching conversation* will also retain their meaning. For brevity, we will write $H_0(\{A, B\}, \pi)$ to mean $H_0(\min(A, B), \max(A, B), \pi)$. In addition, we will write v^* and V^* to mean $H_0(\{A, B\}, \pi^*)$ and g^{v^*}, respectively. (Note that when

v^* and V^* appear in expressions for implicit oracle calls, their values might not yet be determined, i.e., the H_0 queries may not yet have been made.)

The simulator will follow the same general rules as described in Appendix A.1. Responses to *deliver message* and *random oracle* operations that do not fall under those rules, as well as to the new *get verifier* operation, are done as follows:

1. $H_0(\{A, B\}, \pi)$

   We presume that A and B are valid user IDs, say of users i and i' (otherwise, this query would be handled by one of the general rules, as mentioned in Appendix A.1). Perform a *test password* on (i, i') (this is legal, since an *impl* operation will be made next to record the result of this oracle query). Now consider two cases:

   Case 1 No *get verifier* operation has yet been performed on (i, i').
   Generate and store $v[\{A, B\}, \pi] \in_R \{0,1\}^{|q|+\kappa}$. Respond with $v[\{A, B\}, \pi]$.
   This response is indistinguishable from the real system, since the only prior implicit query that could shadow this one is in *get verifier* on (i, i'). However, this case assumes that no *get verifier* operation has yet been performed on (i, i').

   Case 2 A *get verifier* operation has been performed on (i, i').
   In this case, the *test password* operation above has returned whether or not $\pi = \pi^*$. If $\pi \neq \pi^*$, then return $v \in_R \{0,1\}^{|q|+\kappa}$. If $\pi = \pi^*$ (in which case this query is shadowed), then return $v[\{A, B\}]$ (which has been set while processing *get verifier*—see the description of *get verifier* below).

2. $H'_0(A, B, c)$

   Return $u \in_R \{0,1\}^{|q|+\kappa}$.

3. $H_1(A, B, V)$

   Generate $\alpha[A, B, \pi] \in_R Z_q$ and store it. Then generate $h \in_R Z_p^*$ and $\beta \in_R Z_{\lfloor 2^\eta/p \rfloor}$, and return $(h^q g^{\alpha[A,B,\pi]} \bmod p) + \beta p$. Note that this will be indistinguishable from a random bit string of length η, since $h^q g^{\alpha[A,B,\pi]} \bmod p$ is a random element from Z_p^*[11] and $\frac{2^\eta \mathrm{mod} p}{2^\eta}$ is negligible.

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

   Case 1 Some prior B1 query has recorded a tuple of the form $(i', j', m, \mu, y_{i'j'}, c_{i'j'}, k_{i'j'})$, with $A = PID_{i'j'}$ and $B = ID_{i'}$, for some values of $y_{i'j'}$, $c_{i'j'}$, and $k_{i'j'}$. This implies that a B1(m) query was made to (i', j') and returned $(\mu, g^{H'_0(A,B,c_{i'j'})}, k_{i'j'})$, 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. Consider two cases:

   Case 1a Query $H_1(A, B, V)$ has been asked, $(\frac{m}{(H_1(A,B,V))^r})^{y_{i'j'}} = \sigma$, and $\tau = V^{H'_0(A,B,c_{i'j'})}$.
   The simulator proceeds as follows: If there has been a successful guess π^* on $\{A, B\}$ and $V = g^{H_0(\{A,B\},\pi^*)}$, then call this H_{2a} query a *successful* H_{2a} *query on* (i', j').

[11] To see this, note that h^q is a random element from the subgroup of order r in Z_p^* and $g^{\alpha[A,B,\pi]}$ is a random element of the subgroup of order q in Z_p^*.

If there hasn't been a successful guess on $\{A, B\}$, then check all the H_0 queries, and look for π that results in $V = g^{H_0(\{A,B\},\pi)}$. If such a π is found (there can be at most one such π, w.o.p., by the randomness of H_0), and there never was an unsuccessful guess on $\{A, B\}$ for π:

(a) If a *test instance password* operation has not previously been performed on (i', j'), perform a *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')*.

(b) If a *test instance password* operation has previously been performed on (i', j'), then abort. We will call this event *an H_{2a} failure.*

In explanation, let's note that V is independent of V^*, unless either the discrete log of V has been returned by a prior H_0 query, or V has been returned by a *get verifier* query (in which case $V = V^*$). The following claim shows that even in the latter case, this H_{2a} query can't be "correct" unless a prior H_0 query resulted in the discrete log of V (or else there has been a successful guess on $\{A, B\}$):

Claim 9. *Let (μ, a, k) be returned by a B1(m) query to (i', j'). Let $A = PID_{i'j'}$ and $B = ID_{i'}$. Suppose* get verifier *is performed on $\{A, B\}$ (either before or after the B1 action), and returns V^*. Then, w.o.p., no query*

$$H_{2a}(A, B, m', \mu', a, \mathrm{DH}(\mu', \frac{m'}{(H_1(A, B, V^*))^r}), \mathrm{DH}(a, V^*), V^*),$$

for any m' and μ', will be made, unless by the time of the H_{2a} query there has been either a successful guess on $\{A, B\}$, or an $H_0(\{A, B\}, \pi)$ query, for some π, with $V^* = g^{H_0(\{A,B\},\pi)}$.

The proof of the claim appears in Appendix C.2.

Case 1b Otherwise.

Do nothing.

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

$$H_{2a}(A, B, m, \mu, g^{H_0'(A,B,c_{i'j'})}, \mathrm{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), (V^*)^{H_0'(A,B,c_{i'j'})}, V^*).$$

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. there will be no failure:

Claim 10. *W.o.p., no H_{2a} failure will occur.*

The proof appears in Appendix C.2.

As can be seen, an H_{2a} query is the only place in the simulation that a *test instance password* query can be made on a responder instance. We can therefore state the following claim (similarly to PAK and PPK):

Claim 11. *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 reason is that an H_{2a} query can result in a *test instance password* query on (i', j') only if a tuple $(i', j', m, \mu, y_{i'j'}, c_{i'j'}, k_{i'j'})$ 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.

Case 2 No tuple of the form $(i', j', m, \mu, y_{i'j'}, c_{i'j'}, k_{i'j'})$ 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 $\sigma = \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,V))^r})$. However, by the following claim, w.o.p. this will not occur.

Claim 12. *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 to either H_{2a}, H_{2b}, or H_3, that includes $(A, B, m, \mu, \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,V))^r}), V)$, for any V.*

This claim is proved in the same way as Claim 3 (with obvious changes to accommodate the differences between PAK and PAK-X).

5. $H_{2b}(A, B, m, \mu, \sigma, a, k, c, V)$

   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 12. (Note that the H_{2b} queries in case 2 of the A2 action are explicit, i.e., all of their arguments are known, and so a query-response pair would be stored for them.)

6. $H_3(A, B, m, \mu, \sigma, c, V)$

   Return $K \in_R \{0,1\}^\kappa$.

   As for H_{2b} queries, indistinguishability easily follows by Claim 12.

7. A0 query to (i, j)

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

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

   Generate $y_{i'j'} \in_R Z_q$, $c_{i'j'} \in_R \{0,1\}^\kappa$, $k_{i'j'} \in_R \{0,1\}^\kappa$. Send $\mu = g^{y_{i'j'}}$, $a = g^{H_0'(A,B,c_{i'j'})}$, and $k = k_{i'j'}$. Now consider two cases:

   Case 1 The value of m has been sent by an acceptable partner.

   The simulation corresponds to the implicit oracle query

$$k_{i'j'} \oplus c_{i'j'} = H_{2a}(A, B, m, \mu, a, \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,V^*))^r}), (V^*)^{H_0'(A,B,c_{i'j'})}, V^*).$$

   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 $(i', j', m, \mu, y_{i'j'}, c_{i'j'}, k_{i'j'})$. It is important to note (for Claim 11), 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 the previous case, we have an implicit oracle query

$$k_{i'j'} \oplus c_{i'j'} = H_{2a}(A, B, m, \mu, a, \mathrm{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), (V^*)^{H'_0(A,B,c_{i'j'})}, V^*),$$

which is indistinguishable as above by the randomness of μ values.

9. A2(μ, a, 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:

$$k'_{ij} = H_{2b}(A, B, m, \mu, \mathrm{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), a, k_{i'j'}, c_{i'j'}, V^*),$$
$$K_{ij} = H_3(A, B, m, \mu, \mathrm{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), c_{i'j'}, V^*),$$

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

By Claim 12, 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 c for which an $H'_0(A, B, c)$ query has been made and $a = g^{H'_0(A,B,c)}$. (W.o.p. there will be at most one such value of c, due to the randomness of H'_0.) If no such c is found, then reject. This is indistinguishable from the real system, since w.o.p. the a received would not pass the test $a = g^{H'_0(A,B,c)}$, no matter what c would have been computed in the real system.

Otherwise (if such a c is found), look for a value of V for which an $H_1(A, B, V)$ query has been made, and another query

$$H_{2a}(A, B, m, \mu, a, \mu^{w - r\alpha[A,B,V]}, V^{H'_0(A,B,c)}, V)$$

has been made and returned $k \oplus c$. (W.o.p. there will be at most one such value of V, due to the randomness of H_{2a}.) If no such V 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 $k = c \oplus H_{2a}(A, B, m, \mu, a, \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,V^*))^r}), (V^*)^{H_0'(A,B,c)}, V^*)$ would be independent of the adversary's view).

Suppose such a V is found. Consider two cases:

Case 2a A *get verifier* query has been performed on (i, i') and returned V^*.

If $V \neq V^*$, then reject. This is clearly indistinguishable from the real system, since the correct k is independent of the adversary's view (as the V argument to H_{2a} is incorrect).

If $V = V^*$, then proceed as follows:

(a) Set $k' = H_{2b}(A, B, m, \mu, \mu^{w - r\alpha[A,B,V]}, a, k, c, V)$.

(b) Send k'.

(c) Accept.

(d) Set $K = H_3(A, B, m, \mu, \mu^{w - r\alpha[A,B,V]}, c, V)$.

(e) Expose using session key K (this is allowed, since (i, j) is a client instance, and a *get verifier* query has been performed on (i, i')).

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.

Case 2b No *get verifier* query has been performed on (i, i').

Look for a value of π such that $H_0(\{A, B\}, \pi)$ has been asked and $V = g^{H_0(\{A,B\},\pi)}$. (W.o.p. there will be at most one such value of π, due to the randomness of H_0.) 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 (as the V argument to H_{2a} is incorrect w.o.p.).

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 $k' = H_{2b}(A, B, m, \mu, \mu^{w - r\alpha[A,B,V]}, a, k, c, V)$.

(b) Send k'.

(c) Accept.

(d) Set $K = H_3(A, B, m, \mu, \mu^{w - r\alpha[A,B,V]}, c, V)$.

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

This is correct similarly to above.

10. 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 11, 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 $(\mu, a, k_{i'j'})$ 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 The current value m has been sent by some acceptable partner (i,j), but the triple received by (i,j) in A2 is not equal to $(\mu, a, k_{i'j'})$ sent by (i',j').

Reject. By Claim 12, w.o.p., the H_{2b} query that would produce the correct value of k' has never been asked. Also, no implicit query could have produced the correct k' (note that (i,j) did not have a matching conversation, and thus did not "make" an implicit H_{2b} query; also, any other initiator instance would, w.o.p., have a different value of m). Thus, the correct value of k' is independent of the adversary's view, and so it is safe to reject.

Case 4 Otherwise. (Note that in this case, the current value of m has not been sent by an acceptable partner, and thus a tuple must have been recorded by (i',j') in case 2 of the B1 action.)

Look for a value of V such that an $H_1(A,B,V)$ query has been asked and a query $H_{2b}(A,B,m,\mu,(\frac{m}{(H_1(A,B,V))^r})^{y_{i'j'}}, a, k_{i'j'}, c_{i'j'}, V)$ has been asked and returned k'. (W.o.p. there will be at most one such value of V, due to the randomness of H_{2b}.) If no such V 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 V is found), consider two cases:

Case 4a There has been a successful guess π on $\{A,B\}$.

If $V = g^{H_0(\{A,B\},\pi)}$, then expose with key

$$K = H_3(A,B,m,\mu,(\frac{m}{(H_1(A,B,V))^r})^{y_{i'j'}}, c_{i'j'}, V).$$

This is clearly allowed and indistinguishable from the real system.

Otherwise, reject. This is clearly indistinguishable from the real system, since the correct k' is independent of the adversary's view (as the V argument to H_{2b} is incorrect).

Case 4b There has not been a successful guess on $\{A,B\}$.

In this case, reject.

Let's show that this behavior is indistinguishable from the real system. Suppose we were instead supposed to accept. The value $c_{i'j'}$ is independent of the adversary's view, unless the adversary has queried

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

Since the value of $c_{i'j'}$ was present in one of the adversary's H_{2b} queries, it follows that, w.o.p., the adversary has indeed made the above H_{2a} query. Now, from Claim 9 and the discussion preceding that claim, it follows that, w.o.p., the adversary has made a prior query $H_0(\{A, B\}, \pi)$, for some π, such that $V^* = g^{H_0(\{A,B\},\pi)}$. However, that would imply that $H_0(\{A, B\}, \pi) = v^*$. It is easy to see that, w.o.p., $H_0(\{A, B\}, \pi)$ would only return v^* if $\pi = \pi^*$. From the H_{2a} action description above, it is easy to see that this situation would, w.o.p., result in a successful guess on $\{A, B\}$ (made during the H_{2a} query). Thus, there has already been a successful guess on $\{A, B\}$, which contradicts our original assumption.

11. *get verifier* on user pair (i, i')

    Let A and B be the IDs of users i and i', respectively. Send a *get verifier* query on (i, i') to the ring master (i.e., in the ideal world). The ring master will return whether any prior *test password* query on (i, i') was successful. If there was a successful guess on $\{A, B\}$ for some π (whether through *test instance password* or *test password*), then return $g^{H_0(\{A,B\},\pi)}$.

    Otherwise (i.e., if no query was successful), generate and store $v[\{A, B\}] \in_R \{0, 1\}^{|q|+\kappa}$. Return $g^{v[\{A,B\}]}$. This corresponds to the implicit oracle query

    $$v[\{A, B\}] = H_0(\{A, B\}, \pi^*).$$

    Clearly, this implicit query cannot shadow any prior explicit H_0 query, or else some prior *test password* would have been successful.

C.2 Proofs of Claims

Proof of Claim 9. 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 $(\hat{\mu}, \hat{a}, \hat{k})$,

2. *get verifier* is performed on $\{A, B\}$ and returns V^* (either before or after the B1 action), and

3. query

   $$H_{2a}(A, B, m, \mu, \hat{a}, \mathrm{DH}(\mu, \frac{m}{(H_1(A, B, V^*))^r}), \mathrm{DH}(\hat{a}, V^*), V^*)$$

   (for some m and μ) is made, before there has been either a successful guess on $\{A, B\}$ or an $H_0(\{A, B\}, \pi)$ query, for some π, with $V^* = g^{H_0(\{A,B\},\pi)}$.

We will then construct a distinguisher D for DDH. Let (X, Y, Z) be the challenge DDH instance. The idea of the construction is to "incorporate" X into the result of the *get verifier* query, and Y into the a values sent by B instances.

The construction is as follows:

1. Generate random 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 simulation until the dth pair of users (A, B) is mentioned. (This may be from a *get verifier* query on users with IDs A and B, or from an *initialize user instance* query with user ID A and partner ID B, or vice-versa.) If we guessed d correctly, this pair will be the identities of the users in the "bad event."

3. Once A and B are set, continue as in the original simulator, except:

   (a) *get verifier* on users with IDs A and B:
   Respond with X. Note that this results in $V^* = X$.
   If the "bad event" is about to occur, then this response is correct, since there could not have been a successful guess on $\{A, B\}$.

   (b) B1(m) query to instance (i', j') with $ID_{i'} = B$ and $PID_{i'j'} = A$:
   Generate $y_{i'j'} \in_R Z_q$, $z_{i'j'} \in_R Z_q$, and $k_{i'j'} \in_R \{0,1\}^\kappa$. Send $\mu = g^{y_{i'j'}}$, $a = Yg^{z_{i'j'}}$, and $k = k_{i'j'}$.
   Note that this corresponds to an implicit query

$$y + z_{i'j'} = H_0'\Bigg(A, B, k_{i'j'} \oplus H_{2a}\Bigg(A, B, m, \mu, Yg^{z_{i'j'}}, \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,X))^r}), \mathrm{DH}(Yg^{z_{i'j'}}, X), X\Bigg)\Bigg).$$

   W.o.p., this implicit query will not shadow any prior H_0' query (whether implicit or explicit), by the randomness of $k_{i'j'}$.

   (c) $H_0(\{A, B\}, \pi)$ query: Generate and store $v[\{A, B\}, \pi] \in_R \{0,1\}^{|q|+\kappa}$. Respond with $v[\{A, B\}, \pi]$.
   If the "bad event" is about to occur and d has been guessed correctly, then this response is appropriate, as the "bad event" rules out H_0 queries that would return the discrete log of V^*.

   (d) $H_0'(A, B, c)$ query: Behave as in the original simulator.
   The only way this behavior could be distinguishable from the original simulator is if this query was shadowed by an implicit query from a B1 action. However, it is easy to see that this will not occur, w.o.p., unless the adversary first makes the query

$$H_{2a}(A, B, m, \mu, Yg^{z_{i'j'}}, \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,X))^r}), \mathrm{DH}(Yg^{z_{i'j'}}, X), X),$$

   (since without such an H_{2a} query, the value of c that would cause the shadowing would be independent of the adversary's and simulator's view). However, such an H_{2a} query would imply that the bad event has already occurred, in which case we are no longer concerned with indistinguishability.

   (e) A2(μ, a, k) query to instance (i, j) with $ID_i = A$ and $PID_{ij} = B$, not part of a matching conversation: If the value of a is not equal to $Yg^{z_{i'j'}}$ for any (i', j'), then proceed as in the original simulator. Otherwise, reject.
   In the first case, the behavior is clearly indistinguishable. On the other hand, suppose $a = Yg^{z_{i'j'}}$ for some (i', j'). Our response could be incorrect if query

$$H_{2a}(A, B, m, \mu, Yg^{z_{i'j'}}, \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,X))^r}), \mathrm{DH}(Yg^{z_{i'j'}}, X), X)$$

   has been made. However, that would imply that the bad event has already occurred, in which case we are no longer concerned with indistinguishability.

(f) 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 has 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 $\{A, B\}$, and that would contradict the definition of the "bad event" (unless the "bad event" has already occurred, in which case we don't care whether or not the response was correct).

(g) $H_{2a}(A, B, m, \mu, a, \sigma, \tau, X)$ query where $a = Yg^{z_{i'j'}}$ for some (i', j') and $\tau = ZX^{z_{i'j'}}$: Stop the distinguisher and guess "True DH."

4. If the simulation ends without outputting "True DH," then output "Random."

Note that if the "bad event" does occur, we have probability at least $1/T$ of guessing the correct pair of users. Now, the probability that the distinguisher D guesses correctly is at least

$$\begin{aligned}
\Pr(D \text{ is correct}) &= \Pr(D \text{ guesses "DH True"} | \text{DH instance}) \Pr(\text{DH instance}) \\
&\quad + \Pr(D \text{ guesses "Random"} | \text{Random instance}) \Pr(\text{Random instance}) \\
&\geq \left(\frac{\epsilon}{T}\right)\left(\frac{1}{2}\right) + \left(1 - \frac{T}{q}\right)\left(\frac{1}{2}\right),
\end{aligned}$$

where the term T/q comes from the probability of the adversary "guessing" a random Z correctly on a random oracle query. Thus, the probability that D guesses correctly is at least $\frac{1}{2} + \frac{\epsilon}{2T} - \frac{T}{2q}$, which is non-negligibly more than $\frac{1}{2}$. □

Proof of Claim 10. We will call an oracle query of form $H_{2a}(A, B, m, \mu, a, \sigma, \tau, V)$ "bad" if

1. some responder instance (i', j'), with $ID_{i'} = B$ and $PID_{i'j'} = A$, has sent μ (w.o.p., by randomness of μ values, there will be at most one such instance),
2. $\sigma = \mathrm{DH}(\mu, \frac{m}{(H_1(A,B,V))^r})$, and
3. $\tau = \mathrm{DH}(a, V)$.

Suppose that with some nonnegligible probability ϵ there will be an H_{2a} failure. This implies that 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 $(\hat{\mu}, \hat{a}, \hat{k})$, and
2. at least two "bad" queries are made with $(A, B, \hat{m}, \hat{\mu})$ and distinct values of V, before there is a successful guess on $\{A, B\}$.

(Note that we speak of distinct values of V, since, w.o.p., distinct values of π will result in distinct values of V.) We will then show how to construct a distinguisher D for the DDH problem, with input (X, Y, Z).

The idea of the construction is similar to the one in the proof of Claim 1. Again, we will "incorporate" X into half of the H_1 responses (randomly), and Y into B1 responses (i.e., the μ values). The main difference is in the handling of the A2 action: In Claim 1, the distinguisher would check the H_{2a} queries that used the correct value of π. In this claim, the distinguisher needs to check the H_{2a} queries that used the correct value of V. Note that the correct value of V is not determined until either a *get verifier* query is made, or H_0 is queried with the correct value of π.

Our distinguisher D for input (X, Y, Z) runs as follows:

1. Generate random d between 1 and T.

2. Initialize two lists BAD_0 and BAD_1 (initially empty).

3. 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 simulation until the dth pair of users (A, B) is mentioned. (This may be from an oracle query $H_1(A, B, \cdot)$, or from an *initialize user instance* query with user ID A and partner ID B, or vice-versa.) If we guessed d correctly, this pair will be the identities of the users in the "bad event."

   We will say that V^* *is determined* when either the query $H_0(\{A, B\}, \pi^*)$ or the query *get verifier* on $\{A, B\}$ is made. Note that, since the distinguisher itself generates π^*, we will be able to detect whether or not V^* has been determined, and we will also be able to compute V^* after that point.

4. Once A and B are set, continue as in the original simulator, except:

   (a) B1(m) query to instance (i', j') with $ID_{i'} = B$ and $PID_{i'j'} = A$: generate $z_{i'j'} \in_R Z_q$, set $\mu = Yg^{z_{i'j'}}$, and then proceed as in the original simulator.

   (b) $H_1(A, B, V)$: If V^* has been determined and $V = V^*$, then set $b[V] = b[V^*] = 0$. (Note that w.o.p. the query $H_1(A, B, V^*)$ will not be made until after V^* has been determined.) Otherwise, let $b[V] \in_R \{0, 1\}$. Respond with $(X^{b[V]} \cdot h^q g^{\alpha[A,B,V]} \bmod p) + \beta p$, for $h \in_R Z_p^*$, $\alpha[A, B, V] \in_R Z_{p-1}$, and $\beta \in_R Z_{\lfloor 2^\eta / p \rfloor}$.

   (c) A2(μ, a, 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 only check oracle queries with $V = V^*$ (and if V^* has not yet been determined, then simply reject).

   Note that $(H_1(A, B, V^*))^r = g^{r \cdot \alpha[A,B,V^*]}$, since $b[V^*] = 0$. Thus, the original simulator will work correctly in this case. It is safe to ignore oracle queries with $V \neq V^*$, since those wouldn't lead to an accept (as the *test instance password* query would fail).

   (d) 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 has been guessed correctly, then this response is appropriate: If this query was supposed to result in an accept, then there would be a successful guess on $\{A, B\}$ before the second "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.

   (e) $H_{2a}(A, B, m, \mu, a, \sigma, \tau, V)$ query:

   If $H_1(A, B, V)$ was queried and there is an instance (i', j') with $B = ID_{i'}$ that was queried with B1(m) and returned $\mu = Yg^{z_{i'j'}}$, then do the following (and otherwise, answer the query as in the original simulator):

First note that if this is a bad query, then

$$\sigma = DH(\mu, \frac{m}{(H_1(A,B,V))^r}) = DH(Yg^{z_{i'j'}}, \frac{m}{X^{b[V]r}g^{r\cdot\alpha[A,B,V]}}).$$

Now compute

$$\gamma = \sigma m^{-z_{i'j'}} X^{b[V]rz_{i'j'}} Z^{b[V]r} Y^{r\alpha[A,B,V]} g^{r\cdot\alpha[A,B,V]z_{i'j'}}.$$

Put γ on the list $\mathrm{BAD}_{b[V]}$. Then 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 whether the lists BAD_0 and BAD_1 intersect (note that this check can be done in time $O(T \log T)$, by sorting the lists together). If yes, then output "True DH," otherwise output "Random."

Note that the values stored in BAD_0 and BAD_1 are simply guesses of $DH(m,Y)$, assuming $Z = DH(X,Y)$. If Z is random, then the probability of an intersection between the lists would be at most T^2/q by the union bound, since Z^r would be a random element of $G_{p,q}$ (note that q and r are relatively prime, and $m \neq 0 \bmod p$ because of the test by B).

On the other hand, suppose $Z = \mathrm{DH}(X,Y)$. If the adversary makes two "bad" queries for the pair of users (A,B), for values V_1, V_2 with $b[V_1] \neq b[V_2]$, then the distinguisher will correctly answer "True DH" (since each "bad" query will result in $\gamma = \mathrm{DH}(m,Y)$). The probability of the "bad event" is ϵ. The probability of guessing d correctly is $\frac{1}{T}$. The probability of $b[V_1] \neq b[V_2]$ for $V_1 \neq V_2$ is $\frac{1}{2}$. All of these events are independent.

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

$$\begin{aligned}\Pr(D \text{ is correct}) &= \Pr(D \text{ guesses "DH True"}|\text{DH instance}) \Pr(\text{DH instance}) \\ &\quad + \Pr(D \text{ guesses "Random"}|\text{Random instance}) \Pr(\text{Random instance}) \\ &\geq \left(\frac{\epsilon}{2T}\right)\left(\frac{1}{2}\right) + \left(1 - \frac{T^2}{q}\right)\left(\frac{1}{2}\right).\end{aligned}$$

Thus the probability that D is correct is at least $\frac{1}{2} + \frac{\epsilon}{4T} - \frac{T^2}{2q}$, which is non-negligibly more than $\frac{1}{2}$. □