Appears in *Advances in Cryptology – Eurocrypt '00*, Lecture Notes in Computer Science Vol. 1807, B. Preneel ed., Springer-Verlag, 2000. This version corrects a few minor typos.

Authenticated Key Exchange Secure Against Dictionary Attacks

Mihir Bellare[1], David Pointcheval[2], and Phillip Rogaway[3]

[1] Dept. of Computer Science & Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA, mihir@cs.ucsd.edu, WWW home page: http://www-cse.ucsd.edu/users/mihir

[2] Dépt. d'Informatique–CNRS, École Normale Supérieure, 45 rue d'Ulm, 75230 Paris Cedex 05, France, david.pointcheval@ens.fr, WWW home page: http://www.dmi.ens.fr/~pointche

[3] Dept. of Computer Science, University of California at Davis, Davis, CA 95616, USA, rogaway@cs.ucdavis.edu, WWW home page: http://www.cs.ucdavis.edu/~rogaway

Abstract. Password-based protocols for authenticated key exchange (AKE) are designed to work despite the use of passwords drawn from a space so small that an adversary might well enumerate, off line, all possible passwords. While several such protocols have been suggested, the underlying theory has been lagging. We begin by defining a model for this problem, one rich enough to deal with password guessing, forward secrecy, server compromise, and loss of session keys. The one model can be used to define various goals. We take AKE (with "implicit" authentication) as the "basic" goal, and we give definitions for it, and for entity-authentication goals as well. Then we prove correctness for the idea at the center of the Encrypted Key-Exchange (EKE) protocol of Bellovin and Merritt: we prove security, in an ideal-cipher model, of the two-flow protocol at the core of EKE.

1 Introduction

THE PROBLEM. This paper continues the study of password-based protocols for authenticated key exchange (AKE). We consider the scenario in which there are two entities—a client A and a server B—where A holds a password pw and B holds a key related to this. The parties would like to engage in a conversation at the end of which each holds a session key, sk, which is known to nobody but the two of them. There is present an active adversary $\mathcal{A}$ whose capabilities include enumerating, off-line, the words in a dictionary D, this dictionary being rather likely to include pw. In a protocol we deem "good" the adversary's chance to defeat protocol goals will depend on how much she interacts with protocol participants—it won't significantly depend on her off-line computing time.

The above protocol problem was first suggested by Bellovin and Merritt [6], who also offer a protocol, Encrypted Key Exchange (EKE), and some informal security analysis. This protocol problem has become quite popular, with further papers suggesting solutions including [7, 15–18, 22, 10, 11, 10, 21]. The reason for this interest is simple: password-guessing attacks are a common avenue for breaking into systems, and here is a domain where good cryptographic protocols can help.

Contributions. Our first goal was to find an approach to help manage the complexity of definitions and proofs in this domain. We start with the model and definitions of Bellare and Rogaway [4] and modify or extend them appropriately. The model can be used to define the execution of authentication and key-exchange protocols in many different settings. We specify the model in pseudocode, not only in English, so as to provide succinct and unambiguous execution semantics. The model is used to define the ideas of proper partnering, freshness of session keys, and measures of security for authenticated key exchange, unilateral authentication, and mutual authentication. Some specific features of our approach are: partnering via session IDs (an old idea of Bellare, Petrank, Rackoff, and Rogaway—see Remark 1); a distinction between accepting a key and terminating; incorporation of a technical correction to [4] concerning Test queries (this arose from a counter-example by Rackoff—see Remark 5); providing the adversary a separate capability to obtain honest protocol executions (important to measure security against dictionary attacks); and providing the adversary corruption capabilities which enable a treatment of forward secrecy.

We focus on AKE (with no explicit authentication). Philosophically, AKE is more "basic" than a goal like mutual authentication (MA). Pragmatically, AKE is simpler and takes fewer flows (two instead of three). Earlier work [3] began by defining MA and then embellishing the definition to handle an associated key exchange. Protocol development followed the same course. That approach gets complicated when one adds in the concern for password-guessing security.

Under our approach resistance to dictionary attacks is just a question of advantage vs. resource expenditure. It shows up in theorems, not definitions (once the model is adequately refined). A theorem asserting security of some protocol makes quantitative how much *computation* helps and just how much *interaction* does. One sees whether or not one has security against dictionary attacks by looking to see if maximal adversarial advantage grows primarily with the ratio of interaction to the size of the password space.

In Section 4 we define EKE2, which is essentially the pair of flows at the center of Bellovin and Merritt's Diffie-Hellman based Encrypted Key Exchange protocol [6]. We show that EKE2 is a secure AKE protocol, in the ideal-cipher model. Security here entails forward secrecy.

Related work. Recently people have been trying to get this area onto firmer foundations. The approach has been to build on the ideas of Bellare and Rogaway [3, 4], extending their definitions to deal with dictionary attacks. Lucks [17] was the first work in this vein. Halevi and Krawczyk [14] provide definitions and protocols for password-based unilateral authentication (UA) in the model in which the client holds the public key for the server, a problem which is different from, but related to, the one we are considering. Some critiques of [14] are made by [9], who also give their own, simulation-based notion for password-based UA.

In contemporaneous work to ours MacKenzie and Swaminathan [18], building on [3, 14], give definitions and proofs for a password-based MA protocol, and then a protocol that combines MA and AKE. Boyko, MacKenzie and Patel [10], building on [1, 20], give definitions and a proof for a Diffie-Hellman based protocol. In both papers the authors' motivation is fundamentally the same as our own: to have practical and provably secure password-based protocols.

Ongoing work. In [5] we provide a simple AKE protocol for the asymmetric trust model: the client holds pw and the server holds $f(pw)$, where f is a one-way function. If the adversary corrupts the server she must still expend time proportional to the quality of the password. We are working on the analysis. We are also investigating the security of EKE2 when its encryption function $\mathcal{E}$ is instantiated by $\mathcal{E}_{pw}(x) = x \cdot H(pw)$ where H is a random oracle and the arithmetic is in the underlying group.

2 Model

The model described in this section is based on that of [3, 4]. In particular we take from there the idea of modeling instances of principals via oracles available to the adversary; modeling various kinds of attacks by appropriate queries to these oracles; having some notion of partnering; and requiring semantic security of the session key via Test queries.

Protocol participants. We fix a nonempty set ID of *principals*. Each principal is either a *client* or a *server*: ID is the union of the finite, disjoint, nonempty sets $Client$ and $Server$. Each principal $U \in ID$ is named by a string, and that string has some fixed length. When $U \in ID$ appears in a protocol flow or as an argument to a function, we mean to the string which names the principal.

Long-lived keys. Each principal $A \in Client$ holds some password, pw_A. Each server $B \in Server$ holds a vector $pw_B = \langle pw_B[A] \rangle_{A \in Client}$ which contains an entry per client. Entry $pw_B[A]$ is called the *transformed-password*. In a protocol for the *symmetric* model $pw_A = pw_B[A]$; that is, the client and server share the same password. In a protocol for the *asymmetric* model, $pw_B[A]$ will typically be chosen so that it is hard to compute pw_A from A, B, and $pw_B[A]$. The password pw_A (and therefore the transformed password $pw_B[A]$) might be a poor one. Probably some human chose it himself, and then installed $pw_B[A]$ at the server. We call the pw_A and pw_B long-lived keys (LL-keys).

Figure 1 specifies how a protocol is run. It is in *Initialization* that pw_A and pw_B arise: everybody's LL-key is determined by running a *LL-key generator*, PW. A simple possibility for PW is that the password for client A is determined by $pw_A \stackrel{R}{\leftarrow} PW_A$, for some finite set PW_A, and $pw_B[A]$ is set to pw_A. Notice that, in Figure 1, PW takes a superscript h, which is chosen from space Ω. This lets PW's behavior depend on an idealized hash function. Different LL-key generators can be used to capture other settings, like a public-key one.

Initialization	$h \xleftarrow{R} \Omega$; $\langle \mathrm{pw}_A, \mathrm{pw}_B \rangle_{A \in Client,\ B \in Server} \xleftarrow{R} PW^h()$
	for $i \in \mathsf{N}$ and $U \in ID$ **do**
	  $state_U^i \leftarrow$ READY; $acc_U^i \leftarrow term_U^i \leftarrow used_U^i \leftarrow$ FALSE
	  $sid_U^i \leftarrow pid_U^i \leftarrow sk_U^i \leftarrow$ UNDEF

Send $(U,\ i,\ M)$	$used_U^i \leftarrow$ TRUE; **if** $term_U^i$ **then return** INVALID
	$\langle msg\text{-}out, acc, term_U^i, sid, pid, sk, state_U^i \rangle \leftarrow$
	  $P^h(\langle U, \mathrm{pw}_U, state_U^i, M \rangle)$
	if acc **and** $\neg acc_U^i$ **then**
	  $sid_U^i \leftarrow sid$; $pid_U^i \leftarrow pid$; $sk_U^i \leftarrow sk$; $acc_U^i \leftarrow$ TRUE
	return $\langle msg\text{-}out, sid, pid, acc, term_U^i \rangle$

Reveal $(U,\ i)$	**return** sk_U^i
Corrupt (U, pw)	**if** $U \in Client$ **and** $pw \neq$ DONTCHANGE **then** **for** $B \in Server$ **do** $\mathrm{pw}_B[U] = pw[B]$ **return** $\langle \mathrm{pw}_U,\ \{state_U^i\}_{i \in \mathsf{N}} \rangle$

Execute (A, i, B, j)	**if** $A \notin Client$ **or** $B \notin Server$ **or** $used_A^i$ **or** $used_B^j$
	  then return INVALID
	$msg\text{-}in \leftarrow B$
	for $t \leftarrow 1$ **to** ∞ **do**
	  $\langle msg\text{-}out, sid, pid, acc, term_A \rangle \xleftarrow{R}$ Send $(A,\ i,\ msg\text{-}in))$
	  $\alpha_t \leftarrow \langle msg\text{-}out, sid, pid, acc, term_A \rangle$
	  if $term_A$ **and** $term_B$ **then return** $\langle \alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_t \rangle$
	  $\langle msg\text{-}out, sid, pid, acc, term_B \rangle \xleftarrow{R}$ Send $(B,\ j,\ msg\text{-}in))$
	  $\beta_t \leftarrow \langle msg\text{-}out, sid, pid, acc, term_B \rangle$
	  if $term_A$ **and** $term_B$ **then return** $\langle \alpha_1, \beta_1, \alpha_2, \beta_2, \ldots, \alpha_t, \beta_t \rangle$

Test $(U,\ i)$	$sk \xleftarrow{R} SK$; $b \xleftarrow{R} \{0, 1\}$; **if** $\neg term_U^i$ **then return** INVALID **if** $b = 1$ **then return** sk_U^i **else return** sk
Oracle (M)	**return** $h(M)$

Fig. 1. The model. The protocol is P, the LL-key generator is PW, and the session-key space is SK. Probablity space Ω depends on the model of computation.

EXECUTING THE PROTOCOL. Formally, a protocol is just a probabilistic algorithm taking strings to strings. This algorithm determines how instances of the principals behave in response to signals (messages) from their enviornment. It is the adversary who sends these signals. As with the LL-key generator, P may depend on h.

Adversary $\mathcal{A}$ is a probabilistic algorithm with a distinguished query tape. Queries written on this tape are answered as specified in Figure 1. The following English-language description may clarify what is happening.

During the exeuction there may be running many *instances* of each principal $U \in ID$. We call instance i of principal U an *oracle*, and we denote it Π_U^i. Each instance of a principal might be embodied as a process (running on some machine) which is controlled by that principal.

A client-instance speaks first, producing some first message, *Flow1*. A server-instance responds with a message of its own, *Flow2*, intended for the client-instance which sent *Flow1*. This process is intended to continue for some fixed number of flows (usually 2–5), until both instances have *terminated*. By that time each instance should have *accepted*, holding a particular *session key* (SK), *session id* (SID), and *partner id* (PID). Let us describe these more fully.

At any point in time an oracle may *accept*. When an oracle accepts it holds a session key *sk*, a session id *sid*, and a partner id *pid*. Think of these values as having been written on a write-only tape. The SK is what the instance was aiming to get. It can be used to protect an ensuing conversation. The SID is an identifier which can be used to uniquely name the ensuing session. It is also useful definitionally. The PID names the principal with which the instance believes it has just exchanged a key. The SID and PID aren't secret—indeed we will hand them to the adversary—but the SK certainly is. A client-instance and a server-instance can accept at most once.

Remark 1. In this paper we use session IDs as our approach to defining partnering. This idea springs from discussions in 1995 among Bellare, Petrank, Rackoff, and Rogaway. In [3] the authors define partnering via "matching conversations," while in [4] the authors define partnering by way of an existentially guaranteed partnering function. Though all three approaches are reasonable, the use of matching-conversations can be criticized as focussing on a syntactic element that is ultimately irrelevant, while partnering via an existentially-guarateed partnering function allows for some unintuitive partnering functions. An explicit SID seems an elegant way to go. Specification document defining "real" protocols (eg., SSL and IPSec) typically do have SIDs, and in cases where an SID was not made explicit one can readilly define one (eg., by the concatenation of all protocol flows). □

Remark 2. We emphasize that accepting is different from terminating. When an instance terminates, it is done—it has what it wants, and won't send out any further messages. But an instance may wish to accept now, and terminate later. This typically happens when an instance believes it is now holding a good session key, but, prior to using that key, the instance wants confirmation that its desired communication partner really exists, and is also holding that same session key. The instance can accomplish this by accepting now, but waiting for a confirmation message to terminate. The distinction between terminating and accepting may at first seem artificial, but the distinction is convenient and it is typical of real MA protocols. It can be seen as an "asymmetry-breaking device" for dealing with the well-known issue that the party who sends the last flow is never sure if it was received. □

Our communications model places the adversary at the center of the universe. The adversary $\mathcal{A}$ can make queries to any instance: she has an endless supply of Π_U^i oracles ($U \in ID$ and $i \in \mathsf{N}$). There are all together six types of queries that $\mathcal{A}$ can make. The responses to these queries are specified in Figure 1. We now explain the capability that each kind of query captures.

(1) $\mathsf{Send}\,(U,\ i,\ M)$ — This sends message M to oracle Π_U^i. The oracle computes what the protocol says to, and sends back the response. Should the oracle accept, this fact, as well as the SID and PID, will be made visible to the adversary. Should the oracle terminate, this too will be made visible to the adversary. To initiate the protocol with client A trying to enter into an exchange with server B the adversary should send message $M = B$ to an unused instance of A. A Send-query models the real-world possibility of an adversary $\mathcal{A}$ causing an instance to come into existence, for that instance to receive communications fabricated by $\mathcal{A}$, and for that instance to respond in the manner prescribed by the protocol.

(2) $\mathsf{Reveal}\,(U,\ i)$ — If oracle Π_U^i has accepted, holding some session key sk, then this query returns sk to the adversary. This query models the idea (going back to Denning and Sacco [12]) that loss of a session key shouldn't be damaging to other sessions. A session key might be lost for a variety of reasons, including hacking, cryptanalysis, and the prescribed-release of that session key when the session is torn down.

(3) $\mathsf{Corrupt}\,(U, pw)$ — The adversary obtains pw_U and the states of all instances of U (but see Remark 3). This query models the possibility of subverting a principal by, for example, witnessing a user type in his password, installing a "Trojan horse" on his machine, or hacking into a machine. Obviously this is a very damaging type of query. Allowing it lets us deal with forward secrecy and the extent of damage which can be done by breaking into a server. A Corrupt query directed against a client U may also be used to replace the value of $pw_B[U]$ used by server B. This is the role of the second argument to Corrupt. Including this capability allows a dishonest client A to try to defeat protocol aims by installing a strange string as a server B's transformed password $pw_B[A]$.

(4) $\mathsf{Execute}\,(A, i, B, j)$ — Assuming that client oracle Π_A^i and server oracle Π_B^j have not been used, this call carries out an honest execution of the protocol between these oracles, returning a transcript of that execution. This query may at first seem useless since, using Send queries, the adversary already has the ability to carry out an honest execution between two oracles. Yet the query is essential for properly dealing with dictionary attacks. In modeling such attacks the adversary should be granted access to plenty of honest executions, since collecting these involves just passive eavesdropping. The adversary is comparatively constrained in its ability to actively manipulate flows to the principals, since bogus flows can be auditied and punative measures taken should there be too many.

(5) $\mathsf{Test}\,(U,\ i)$ — If Π_U^i has accepted, holding a session key sk, then the following happens. A coin b is flipped. If it lands $b = 0$, then sk is returned to the adversary. If it lands $b = 1$, then a random session key, drawn from the distribution from which session keys are supposed to be drawn, is returned. This type of query is only used to measure adversarial success—it does not correspond to any actual adversarial ability. You should think of the adversary asking this query just once.

(6) $\mathsf{Oracle}\,(M)$ — Finally, we give the adversary oracle access to a function h, which is selected at random from some probability space Ω. As already remarked,

not only the adversary, but the protocol and the LL-key generator may depend on h. The choice of Ω determines if we are woking in the *standard* model, *ideal-hash* model, or *ideal-cipher* model. See the discussion below.

Remark 3. As described in Figure 1, a Corrupt query directed against U releases the LL-key pw_U and also the current state of all instances of U. We call this the "strong-corruption model." A weaker type of Corrupt query returns *only* the LL-key of that principal. We call this the "weak-corruption model." The weak-corruption model corresponds to acquiring a principal's password by coaxing it out of him, as opposed to completely compromising his machine. $\square$

Remark 4. Notice that a Corrupt query to U does not result in the release of the session keys owned by U. The adversary already has the ability to obtain session keys through Reveal queries, and releasing those keys by a Corrupt query would make forward secrecy impossible. $\square$

Remark 5. Soon after the appearance of [4], Rackoff [19] came up with an example showing how the definition given in that paper was not strong enough to guarantee security for certain applications using the distributed session key. The authors of [4] traced the problem to a simple issue: they had wrongly made the restriction that the Test query be the adversary's last. Removal of this restriction solved the problem. This minor but important change in the definition of [4], made in 1995, has since been folklore in the community of researchers in this area, and is explicitly incorporated into our current work. $\square$

STANDARD MODEL, IDEAL-HASH MODEL, IDEAL-CIPHER MODEL. Figure 1 refers to probability space Ω. We consider three possiblities for Ω, giving rise to three different models of computation.

In the **standard model** Ω is the distribution which puts all the probability mass on one function: the constant function which returns the empty-string, ε, for any query M. So in the standard model, all mention of h can be ignored.

Fix a finite set of strings $\mathcal{C}$. In the **ideal-hash model** (also called the random-oracle model) choosing a random function from Ω means choosing a random function h from $\{0,1\}^*$ to $\mathcal{C}$. This models the use of a cryptographic hash function which is so good that, for purposes of analysis, one prefers to think of it as a public random function.

Fix finite sets of strings $\mathcal{G}$ and $\mathcal{C}$ where $|\mathcal{G}| = |\mathcal{C}|$. In the **ideal-cipher model** choosing a random function h from Ω amounts to giving the protocol (and the adversary) a perfect way to encipher strings in $\mathcal{G}$: namely, for $K \in \{0,1\}^*$, we set $\mathcal{E}_K \colon \mathcal{G} \to \mathcal{C}$ to be a random bijective function, and we let $\mathcal{E}_K \colon \{0,1\}^* \to \mathcal{G}$ be defined by $\mathcal{D}_K(y)$ is the value x such that $\mathcal{E}_K(x) = y$, if $y \in \mathcal{C}$, and BAD otherwise. We let $h(\mathsf{encrypt}, K, M) = \mathcal{E}_K(M)$ and $h(\mathsf{decrypt}, K, C) = \mathcal{D}_K(C)$. The capabilities of the ideal-hash model further include those of the ideal-cipher model, by means of a query $h(\mathsf{hash}, x)$ which, for shorthand, we denote $H(x)$.

The ideal-cipher model is very strong (even stronger than the ideal-hash model) and yet there are natural and apparently-good ways to instantiate an

The *basic* notion of freshess (no requirement for forward secrecy): **if** [RevealTo (U, i)] **or** [RevealToPartnerOf (U, i)] **or** [SomebodyWasCorrupted] **then** unfresh **else** fresh
A notion of freshness the incorporates a requirement for *forward secrecy*: **if** [RevealTo (U, i)] **or** [RevealToPartnerOf (U, i)] **or** [SomebodyWasCorruptedBeforeTheTestQuery **and** Manipulated(U, i)] **then** fs-unfresh **else** fs-fresh

Fig. 2. Session-key freshness. A Test query is made to oracle Π_U^i. The chart specifies how, at the end of the execution, the session key of that oracle should be regarded (fresh or unfresh, and fs-fresh or fs-unfresh). Notation is described in the accompanying text.

ideal cipher for use in practical protocols. See [8]. Working in this model does not render trivial the goals that this paper is interested in, and it helps make for protocols that don't waste any bits. A protocol will always have a clearly-indicated model of computation for which it is intended so, when the protocol is fixed, we do not make explicit mention of the model of compuation.

Remark 6. The ideal-cipher model is richer than the RO-model, and you can't just say "apply the Feistel construction to your random oracle to make the cipher." While this may be an approach to instantiating an ideal-cipher, there is no formal sense we know in which you can simulate the ideal-cipher model using only the RO-model. $\square$

3 Definitions

Our definitional approach is from [4], but adaptations must be made since partnering is defined in a different manner than in [4] (as discussed in Section 2), and since we now consider forward secrecy as one of our goals.

PARTNERING USING SIDS. Fix a protocol P, adversary $\mathcal{A}$, LL-key generator PW, and session-key space SK. Run P in the manner specified in Section 2. In this execution, we say that oracles Π_U^i and $\Pi_{U'}^{i'}$ are **partnered** (and each oracle is said to be a partner of the other) if both oracles accept, holding (sk, sid, pid) and (sk', sid', pid') respectively, and the following hold:

(1) $sid = sid'$ and $sk = sk'$ and $pid = U'$ and $pid' = U$.

(2) $U \in Client$ and $U' \in Server$, or $U \in Server$ and $U' \in Client$.

(3) No oracle besides Π_U^i and $\Pi_{U'}^{i'}$ accepts with a PID of pid.

The above definition of partnering is quite strict. For two oracles to be partners with one another they should have the same SID and the same SK, one should be a client and the other a server, each should think itself partnered with the other, and, finally, no third oracle should have the same SID. Thus an oracle that has accepted will have a single partner, if it has any partner at all.

TWO FLAVORS OF FRESHNESS. Once again, run a protocol with its adversary. Suppose that the adversary made exactly one Test query, and it was to Π_U^i.

Intuitively, the oracle Π_U^i should be considered unfresh if the adversary may know the SK contained within it.

In Figure 2 we define two notions of freshness—with and witout forward secrecy (fs). Here is the notation used in that figure. We say "RevealTo $(U,\ i)$" is true iff there was, at some point in time, a query $\mathsf{Reveal}\,(U,\ i)$. We say "RevealToPartnerOf $(U,\ i)$" is true iff there was, at some point in time, a query $\mathsf{Reveal}\,(U',\ i')$ and $\Pi_{U'}^{i'}$ is a partner to Π_U^i. We say "SomebodyWasCorrupted" is true iff there was, at some point in time, a query $\mathsf{Corrupt}\,(U', pw)$ for some U', pw. We say "SomebodyWasCorruptedBeforeTheTestQuery" is true iff there was a $\mathsf{Corrupt}\,(U', pw)$ query and this query was made before the $\mathsf{Test}\,(U,\ i)$ query. We say that "Manipulated(U, i)" is true iff there was, at some point in time, a $\mathsf{Send}\,(U,\ i,\ M)$ query, for some string M.

Explanation. In our definition of security we will be "giving credit" to the adversary $\mathcal{A}$ if she specifies a fresh (or fs-fresh) oracle and then correctly identifies if she is provided the SK from that oracle or else a random SK. We make two cases, according to whether or not "forward secrecy" is expected. Recall that forward secrecy entails that loss of a long-lived key should not compromise already-distributed session keys.

Certainly an adversary can know the SK contained within an oracle Π_U^i if she did a Reveal query to Π_U^i, or if she did a Reveal query to a partner of Π_U^i. This accounts for the first two disjuncts in each condition of Figure 2. The question is whether or not a Corrupt query may divulges the SK. Remember that a Corrupt query does not return the SK, but it does return the LL-key. For the "basic" notion of security (fresh/unfresh) we pessimistically assume that a Corrupt query does reveal the session key, so any Corrupt query makes all oracles unfresh. (One could tighten this a little, if desired.) For the version of the definition with forward secrecy a Corrupt query may reveal a SK only if the Corrupt query was made *before* the Test query. We also require that the Test query was to an oracle that was the target of a Send query (as opposed to an oracle that was used in an Execute query). (Again, this can be tightened up a little.) This acts to build in the following requirement: that even *after* the Corrupt query, session keys exchanged by principals who behave honestly are *still* fs-fresh. This is a nice property, and since it seems to always be achieved in protocols which achieve forward secrecy, we have lumped it into that notion. It amounts to saying that an "honest" oracle—one that is used only for an Execute call—is always fs-fresh, even if there is a Corrupt query. (Of course you still have to exclude the the possiblity that the oracle was the target of a Reveal query, or that its partner was.)

Remark 7. Forward secrecy, in the strong-corruption model, is not achievable by two-flow protocols. The difficulty is the following. A two-flow protocol is client-to-server then server-to-client. If the client oracle is corrupted after the server oracle has terminated but before the client oracle has received the response, then the server oracle will be fs-fresh but the adversary can necessarilly compute the shared SK since the adversary has the exact same information that the client oracle would have had the client oracle received the server oracle's flow.

One way around this is to go to the weak-corruption model. A second way around this is to add a third flow to the protocol. A final way around this is to define a slightly weaker notion of forward secrecy, *weak* forward-secrecy, in which an oracle is regarded as "wfs-unfresh" if it fs-unfresh, or the test query is to a manipulated oracle, that oracle is unpartnered at termination, and somebody gets corrupted. Otherwise the oracle is wfs-fresh. □

AKE SECURITY (WITH AND WITHOUT FORWARD SECRECY). In a protocol execution of $P, PW, SK, \mathcal{A}$ we say that $\mathcal{A}$ *wins*, in the AKE sense, if she asks a single Test-query, $\mathsf{Test}\,(U,\ i)$, where Π_U^i has terminated and is fresh, and $\mathcal{A}$ outputs a single bit, b', and $b' = b$ (where b is the bit selected during the Test query). The ake advantage of $\mathcal{A}$ in attacking (P, PW, SK) is twice the probability that $\mathcal{A}$ wins, minus one. (The adversay can trivially win with probability $1/2$. Multiplying by two and subtracting one simply rescales this probability.) We denote the **ake advantage** by $\mathsf{Adv}^{\mathrm{ake}}_{P,PW,SK}(\mathcal{A})$.

We similarly define the **ake-fs advantage**, $\mathsf{Adv}^{\mathrm{ake\text{-}fs}}_{P,PW,SK}(\mathcal{A})$, where now one insists that the oracle Π_U^i to which the Test-query is directed be fs-fresh.

AUTHENTICATION. In a protocol execution of $P, PW, SK, \mathcal{A}$, we say that an adversary violates client-to-server authentication if some server oracle terminates but has no partner oracle. We let the **c2s advantage** be the probablity of this event, and denote it by $\mathsf{Adv}^{\mathrm{c2s}}_{P,PW,SK}(\mathcal{A})$. We say that an adversary violates server-to-client authentication if some client oracle terminates but has no partner oracle. We let the **s2c advantage** be the probability of this event, and denote it by $\mathsf{Adv}^{\mathrm{s2c}}_{P,PW,SK}(\mathcal{A})$. We say that an adversary violates mutual authentication if some oracle terminates, but has no partner oracle. We let the **ma advantage** denote the probablity of this event, and denote it by $\mathsf{Adv}^{\mathrm{ma}}_{P,PW,SK}(\mathcal{A})$.

MEASURING ADVERSARIAL RESOURCES. We are interested in an adversary's maximal advantage in attacking some protocol as a function of her resources. The resources of interest are:

- t — the adversary's running time. By convention, this includes the amount of space it takes to describe the adversary.
- q_{se}, q_{re}, q_{co}, q_{ex}, q_{or} — these count the number of Send, Reveal, $\mathsf{Corrupt}$, $\mathsf{Execute}$, and Oracle queries, respectively.

When we write $\mathsf{Adv}^{\mathrm{ake}}_{P,PW,SK}(\mathit{resources})$, overloading the Adv-notation, it means the maximal possible value of $\mathsf{Adv}^{\mathrm{ake}}_{P,PW,SK}(\mathcal{A})$ among all adversaries that expend at most the specified resources. By convention, the time to sample in PW (one time) and to sample in SK (one time) are included in $\mathsf{Adv}_{P,PW,SK}(\mathit{resources})$ (for each type of advantage).

DIFFIE-HELLMAN ASSUMPTION. We will prove security under the computational Diffie-Hellman assumption. The concrete version of relevance to us is the following. Let $G = \langle g \rangle$ be a finite group. We assume some fixed representation for group elements, and implicitly switch between group elements and their string representations. Let $\mathcal{A}$ be an adversary that outputs a list of group elements,

$$\begin{array}{ccc} A^{\,pw} & & B^{\,pw} \\ x \stackrel{R}{\leftarrow} \{1,\ldots,|\mathcal{G}|\} & \xrightarrow{\quad A \;\|\; \mathcal{E}_{pw}\,(g^x) \quad} & \\ & \xleftarrow{\quad \mathcal{E}_{pw}\,(g^y) \quad} & y \stackrel{R}{\leftarrow} \{1,\ldots,|\mathcal{G}|\} \end{array}$$

Fig. 3. The protocol EKE2. Depicted are flows of an honest execution. The shared session key is $sk = H(A \,\|\, B \,\|\, g^x \,\|\, g^y \,\|\, g^{xy})$ and shared session ID is $sid = A \,\|\, \mathcal{E}_{pw}\,(g^x) \,\|\, B \,\|\, \mathcal{E}_{pw}\,(g^y)$. The partner ID for A is $pid_A = B$ and the partner ID for B is $pid_B = A$.

$z_1, \ldots, z_q$. Then we define

$$\mathsf{Adv}^{\mathrm{dh}}_{G}(\mathcal{A}) = \Pr[x, y \leftarrow \{1, \ldots, |G|\} :\; g^{xy} \in \mathcal{A}(g^x, g^y)], \text{ and}$$
$$\mathsf{Adv}^{\mathrm{dh}}_{G}(t, q) = \max_{\mathcal{A}}\{\; \mathsf{Adv}^{\mathrm{dh}}_{G,g}(\mathcal{A}) \;\}\;,$$

where the maximum is over all adversaries that run in time at most t and output a list of q group elements. As before, t includes the description size of adversary $\mathcal{A}$.

4 Secure AKE: Protocol EKE2

In this section we prove the security of the two flows at the center of Bellovin and Merritt's EKE protocol [6]. Here we define the (slightly modified) "piece" of EKE that we are interested in.

Description of EKE2. This is a Diffie-Hellman key exchange in which each flow is enciphered by the password, the SK is $sk = H(A \,\|\, B \,\|\, g^x \,\|\, g^y \,\|\, g^{xy})$, and the SID and PID are appropriately defined. The name of the sender also accompanies the first flow. See Figures 3 and 4.

Arithmetic is in a finite cyclic group $\mathcal{G} = \langle g \rangle$. This group could be $\mathcal{G} = \mathbb{Z}_p^{\star}$, or it could be a prime-order subgroup of this group, or it could be an elliptic curve group. We denote the group operation multiplicatively. The protocol uses a cipher $\mathcal{E} : \; \mathit{Password} \times \mathcal{G} \to \mathcal{C}$, where $pw_A \in \mathit{Password}$ for all $A \in \mathit{Client}$. There are many concrete constructions that could be used to instantiate such an object; see [8]. In the analysis this is treated as an ideal cipher. Besides the cipher we use a hash function H. It outputs ℓ-bits, where ℓ is the length of the session key we are trying to distribute. Accordingly, the session-key space SK associated to this protocol is $\{0,1\}^{\ell}$ equipped with a uniform distribution.

Security theorem. The following indicates that the security of EKE2 is about as good as one could hope for. We consider the simple case where *Password* has size N and all client passwords are chosen uniformly (and independently) at random from this space. Formally this initialization is captured by defining the appropriate LL-key generator PW. It picks $pw_A \stackrel{R}{\leftarrow} \mathit{Password}$ for each $A \in \mathit{Client}$

if *state* = READY **and** $U \in \mathit{Client}$ **then** // A `sends the first flow`
$\langle A\rangle \leftarrow U \quad \langle B\rangle \leftarrow \mathit{msg\text{-}in}$, where $B \in \mathit{Server}$
$x \xleftarrow{R} \{1,\ldots,|G|\} \quad X \leftarrow g^x \quad X^* \leftarrow \mathcal{E}_{pw}(X) \quad \mathit{msg\text{-}out} \leftarrow A \parallel X^*$
$\mathit{sid} \leftarrow \mathit{pid} \leftarrow \mathit{sk} \leftarrow \varepsilon \quad \mathit{acc} \leftarrow \mathit{term} \leftarrow \text{FALSE} \quad \mathit{state} \leftarrow \langle x, B\rangle$
return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$
else if *state* = READY **and** $U \in \mathit{Server}$ **then** // B `sends the second flow`
$\langle B\rangle \leftarrow U \quad \langle A, X^*\rangle \leftarrow \mathit{msg\text{-}in}$, where $A \in \mathit{Client}$ and X^* is a ciphertext
$y \xleftarrow{R} \{1,\ldots,|G|\} \quad Y \leftarrow g^y \quad Y^* \leftarrow \mathcal{E}_{pw}(Y)$
$X \leftarrow \mathcal{D}_{pw}(X^*) \quad K \leftarrow X^y \quad \mathit{msg\text{-}out} \leftarrow Y^*$
$\mathit{sid} \leftarrow A \parallel X^* \parallel B \parallel Y^* \quad \mathit{pid} \leftarrow A \quad \mathit{sk} \leftarrow H(A \parallel B \parallel X \parallel Y \parallel K)$
$\mathit{acc} \leftarrow \mathit{term} \leftarrow \text{TRUE} \quad \mathit{state} \leftarrow \text{DONE}$
return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$
else if $\mathit{state} = \langle x, B\rangle$ **and** $U \in \mathit{Client}$ **then** // A `receives the second flow`
$\langle Y^*\rangle \leftarrow \mathit{msg\text{-}in}$, where Y^* is a ciphertext
$Y \leftarrow \mathcal{D}_{pw}(Y^*) \quad K \leftarrow Y^x$
$\mathit{sid} \leftarrow A \parallel X^* \parallel B \parallel Y^* \quad \mathit{pid} \leftarrow B \quad \mathit{sk} \leftarrow H(A \parallel B \parallel X \parallel Y \parallel K)$
$\mathit{acc} \leftarrow \mathit{term} \leftarrow \text{TRUE} \quad \mathit{state} \leftarrow \text{DONE}$
return $(\mathit{msg\text{-}out}, \mathit{acc}, \mathit{term}, \mathit{sid}, \mathit{pid}, \mathit{sk}, \mathit{state})$

Fig. 4. Definition of EKE2. The above defines both client and server behavior, $P^h(\langle U, pw, \mathit{state}, \mathit{msg\text{-}in}\rangle)$.

and sets $pw_B[A] = pw_A$ for each $B \in \mathit{Server}$ and $A \in \mathit{Client}$. It then sets $pw_B = \langle pw_B[A]\rangle_{A\in \mathit{Client}}$ and outputs $\langle pw_A, pw_B\rangle_{A\in\mathit{Client},\, B\in\mathit{Server}}$. The theorem below assumes that the space *Password* is known in the sense that it is possible to sample from it efficiently.

Theorem 1. *Let $q_{\rm se}, q_{\rm re}, q_{\rm co}, q_{\rm ex}, q_{\rm or}$ be integers and let $q = q_{\rm se} + q_{\rm re} + q_{\rm co} + q_{\rm ex} + q_{\rm or}$. Let Password be a finite set of size N and assume $1 \le N \le \sqrt{|\mathcal{G}|}/q$. Let PW be the associated LL-key generator as discussed above. Let P be the EKE2 protocol and let SK be the associated session-key space. Assume the weak-corruption model. Then*

$$\begin{aligned}&\mathsf{Adv}^{\text{ake-fs}}_{P,PW,SK}(t, q_{\rm se}, q_{\rm re}, q_{\rm co}, q_{\rm ex}, q_{\rm or})\\ &\le \frac{q_{\rm se}}{N} + q_{\rm se}\cdot q_{\rm or}\cdot \mathsf{Adv}^{\rm dh}_{\mathcal{G},g}(t', q_{\rm or}) + \frac{O(q^2)}{|\mathcal{G}|} + \frac{O(1)}{\sqrt{|\mathcal{G}|}}\end{aligned}$$

where $t' = t + O(q_{\rm se} + q_{\rm or})$. □

Remark 8. Since EKE2 is a two-flow protocol, Remark 7 implies that it cannot achieve forward secrecy in the strong-corruption model. Accordingly the above theorem considers the weak-corruption model with regard to forward secrecy. The resistance to dictionary attacks is captured by the first term which is the number of send queries divided by the size of the password space. The other terms can be made negligible by an appropriate choice of parameters for the group $\mathcal{G}$. □

Remark 9. The upper bound imposed in the theorem on the size N of the password space is not a restriction because if the password space were larger the question of dictionary attacks becomes moot: the adversary cannot exhaust the password space off-line anyway. Nonetheless it may be unclear why we require such a restriction. Intuitively, as long as the password space is not too large the adversary can probably eliminate at most one candidate password from consideration per Send query, but for a larger password space it might in principle be able to eliminate more at a time. This doesn't damage the success probability because although it eliminates more passwords at a time, there are also more passwords to consider. □

The proof of Theorem 1 is omitted due to lack of space and can be found in the full version of this paper [2]. We try however to provide a brief sketch of the main ideas in the analysis.

Assume for simplicity there is just one client A and one server B. Consider some adversary $\mathcal{A}$ attacking the protocol. We view $\mathcal{A}$ as trying to guess A's password. We consider at any point in time a set of "remaining candidates." At first this equals *Password*, and as time goes on it contains those candidate passwords that the adversary has not been able to eliminate from consideration as values of the actual password held by A. We also define a certain "bad" event in the execution of the protocol with this adversary, and show that as long as this event does not occur, two things are true:

(1) A's password, from the adversary's point of view, is equally likely to be any one from the set of remaining passwords, and

(2) The size of the set of remaining passwords decreases by at most one with each oracle query, and the only queries for which a decrease occurs are reveal or test queries to manipulated oracles.

The second condition implies that the number of queries for which the decrease of size in the set of remaining candidates occurs is bounded by q_{se}. We then show that the probability of the bad event can be bounded in terms of the advantage function of the DH problem over $\mathcal{G}$.

Making this work requires isolating a bad event with two properties. First, whenever it happens we have a way to "embed" instances of the DH problem into the protocol so that adversarial success leads to our obtaining a solution to the DH problem. Second, absence of the bad event leads to an inability of the adversary to obtain information about the password at a better rate than eliminating one password per reveal or test query to a manipulated oracle. Bounding the probability of the bad event involves a "simulation" argument as we attempt to "plant" DH problem instances in the protocol. Bounding adversarial success under the assumption the bad event does not happen is an information-theoretic argument. Indeed, the difficulty of the proof is in choosing the bad event so that one can split the analysis into an information-theoretic component and a computational component in this way.

5 Adding Authentication

In this section we sketch generic transformations for turning an AKE protocol P' into a protocol P that provides client-to-server authentication, server-to-client authentication, or both. The basic approach is well-known in folklore—use the distributed session key to construct a simple "authenticator" for the other party—but one has to be careful in the details, and people often get them wrong.

The ease with which an AKE protocol can be modified to provide authentication is one of the reasons for using AKE as a starting point.

In what follows we assume that the AKE protocol P' is designed to distribute session keys from a space $SK = U_\ell$, the uniform distribution on ℓ-bit strings.

While a pseudorandom function is sufficient for adding authentication to an AKE protocol, for simplicity (and since one likely assumes it anyway, in any practical password-based AKE construction) we assume (at least) the random-oracle model. The random hash function is denoted H. Its argument (in our construction) will look like $sk' \parallel i$, where sk' is an ℓ-bit string and i is a fixed-length string encoding one of the numbers 0, 1, or 2. We require that the AKE protocol P never evaluates H at any point of the form $sk' \parallel 0$, $sk' \parallel 1$, or $sk' \parallel 2$, where $sk' \in \{0,1\}^\ell$.

The transformations. The transformation AddCSA (add client-to-server authentication) works as follows. Suppose that in protocol P' the client A has accepted sk'_A, sid'_A, pid'_A, and suppose that A then terminates. In protocol $P = \mathrm{AddCSA}(P')$ have A send one additional flow, $auth_A = H(sk'_A \parallel 2)$, have A accept $sk_A = H(sk'_A \parallel 0)$, $sid_A = sid'_A$, $pid_A = pid'_A$, and have A terminate, saving no state. On the server side, suppose that in P' the server B accepts sk'_B, sid'_B, pid'_B, and B terminates. In protocol P have B receive one more flow, $auth'_A$. Have B check if $auth'_A = H(sk'_B \parallel 2)$. If so, then B accepts $sk_B = H(sk'_B \parallel 0)$, $sid_B = sid'_B$, $pid_B = pid'_B$, and then B terminates, without saving any state. Otherwise, B terminates (rejecting), saving no state.

Transformations AddSCA (add server-to-client authentication) and AddMA (add mutual authentication) are analogous. The latter is illustrated in Figure 5. In all of these transformation, when a party ends up sending two consecutive flows, one can always collapse them into one.

Remark 10. It is crucial in these transformations that the SK produced by P' is not used both to produce an authenticator and as the final session key; if one does this, the protocol is easily seen to be insecure under our definitions. This is a common "error" in the design of authentication protocols. It was first discussed [3]. □

Properties. Several theorems can be pursued about how the security of P' relates to that of AddCSA(P'), AddSCA(P'), and AddMA(P'). These capture the following. If P' is good in the sense of $\mathsf{Adv}^{\mathrm{ake}}$ then AddCSA(P') is good in the sense of $\mathsf{Adv}^{\mathrm{ake}}$ and $\mathsf{Adv}^{\mathrm{c2s}}$. If P' is good in the sense of $\mathsf{Adv}^{\mathrm{ake}}$ then AddSCA(P') is good in the sense of $\mathsf{Adv}^{\mathrm{ake}}$ and $\mathsf{Adv}^{\mathrm{s2c}}$. If P' is good in the sense of $\mathsf{Adv}^{\mathrm{ake}}$ then AddMA(P') is good in the sense of $\mathsf{Adv}^{\mathrm{ake}}$, $\mathsf{Adv}^{\mathrm{s2c}}$, and $\mathsf{Adv}^{\mathrm{c2s}}$. The weak form

$$
\begin{array}{lcl}
A^{pw} & & B^{pw} \\[1ex]
x \stackrel{R}{\leftarrow} \{1,\ldots,|\mathcal{G}|\} & & \\
 & \xrightarrow{\quad A \,\|\, \mathcal{E}_{pw}(g^x) \quad} & \\
 & & y \stackrel{R}{\leftarrow} \{1,\ldots,|\mathcal{G}|\} \\
 & & sk' \leftarrow H(A\|B\|g^x\|g^y\|g^{xy}) \\
 & \xleftarrow{\quad \mathcal{E}_{pw}(g^y) \,\|\, H(sk'\|1) \quad} & \\
sk' \leftarrow H(A\|B\|g^x\|g^y\|g^{xy}) & & \\
 & \xrightarrow{\quad H(sk'\|2) \quad} &
\end{array}
$$

Fig. 5. Flows of an honest execution of AddMA(EKE2). The shared SK is $sk = H(sk' \,\|\, 0)$ and the shared SID is $sid = A \,\|\, \mathcal{E}_{pw}(g^x) \,\|\, B \,\|\, \mathcal{E}_{pw}(g^y)$. The PID for A is B and the PID for B is A.

of forward secrecy mentioned in Remark 7 is also interesting in connection with AddCSA and AddMA, since these transformations apparently "upgrade" good weak forward secrecy, $\mathsf{Adv}^{\text{ake-wfs}}$, to good ordinary forward secrecy, $\mathsf{Adv}^{\text{ake-fs}}$.

Simplifications. The generic transformations given by AddCSA, AddSCA and AddMA do not always give rise to the most efficient method for the final goal. Consider the protocol AddMA(EKE2) of Figure 5. It would seem that the encryption in the second flow can be eliminated and one still has a good protocol for AKE with MA. However, we know of no approach towards showing such a protocol secure short of taking the first two flows of that protocol and showing that they comprise a good AKE protocol with server-to-client authentication, and then applying AddCSA transformation.

Given the complexity of proofs in this domain and the tremendous variety of simple and plausibly correct protocol variants, it is a major open problem in this area to find techniques which will let us deal with the myriad of possibilities, proving the correct ones correct, without necessitating an investment of months of effort to construct a "rigid" proof for each and every possibility.

Acknowledgments

We thank Charlie Rackoff for extensive discussions on the subject of session-key exchange over the last five years, and for his corrections to our earlier works. We thank Victor Shoup for useful comments and criticisms on this subject. We thank the Eurocrypt 2000 committee for their excellent feedback and suggestions.

Mihir Bellare is supported in part by NSF CAREER Award CCR-9624439 and a 1996 Packard Foundation Fellowship in Science and Engineering. Phillip Rogaway is supported in part by NSF CAREER Award CCR-9624560. Much of Phil's work on this paper was carried out while on sabbatical in the Dept. of Computer Science, Faculty of Science, Chiang Mai University, Thailand.

References

1. M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. *Proc. of the 30th STOC.* ACM Press, New York, 1998.
2. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. Full version of this paper, available from `http://www-cse.ucsd.edu/users/mihir`
3. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *CRYPTO '93*, LNCS 773, pages 232–249. Springer-Verlag, Berlin, 1994.
4. M. Bellare and P. Rogaway. Provably Secure Session Key Distribution: the Three Party Case. *Proc. of the 27th STOC.* ACM Press, New York, 1995.
5. M. Bellare and P. Rogaway, work in progress.
6. S. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure against Dictionary Attacks. *Proc. of the Symposium on Security and Privacy*, pages 72–84. IEEE, 1992.
7. S. Bellovin and M. Merritt. Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise. *Proceedings of the 1st Annual Conference on Computer and Communications Security*, ACM, 1993.
8. J. Black and P. Rogaway. Ciphers with Arbitrary Finite Domains. Manuscript, 2000.
9. M. Boyarsky. Public-Key Cryptography and Password Protocols: The Multi-User Case. *Proceedings of the 7th Annual Conference on Computer and Communications Security*, ACM, 1999.
10. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password Authenticated Key Exchange Using Diffie Hellman. Eurocrypt 2000.
11. P. Buhler, T. Eirich, M. Steiner, and M. Waidner. Secure Password-Based Cipher Suite for TLS. Proceedings of Network and Distributed Systems Security Symposium. February 2000.
12. D. Denning and G. Sacco. Timestamps in Key Distribution Protocols. *Communications of the ACM*, 24, 1981, pp 533–536.
13. L. Gong, M. Lomas, R. Needham, and J. Saltzer. Protecting Poorly Chosen Secrets from Guessing Attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, June 1993.
14. S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. *ACM Transactions on Information and System Security*, Vol. 2, No. 3, pp. 230–268, August 1999. Earlier version in *Proc. of the 5th CCS* conference, ACM Press, New York, 1998.
15. D. Jablon. Strong Password-Only Authenticated Key Exchange. *ACM Computer Communications Review*, October 1996.
16. D. Jablon. Extended Password Key Exchange Protocols Immune to Dictionary Attacks. *Proc. of WET-ICE '97*, pp. 248–255. IEEE Computer Society, June 1997.
17. S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. *Proc. of the Security Protocols Workshop*, LNCS 1361. Springer-Verlag, Berlin, 1997.
18. P. MacKenzie and R. Swaminathan. Secure Authentication with a Short Secret. Manuscript. November 2, 1999. Earlier version as Secure Network Authentication with Password Identification. Submission to IEEE P1363a. August 1999. Available from `http://grouper.ieee.org/groups/1363/addendum.html`

19. C. Rackoff, private communication, 1995.
20. V. Shoup. On Formal Models for Secure Key Exchange. Theory of Cryptography Library Record 99-12, `http://philby.ucsd.edu/cryptolib/` and invited talk at ACM Computer and Communications Security conference, 1999.
21. M. Roe, B. Christianson, and D.Wheeler. Secure Sessions from Weak Secrets. Technical report from University of Cambridge and University of Hertfordshire. Manuscript, 1998.
22. T. Wu. The Secure Remote Password Protocol. *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, pp. 97–111, 1998.