Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman

Victor Boyko[1], Philip MacKenzie[2], and Sarvar Patel[2]

[1] MIT Laboratory for Computer Science, boyko@theory.lcs.mit.edu
[2] Bell Laboratories, Lucent Technologies, {philmac, sarvar}@lucent.com

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. 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.

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 [30]).

Over the past decade, many password-authenticated key exchange protocols that promised increased security have been developed, e.g., [8, 9, 19, 18, 28, 21, 22, 24, 29]. Some of these have been broken [26], and, in fact, only two very recent ones have been formally proven secure. The SNAPI protocol in [25] is proven secure in the random oracle model [5, 6, 14], assuming the security of RSA (and

also Decision Diffie-Hellman [11], 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 protocol and the protocol of [3] were developed independently.)

We present a new password-authenticated key exchange protocol called **PAK** (Password Authenticated Key exchange), which provides perfect forward secrecy and is proven to be as secure as Decision Diffie-Hellman in the random oracle model. Compared to the protocol of [25], PAK (1) does not require the RSA assumption, (2) has fewer 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.) In the full paper [13], we also show how the security of PAK can be related to the Computational Diffie-Hellman problem.

In addition to PAK, we also show a more efficient 2 round protocol called PPK (Password-Protected Key exchange) 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. 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 [27] (which follows the work of [2]), enhanced with notions of password authentication security from [20, 25]. 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

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. 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 paper.

In fact, we will focus on the harder problem of *remote user authentication*. 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 expect that a powerful adversary would be capable of eavesdropping, deleting and inserting messages, and also initiating sessions.

Now we 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 symmetric or private/public) 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.

Password-Authentication Protocols: Traditional password 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 likely passwords. Obviously, we can't prevent the adversary from trying the passwords on-line, but such an attack can be made infeasible by 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 an 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. [23] 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. Other protocols for this scenario were developed in [19, 20, 12].

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 [15]). 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 password authenticated key exchange protocols were proposed [9, 19, 18, 28, 21, 22, 24, 29]. 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.[1] All of these protocol proposals contained informal arguments for security. However, the fact that some versions of these protocols were subsequently shown to be insecure [26] should emphasize the importance of formal proofs of security.

Models for Secure Authentication and Key Exchange: Bellare and Rogaway [4] presented 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, [25] and [3] present extensions to the model to allow for password authentication. Halevi and Krawczyk [20] and Boyarsky [12] present models which include both passwords and asymmetric keys (since they deal with password-based protocols that also 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 [27] refines and extends their model. We present a further extension of [27] that includes password authentication.

3 Model

For our proofs, we extend the formal notion of security for key exchange protocols from Shoup [27] to password-authenticated key exchange. We assume that the adversary totally controls the network, a la [4].

Security for key exchange in [27] 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 Ideal system

Our ideal system follows [27], except for the addition of password authentication and a slight modification to explicitly handle mutual authentication. We assume that there is a set of (honest) *users*, indexed $i = 1, 2, \ldots$. Each user i may have

[1] Naturally, given the data from a server, an attacker could perform an off-line dictionary attack, since the server must know something to verify a user's password.

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 since 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' (see the discussion below on the distribution from which these passwords are generated). 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: (`"initialize user instance"`, 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 user instance (i, j) to terminate.

test instance password
This is called with an instance (i, j) and a password guess π. The returned result is either `true` or `false`, depending on whether $\pi = \pi[ID_i, PID_{ij}]$. If

the result is `true`, then this 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:

open for connection from (i', j')**.** This requires that $role(i, j)$ is "open," (i', j') has been initialized and has not been terminated, $role(i', j')$ is "connect," $PID_{ij} = ID_{i'}$, $PID_{i'j'} = ID_i$, no other instance is open for connection from (i', j'), and no *test instance password* 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').

connect to (i', j')**.** This requires that $role(i, j)$ is "connect," (i', j') has been initialized and has not been terminated, $role(i', j')$ 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 is still open for connection from (i, j), and no *test instance password* 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.

expose. 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.

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.

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

Discussion (password authentication): Our system correctly describes 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 on an instance after a successful key exchange. Our ideal model explicitly uses

(ring master generated) passwords, and an adversary can only obtain information about a password by issuing a *test instance password* query for an instance, signifying an impersonation attempt by the adversary against the key exchange protocol run by that instance. (One may think of this as modeling an adversary who attempts to log in to a server by sending a guessed password.)

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.

We also model the ability of the adversary to set up passwords between any users and herself, 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.

3.2 Real System with Passwords

Now we describe the real system in which we assume a password-authenticated key exchange protocol runs. Again, this is basically from [27], except that we do not concern ourselves with public keys and certification authorities, since all authentication is performed using shared passwords.

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: $(\texttt{"initialize user"}, i, ID_i)$]
initialize user instance [Transcript: $(\texttt{"initialize user instance"}, i, j, role(i, j), PID_{ij})$]
set password [Transcript: $(\texttt{"set password"}, i, ID', \pi)$]

application [Transcript: ("application", $f, f(R, \{K_{ij}\})$)]
All above as in the ideal system.

deliver message [Transcript: ("impl", "message", i, j, *InMsg*, *OutMsg*, *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 ("start session", i, j) is added to the transcript, and if *status* is "reject", the record ("terminate instance", i, j) is added to the transcript.

random oracle [Transcript: ("impl", "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 oracles). 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 ("impl", "coins", *coins*).

3.3 Definition of Security

The definition of security for key exchange given in [27] requires

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

We will use this definition for password-authenticated key exchange as well.[2]

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

[2] We can do this because 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 has made.

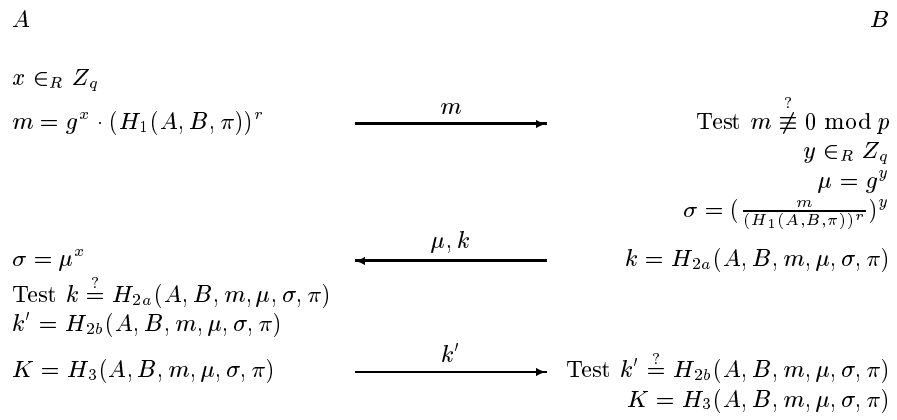


Fig. 1. PAK protocol, with $\pi = \pi[A,B]$. The resulting session key is K. If a "Test" returns false, the protocol is aborted.

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 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.*

Proof: *(Sketch)* The completeness requirement follows directly by inspection. Here we sketch the proof that the simulatability requirement holds. Complete

details are presented in the full paper [13]. The basic technique is essentially that of Shoup [27]. 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.

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 detect 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 respond to the adversary's requests without knowing the actual passwords. This causes us to use random values in place of the results of those random oracle calls that take the password as an argument. We can think of these as "implicit" oracle calls. 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. Specifically, we must make sure the random oracle queries to H_{2a} and H_{2b} are consistent with the k and k' values sent or received by the user instances. This is relatively straightforward (using *test instance password* queries) except when the adversary sends a μ value back to an initiator instance. To be able to determine the password being tested by the adversary in this case, we will make sure the simulator has answered each $H_1(A, B, \pi)$ query using a random value for which it knows the discrete log (after that value is raised to the r).

Indistinguishability The simulation described above is indistinguishable from the real world as long as the simulator does not need to perform a *test instance password* query that is disallowed in the ideal world. Specifically, by the rules of the ideal world, (1) only one of these queries can be made for each user instance, and (2) the query cannot be made at all for any instance that performs a *start session* operation (previously or in the future). So to finish our proof, we need to show that if the adversary can break either rule with non-negligible probability, then we can break the DDH Assumption with non-negligible probability.

The idea of the proof of (2) goes as follows. Say that the offending query is made within the first T queries. (T is bounded by the adversary's running time and must be polynomial.) Take a DDH challenge (X, Y, Z). Run the simulation (playing the ringmaster also, i.e., choosing our own passwords) with the following changes: Choose a random $d \in [0, T]$. On the dth *deliver message* query to initiate a protocol, say for users A and B, set $m = X$. For any B instance that receives

$m = X$, set $\mu = Yg^y$ for some random y. If the adversary makes a query to H_{2a}, H_{2b}, or H_3 with A, B, m, a μ as calculated above, σ, and π, where $\sigma = ZX^y/\mu^{r\alpha_\pi}$ for α_π the discrete log of $(H_1(A, B, \pi))^r$, guess that the DDH challenge is a true DH instance. All other queries are answered in a straightforward way, except that the adversary may make a valid password guess using its own μ and σ, for which the simulator cannot verify the σ value (because the simulator does not know the discrete log of X). In this case we flip a coin to decide whether to accept or not, and continue the simulation. It can be shown that if the adversary is able to break this ideal world rule with probability ϵ, then we will give a correct response to the DDH challenge with probability roughly $\frac{1}{2} + \frac{\epsilon}{4T}$. (The 4 in the denominator comes from the half probability of the DDH challenge being a true DH instance and the half probability of a correct coin flip.)

The idea of the proof of (1) goes as follows. Say that the offending queries occur within the first T queries. Let the DDH challenge be (X, Y, Z). Run the simulation (playing the ringmaster also) with the following changes: Choose a random $d \in [0, T]$. Assume the bad event will occur for the dth pair of users mentioned (either in an $H_1(A, B, \cdot)$ query or an *initialize user instance* with A and partner B) Each time $H_1(A, B, \pi)$ is queried for some π, flip a coin to decide whether to include a factor of X in the return value. For any first message to a B instance with partner A, set $\mu = Yg^y$ for some random y. Note that the σ values used in any pair of H_{2a}, H_{2b}, and H_3 queries for the same A, B, m, μ (where μ was calculated as Yg^y), and using two different password guesses (π_1 and π_2) can be tested against the Z value if exactly one of $H_1(A, B, \pi_1)$ and $H_1(A, B, \pi_2)$ included a factor of X in its calculation. If any of these pairs tests positively for the Z value, guess that the DDH challenge is a true DH instance. All other queries are answered in a straightforward way. It can be shown that if the adversary is able to break this ideal world rule with probability ϵ, then we will give a correct response to the DDH challenge with probability roughly $\frac{1}{2} + \frac{\epsilon}{4T}$. (The 4 in the denominator comes from the half probability of the DDH challenge being a true DH instance and the half probability of the adversary making queries for two passwords in which exactly one included a factor of X in the $H_1()$ calculation.) □

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 3.2.

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 she claims to be, she simply won't be able to obtain the session key of the honest party. The honest party (which could be playing the role of either `"open"` or `"connect"`) would still open a session, but no one would be able to actually communicate

with her on that session.[3] 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:

- 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, even if it has already started a session with a dangling connection assignment.

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 this permissiveness 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. 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.

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 is omitted due to page limits. The basic structure of the proof is very similar to that of the PAK protocol.

[3] In a later version of [27], Shoup also deals with implicit authentication, but in a different way. We feel our solution is more straightforward and intuitive.

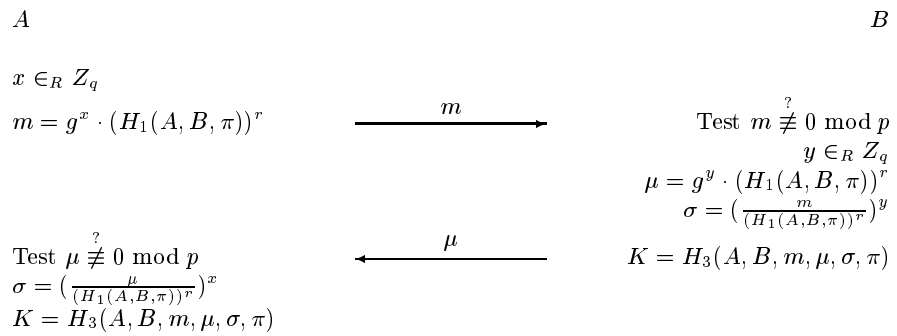


Fig. 2. PPK protocol, with $\pi = \pi[A,B]$. The resulting session key is K.

6 Resilience to Server Compromise—The PAK-X Protocol

6.1 Ideal System with Passwords—Resilience to Server Compromise

Now we 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. To accomplish this, 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.

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 or 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 *ex-*

pose for client instance (i, j) is allowed at any point after a *get verifier* query on users i and 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

In a real system that has any resilience to server compromise, the server must not store the plaintext password. Instead, the server stores a *verifier* to verify a user's password. Thus, the protocol has to specify a PPT 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 $V[\{A, B\}]$.

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 given here:

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

get verifier [Transcript:$(\texttt{"get verifier"}, i, i')$, followed by $(\texttt{"impl"}, \texttt{"verifier"}, i, i', V[ID_i, ID_{i'}])$]
The adversary performs this query with i and i' as arguments, with $V[ID_i, ID_{i'}]$ being returned.

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 is omitted due to page limits. (The technique that allows us to perform authentication where the server stores a verifier instead of the password itself is similar to the technique developed independently in [17] to obtain an efficient encryption scheme secure against an adaptive chosen-ciphertext attack.)

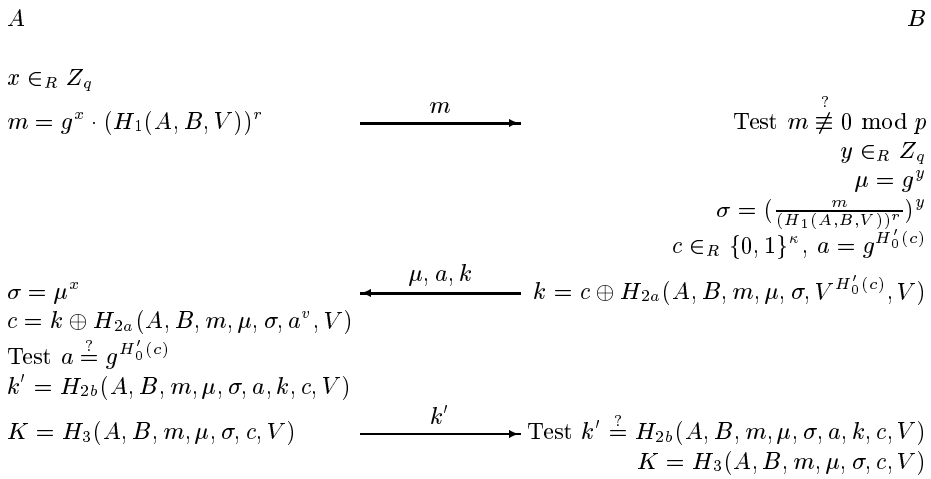


Fig. 3. PAK-X protocol, with $\pi = \pi[A,B]$, $v = v[A,B]$, and $V = V[A,B]$. The resulting session key is K.

Acknowledgements. We would like to thank Daniel Bleichenbacher for an improvement to our method of generating simulated random oracle responses (as shown in the full paper [13]).

References

1. D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *J. 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, pages 419–428.
3. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In EUROCRYPT 2000, pages 140–156.
4. M. Bellare and P. Rogaway. Entity authentication and key distribution. In CRYPTO'93, pages 232–249.
5. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In ACM Security (CCS'93), pages 62–73.
6. M. Bellare and P. Rogaway. Optimal asymmetric encryption. In EUROCRYPT'94, pages 92–111.
7. M. Bellare and P. Rogaway. Provably secure session key distribution—the three party case. In STOC'95, pages 57–66.

8. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In Proceedings of IEEE Security and Privacy, pages 72–84, 1992.
9. S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In ACM Security (CCS'93), 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 ACM Security (CCS'99), pages 63–72.
13. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman (full version). http://www.bell-labs.com/user/philmac/research/pak.ps.gz
14. R. Canetti, O. Goldreich, and S. Halevi. The random oracle methodology, revisited. In STOC'98, pages 209–218.
15. W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Info. Theory*, 22(6):644–654, 1976.
16. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithm. *IEEE Trans. Info. Theory*, 31:469–472, 1985.
17. E. Fujisaki and T. Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In CRYPTO'99, pages 537–554.
18. L. Gong. Optimal authentication protocols resistant to password guessing attacks. In *8th IEEE Computer Security Foundations Workshop*, pages 24–29, 1995.
19. 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.
20. S. Halevi and H. Krawczyk. Public-key cryptography and password protocols. In ACM Security (CCS'98), pages 122–131.
21. D. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communication Review, ACM SIGCOMM*, 26(5):5–20, 1996.
22. D. Jablon. Extended password key exchange protocols immune to dictionary attack. In *WETICE'97 Workshop on Enterprise Security*, 1997.
23. 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.
24. S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Proceedings of the Workshop on Security Protocols*, 1997.
25. P. MacKenzie and R. Swaminathan. Secure network authentication with password information. manuscript.
26. S. Patel. Number theoretic attacks on secure password schemes. In *Proceedings of IEEE Security and Privacy*, pages 236–247, 1997.
27. V. Shoup. On formal models for secure key exchange. IBM Research Report RZ 3120. April, 1999.
28. M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of encrypted key exchange. *ACM Operating System Review*, 29:22–30, 1995.
29. T. Wu. The secure remote password protocol. In NDSS'98, pages 97–111.
30. T. Wu. A real world analysis of Kerberos password security. In NDSS'99.