Entity Authentication and Key Distribution

Mihir Bellare[1] and Phillip Rogaway[2]

[1] High Performance Computing and Communications, IBM T.J. Watson Research Center, PO Box 704, Yorktown Heights, NY 10598, USA. e-mail: mihir@watson.ibm.com.

[2] PS LAN System Design, IBM Personal Software Products, 11400 Burnet Road, Austin, TX 78758, USA. e-mail: rogaway@austin.ibm.com

Abstract. We provide the first formal treatment of entity authentication and authenticated key distribution appropriate to the distributed environment. Addressed in detail are the problems of mutual authentication and authenticated key exchange for the symmetric, two-party setting. For each we present a definition, protocol, and proof that the protocol meets its goal, assuming only the existence of a pseudorandom function.

1 Introduction

Entity authentication is the process by which an agent gains confidence in the identity of a communication partner. Though central to computing practice, entity authentication for the distributed environment rests on no satisfactory formal foundations.This is more than an academic complaint; entity authentication is an area in which an informal approach has often lead to work which is at worst wrong, and at best only partially analyzable. In particular, an alarming fraction of proposed protocols have subsequently been found to be flawed (see, e.g., [5, 3]) and the bugs have, in some cases, taken years to discover. It is therefore desirable that confidence in an authentication protocol should stem from more than a few people's inability to break it. In fact, each significant entity authentication goal should be formally defined and any candidate protocol should be proven to meet its goal under a standard cryptographic assumption.

More often than not the entity authentication process is coupled with the distribution of a "session key" which the communicating partners may later use for message confidentiality, integrity, or whatever else. This "authenticated key distribution" goal may be considered even more important in practice than the pure entity authentication goal. As a problem, it is beset with the same foundational difficulties as the entity authentication problem of which it is an extension.

Authentication and authenticated key distribution problems come in many different flavors: there may be two parties involved, or more; the authentication may be unilateral or mutual; parties might (the symmetric case) or might not (the asymmetric case) share a secret key. Here we focus on two version of the the two-party, mutual, symmetric case. In the *mutual authentication* problem the

D.R. Stinson (Ed.): Advances in Cryptology - CRYPTO '93, LNCS 773, pp. 232-249, 1994.

parties, representing processes in a distributed system, engage in a conversation in which each gains confidence that it is the other with whom he speaks. In the *authenticated key exchange* problem the parties also want to distribute a "fresh" and "secret" *session key*.[3]

1.1 Contributions of this Paper

A COMMUNICATION MODEL FOR DISTRIBUTED SECURITY. It has been pointed out in many places that one difficulty in laying foundations for entity authentication and authenticated key distribution protocols has been the lack of a formal communications model for authentication in the distributed environment. Here we specify such a model. To be fully general, we assume that all communication among interacting parties is under the adversary's control. She can read the messages produced by the parties, provide messages of her own to them, modify messages before they reach their destination, and delay messages or replay them. Most importantly, the adversary can start up entirely new "instances" of any of the parties, modeling the ability of communicating agents to simultaneously engage in many *sessions* at once. This gives us the ability to model the kinds of attacks that were suggested by [3]. Formally, each party will be modeled by an infinite collection of oracles which the adversary may run. These oracles only interact with the adversary, they never directly interact with one another. See Section 3.

DEFINITIONS. In the presence of an adversary as powerful as the one we define, it is unclear what it could possibly mean to be convinced that one has engaged in a conversation with a specified partner; after all, every bit communicated has really been communicated to the the adversary, instead. We deal with this problem as follows.

As has often been observed, an adversary in our setting can always make the parties accept by faithfully relaying messages among the communication partners. But this behavior does not constitute a damaging attack; indeed, the adversary has functioned just like a wire, and may as well not have been there. The idea of our definition of a mutual authentication is simple but strong: we formalize that a protocol is secure if the *only* way that an adversary can get a party to accept is by faithfully relaying messages in this manner. In other words, any adversary effectively behaves as a trusted wire, if not a broken one.

To define authenticated key exchange it is necessary to capture a protocol's robustness against the loss of a session key; even if the adversary gets hold of one, this isn't supposed to compromise security beyond the particular session which that key protects. We model this requirement by allowing the adversary to obtain session keys just by asking for them. When this inquiry is made, the

[3] At first glance it might seem unnecessary for two parties who already share a key a to come up with another key α. One reason a new key is useful is the necessity of avoiding cross-session "replay attacks" —messages copied from one session being deemed authentic in another— coupled with an insistence on *not* attempting to carry "state" information (e.g., a message counter) across distinct sessions.

key is no longer *fresh*, and the partner's key is declared unfresh, too. Fresh keys must remain unknown to the adversary, which we define along the lines of formalizations of security for probabilistic encryption [12, 8, 9].

PROTOCOLS. Four protocols are specified. Protocol MAP1, an extension of the 2PP of [3], is a mutual authentication protocol for an arbitrary set I of players. Protocol MAP2 is an extension of MAP1, allowing arbitrary text strings to be authenticated along with its flows. Protocol AKEP1 is a simple authenticated key exchange which uses MAP2 to do the key distribution. Protocol AKEP2 is a particularly efficient authenticated key exchange which introduces the idea of "implicitly" distributing a key; its flows are identical to MAP1, but it accomplishes a key distribution all the same. The primitive required for all of these protocols is a pseudorandom function.

PROOFS OF SECURITY. Assuming that pseudorandom functions exist, each protocol that we give is proven to meet the definition for the task which this protocol is claimed to carry out. The proofs for MAP1 and AKEP1 are given in this paper; the proofs for MAP2 and AKEP2 are omitted because they are essentially identical. The asymptotics implicit in all of our proofs are not so bad as to render the reductions meaningless for cryptographic practice. In other words, if one had a practical method to defeat the entity authentication this would translate into a practical method to defeat the underlying pseudorandom function.

DESIGN FOR PRACTICE. Every protocol presented in this paper is practical. Each is efficient in terms of rounds, communication, and computation. This efficiency was designed into our protocols in part through the choice of the underlying primitive—a pseudorandom function.

From a theoretical perspective, the existence of pseudorandom functions and the existence of many other important cryptographic primitives (e.g., one-way functions, pseudorandom generators, digital signatures) are all equivalent [16, 14, 10, 23].[4] From a practical perspective, pseudorandom functions (with the right domain and range) are a highly desirable starting point for efficient protocols in the symmetric setting. The reason is that beginning with primitives like DES and MD5 one can construct efficient pseudorandom functions with arbitrary domain and range lengths, and these constructions are themselves provably secure given plausible assumptions about DES and MD5. See Section 6 for discussion of these issues.

IMPLEMENTATIONS. A derivative of our AKEP2 is implemented in an IBM prototype of a secure high speed transport protocol. Another derivative of AKEP2 is implemented in an IBM product for Remote LAN Access.

Combining ideas from our proofs and a lemma from [1], we can show that a special case of the 2PP of [3] meets our definition of a secure mutual authentication. (See the end of Section 4 in our full paper for further details.) The 2PP protocol is implemented in an IBM prototype called *KryptoKnight* [20].

[4] We remark that the existence of a secure mutual authentication protocol implies the existence of a one-way function, as can be shown using techniques of [15]; thus mutual authentication also exists if and only if one-way functions do.

1.2 History and Related Work

PROVABLE SECURITY. Provable security means providing: (1) a formal definition of the goal; (2) a protocol; (3) a statement of a (standard) assumption; and (4) a proof that the protocol meets its goal given the assumption. The notion emerged in the work of Blum-Micali [4] and Yao [26] (who introduced provably secure pseudorandom generators) and Goldwasser-Micali [12] (who introduced provably secure encryption). A definition for digital signatures (Goldwasser, Micali and Rivest [13]) took slightly longer. We follow in spirit this early foundational work and enable entity authentication to join the ranks of those key primitives having a well-defined goal proven to be achievable under a standard complexity-theoretic assumption.

PROTOCOLS. The number of protocols suggested for entity authentication is too large to survey here; see [5, 17] for some examples.

TOWARDS A MODEL AND DEFINITIONS. Bird, Gopal, Herzberg, Janson, Kutten, Molva and Yung [3] described a new class of attacks, called "interleaving attacks," which they used to break existing protocols. They then suggested a protocol (2PP) defeated by none of these attacks. The recognition of interleaving attacks helped lead us to the formal model of Section 3, and our MAP1 protocol is an extension of 2PP. However, while an analysis such as theirs is useful as a way to spot errors in a protocol, resistance to interleaving attacks does not make a satisfactory notion of security; in particular, it is easy to construct protocols which are insecure but defeated by no attack from the enumeration. When our work was announced, the authors of [3] told us that they understood this limitation and had themselves been planning to work on general definitions; they also told us that the CBC assumption of their paper [3, Definition 2.1] was intended for proving security under a general definition.

Mentioned in the introduction of [3] is an idea of "matching histories." Diffie, Van Oorschot and Wiener [6] expand on this to introduce a notion of "matching protocol runs." They refine this idea to a level of precision adequate to help them separate out what are and what are not "meaningful" attacks on the protocols they consider. Although [6] stops short of providing any formal definition or proof, the basic notion these authors describe is the same as ours and is the basis of a definition of entity authentication. Thus there is a clear refinement of definitional ideas first from [3] to [6], and then from [6] to our work.

RELATION TO OTHER FOUNDATIONAL WORK. Beginning with the paper of Burrows, Abadi and Needham [5], the "logic-based approach" attempts to *reason* that an authentication protocol is correct as it evolves the set of *beliefs* of its participants. This idea is useful and appealing, but it has not been used to define when an arbitrary set of flows constitutes a secure entity authentication. Nor does a correctness proof in this setting guarantee that a protocol is "right," but only that it lacks the flaws in reasoning captured by the underlying logic.

More closely related to our approach is the idea of a *non-transferable proof,* a notion for (asymmetric, unilateral) authentication due to Feige, Fiat and Shamir [7]. Here an (honest) claimant P interacts with a (cheating) verifier $\tilde{V}$,

and then a ($\tilde{V}$-conspiring cheating) prover $\tilde{P}$ tries to convince an (honest) verifier V that she ($\tilde{P}$) is really P. This definition accurately models a world of smart-card claimants and untrusted verifiers, but not a distributed system of always-running processes.

PUBLICATION NOTES. A preliminary version of this paper (which included, in addition to the material here, definitions three party authentication) appeared in the proceedings of an IBM internal conference in October 1992. The version of this paper you are now reading has been edited due to page limits. Ask either author for the complete version.

2 Preliminaries

The set of infinite strings is $\{0,1\}^\infty$ and $\{0,1\}^{\leq L}$ is the set of strings of length at most L. The empty string is λ. When a, b, c, ... are strings used in some context, by $a.b.c.\cdots$ we denote an encoding of these strings such that each constituent string is efficiently recoverable given the encoding and the context of the string's receipt. In our protocols, concatenation will usually be adequate for this purpose. A function is *efficiently computable* if it can be computed in time polynomial in its first argument. A real-valued function $\epsilon(k)$ is *negligible* if for every $c > 0$ there exists a $k_c > 0$ such that $\epsilon(k) < k^{-c}$ for all $k > k_c$. The protocols we consider are two party ones, formally specified by an efficiently computable function Π on the following inputs:

1^k — the "security parameter" — $k \in \mathsf{N}$.
i — the "identity of the sender" — $i \in I \subseteq \{0,1\}^k$.
j — the "identity of the (intended) partner" — $j \in I \subseteq \{0,1\}^k$.
a — the "secret information of the sender" — $a \in \{0,1\}^*$.
κ — the "conversation so far" — $\kappa \in \{0,1\}^*$.
r — the "random coin flips of the sender" — $r \in \{0,1\}^\infty$.

The value of $\Pi(1^k, i, j, a, \kappa, r) = (m, \delta, \alpha)$ specifies:
m — the "next message to send out" — $m \in \{0,1\}^* \cup \{*\}$.
δ — the "decision" — $\delta \in \{\mathsf{A}, \mathsf{R}, *\}$.
α — the "private output" — $\alpha \in \{0,1\}^* \cup \{*\}$.

Here I is a set of *identities* which defines the *players* who can participate in the protocol. Although our protocols involve only two parties, the set of players I could be larger, to handle the possibility (for example) of an arbitrary pool of players who share a secret key. Elements of I will sometimes be denoted A or B (Alice and Bob), rather than i, j; we will switch back and forth irrationally between these notations. We stress that A, B (and i, j) are variables ranging over I (not fixed members of I), so $A = B$ (or $i = j$) is quite possible. Note that the adversary is *not* a player in our formalization. The value a that a player sees is the private information provided to him. This string is sometimes called the *long-lived key* (or LL-key) of a player. In the case of (pure) symmetric authentication, all players $i \in I$ will get the same LL-key, and the adversary will be denied this key. In general, a LL-key generator $\mathcal{G}$ associated to a protocol will

determine who gets what initial LL-key (see below). The value "$*$" is supposed to suggest, for m, that "the player sends no message." For δ, it means that "the player has not yet reached a decision." For α, it means "the player does not currently have any private output." The values A and R, for δ, are supposed to suggest "accept" and "reject," respectively. We denote the t-th component of Π (for $t \in \{1,2,3\}$) by Π_t. Acceptance usually does not occur until the end of the protocol, although rejection may occur at any time. Some protocol problems, such as mutual authentication, do not make use of the private output; these protocol are concerned only with acceptance or rejection. For others, including key exchange protocols, the private output of a party will be what this party thinks is the key which has been exchanged. It is convenient to assume that once a player has accepted or rejected, this output cannot change. To each protocol is associated its number of moves, R. In general this is a polynomially bounded, polynomial time computable function of the security parameter; in all our protocols, however, it is a constant.

Associated to a protocol is a *long-lived key generator* (LL-key generator) $\mathcal{G}(1^k, \iota, r_G)$. This is a polynomial time algorithm which takes as input a security parameter 1^k, the identity of a party $\iota \in I \cup \{\mathsf{E}\}$, and an infinite string $r_G \in \{0,1\}^\infty$ (coin flips of the generator). For all of the protocols of this paper, the associated LL-key generator will be a *symmetric* one, where for each $i, j \in I$ we have that $\mathcal{G}(1^k, i, r_G) = \mathcal{G}(1^k, j, r_G)$; while, on the other hand, $\mathcal{G}(1^k, \mathsf{E}, r_G) = \lambda$. The value of $\mathcal{G}(1^k, i, r_G)$ will just be a prefix of r_G (that is, a random string). The length of this prefix will vary according to the protocol we consider.

3 A Communication Model for Distributed Security

Formally the adversary E is a probabilistic machine[5] $E(1^k, a_E, r_E)$ equipped with an infinite collection of oracles $\Pi^s_{i,j}$, for $i, j \in I$ and $s \in \mathsf{N}$. Oracle $\Pi^s_{i,j}$ models player i attempting to authenticate player j in "session" s. Adversary E communicates with the oracles via queries of the form (i, j, s, x) written on a special tape. The query is intended to mean that E is sending message x to i, claiming it is from j in session s. Running a protocol Π (with LL-key generator $\mathcal{G}$) in the presence of an adversary E, using security parameter k, means performing the following experiment:

(1) Choose a random string $r_G \in \{0,1\}^\infty$ and set $a_i = \mathcal{G}(1^k, i, r_G)$, for $i \in I$, and set $a_E = (1^k, \mathsf{E}, r_G)$.

(2) Choose a random string $r_E \in \{0,1\}^\infty$ and, for each $i, j \in I$, $s \in \mathsf{N}$, a random string $r^s_{i,j} \in \{0,1\}^\infty$.

(3) Let $\kappa^s_{i,j} = \lambda$ for all $i, j \in I$ and $u \in \mathsf{N}$. (The variable $\kappa^s_{i,j}$ will keep track of the conversation that $\Pi^s_{i,j}$ engages in.)

[5] Adversaries can be uniform or non-uniform, and the results of this paper hold in both cases, with uniform adversaries requiring a uniform complexity assumptions and non-uniform adversaries requiring non-uniform ones.

(4) Run adversary E on input $(1^k, a_E, r_E)$, answering oracle calls as follows. When E asks a query (i, j, s, x), oracle $\Pi^s_{i,j}$ computes $(m, \delta, \alpha) = \Pi(1^k, i, j, a_i, \kappa^s_{i,j} \cdot x, r^s_{i,j})$ and answers with (m, δ). Then $\kappa^s_{i,j}$ gets replaced by $\kappa^s_{i,j} \cdot x$.

We point out that in response to an oracle call E learns not only the outgoing message but also whether or not the oracle has accepted or rejected. (For convenience of discourse, we often omit mention of the latter.) According to the above, E doesn't learn the oracle's private output. For some problems (such as authenticated key exchange) we will need to give the adversary the power to sometimes learn these private outputs. Such an extension is handled by specifying a new kind of oracle query and then indicating how the experiment is extended with responses to the new class of queries. An adversary is called *benign* if it is deterministic and restricts its action to choosing a pair of oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ and then faithfully conveying each flow from one oracle to the other, with $\Pi^s_{i,j}$ beginning first. While the choice of i, j, s, t is up to the adversary, this choice is the same in all executions with security parameter k.

In a particular execution of a protocol, the adversary's i-th query to an oracle is said to occur at time $\tau = \tau_i \in \mathbf{R}$. We intentionally do not specify $\{\tau_i\}$, except to demand that $\tau_i < \tau_j$ when $i < j$. Conforming notions of time include "abstract time," where $\tau_i = i$, and "Turing machine time," where $\tau_i =$ the i-th step in E's computation, when parties are realized by interacting Turing machines.

4 Entity Authentication

A central idea in the definition is that of matching conversations. Consider running the adversary E with security parameter k. When E terminates, each oracle $\Pi^s_{i,j}$ has had a certain conversation $\kappa^s_{i,j}$ with E, and it has reached a certain decision $\delta \in \{\mathsf{A}, \mathsf{R}, *\}$. Fix an execution of an adversary E (that is, fix the coins of the LL-key generator, the oracles, and the adversary). For any oracle $\Pi^s_{i,j}$ we can capture its *conversation* (for this execution) by a sequence

$$K = (\tau_1, \alpha_1, \beta_1),\ (\tau_2, \alpha_2, \beta_2),\ \ldots,\ (\tau_m, \alpha_m, \beta_m).$$

This sequence encodes that at time τ_1 oracle $\Pi^s_{i,j}$ was asked α_1 and responded with β_1; and then, at some later time $\tau_2 > \tau_1$, the oracle was asked α_2 and answered β_2; and so forth, until, finally, at time τ_m it was asked α_m and answered β_m. Adversary E terminates without asking oracle $\Pi^s_{i,j}$ any more questions. Suppose oracle $\Pi^s_{i,j}$ has conversation prefixed by $(\tau_1, \alpha_1, \beta_1)$. Then if $\alpha_1 = \lambda$ we call $\Pi^s_{i,j}$ an *initiator* oracle; if α_1 is any other string we call $\Pi^s_{i,j}$ a *responder* oracle. We now define matching conversations. For simplicity we focus on the case where R is odd; the case of even R is analogous and is left to the reader. Explanations follow the formal definition.

Definition 1. (Matching conversations) *Fix a number of moves $R = 2\rho - 1$ and an R-move protocol Π. Run Π in the presence of an adversary E and consider two oracles, $\Pi^s_{A,B}$ and $\Pi^t_{B,A}$, that engage in conversations K and K', respectively.*

(1) *We say that K' is a* matching conversation *to K if there exist $\tau_0 < \tau_1 < \ldots < \tau_R$ and $\alpha_1, \beta_1, \ldots, \alpha_\rho, \beta_\rho$ such that K is prefixed by*

$$(\tau_0, \lambda, \alpha_1), (\tau_2, \beta_1, \alpha_2), (\tau_4, \beta_2, \alpha_3), \ldots, (\tau_{2\rho-4}, \beta_{\rho-2}, \alpha_{\rho-1}), (\tau_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho)$$

and K' is prefixed by

$$(\tau_1, \alpha_1, \beta_1), (\tau_3, \alpha_2, \beta_2), (\tau_5, \alpha_3, \beta_3), \ldots, (\tau_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}) .$$

(2) *We say that K is a* matching conversation *to K' if there exist $\tau_0 < \tau_1 < \ldots < \tau_R$ and $\alpha_1, \beta_1, \ldots, \alpha_\rho, \beta_\rho$ such that K' is prefixed by*

$$(\tau_1, \alpha_1, \beta_1), (\tau_3, \alpha_2, \beta_2), (\tau_5, \alpha_3, \beta_3), \ldots, (\tau_{2\rho-3}, \alpha_{\rho-1}, \beta_{\rho-1}), (\tau_{2\rho-1}, \alpha_\rho, *) .$$

and K is prefixed by

$$(\tau_0, \lambda, \alpha_1), (\tau_2, \beta_1, \alpha_2), (\tau_4, \beta_2, \alpha_3), \ldots, (\tau_{2\rho-4}, \beta_{\rho-2}, \alpha_{\rho-2}), (\tau_{2\rho-2}, \beta_{\rho-1}, \alpha_\rho)$$

Case (1) defines when the conversation of a responder oracle matches the conversation of an initiator oracle. Case (2) defines when the conversation of an initiator oracle matches the conversation of a responder oracle. Let us paraphrase our definition. Consider an execution in which $\Pi^s_{A,B}$ is an initiator oracle and $\Pi^t_{B,A}$ is a responder oracle. If every message that $\Pi^s_{A,B}$ sends out, except possibly the last, is subsequently delivered to $\Pi^t_{B,A}$, with the response to this message being returned to $\Pi^s_{A,B}$ as its own next message, then we say that the conversation of $\Pi^t_{B,A}$ matches that of $\Pi^s_{A,B}$. Similarly, if every message that $\Pi^t_{B,A}$ receives was previously generated by $\Pi^s_{A,B}$, and each message that $\Pi^t_{B,A}$ sends out is subsequently delivered to $\Pi^s_{A,B}$, with the response that this message generates being returned to $\Pi^t_{B,A}$ as its own next message, then we say that the conversation of $\Pi^s_{A,B}$ matches the one of $\Pi^t_{B,A}$. Note that this second condition is easily seen to imply the first one.

We comment that the party who sends the last flow ($\Pi^s_{A,B}$, above) can't "know" whether or not its last message was received by its partner, so when this oracle accepts accepts, it cannot "know" (assuming this last message to be relevant) whether or not its partner will accept. This asymmetry is an inherent aspect of authentication protocols with a fixed number of moves.

We will say that oracle $\Pi^t_{j,i}$ has a matching conversation with oracle $\Pi^s_{i,j}$ if the first has conversation K', the second has conversation K, and K' matches K. Either party here may be the initiator.

We require that any mutual authentication protocol have $R \geq 3$ rounds. We implicitly make this assumption throughout the remainder of this paper. Let No-Matching$^E(k)$ be the event that there exist i, j, s such that $\Pi^s_{i,j}$ accepted and there is no oracle $\Pi^t_{j,i}$ which engaged in a matching conversation.

Definition 2. (Secure mutual authentication) *We say that Π is a* secure mutual authentication *protocol if for any polynomial time adversary E,*

(1) (Matching conversations $\Rightarrow$ acceptance.) *If oracles $\Pi^s_{A,B}$ and $\Pi^t_{B,A}$ have matching conversations, then both oracles accept.*

(2) (Acceptance $\Rightarrow$ matching conversations.) *The probability of* No-Matching$^E(k)$ *is negligible.*

An oracle's matching partner is unique. Formally, let Multiple-Match$^E(k)$ be the event that some $\Pi^s_{i,j}$ accepts, and there are at least two distinct oracles $\Pi^t_{j,i}$ and $\Pi^{t'}_{j,i}$ which have had matching conversations with $\Pi^s_{i,j}$. The proof of the following is in Appendix C (omitted here due to lack of space; see our full paper).

Proposition 3. *Suppose Π is a secure MA protocol. Let E be any polynomial time adversary. Then the probability of* Multiple-Match$^E(k)$ *is negligible.*

We now proceed to protocols. Let f be a pseudorandom function (PRF) family [10]. Denote by f_a: $\{0,1\}^{\leq L(k)} \to \{0,1\}^{l(k)}$ the function specified by key a. In general, the length of the key, the length L of the input to f_a, and the length l of the output, are all functions of the security parameter. Here we assume the key length is just k, and, for our first protocol (MAP1) it suffices to assume $L(k) = 4k$ and $l(k) = k$. For any string $x \in \{0,1\}^{\leq L(k)}$ define $[x]_a = (x, f_a(x))$; this will serve as an authentication of message x [10, 11]. For any $i \in I$, $[i\,.\,x]_a$ will serve as i's authentication of message x.

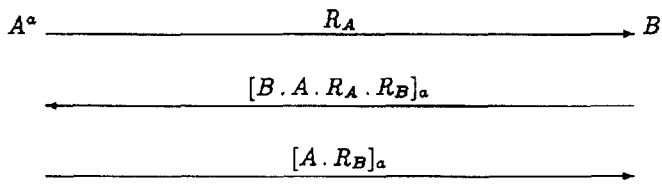

Fig. 1. *Protocol MAP1: a mutual authentication of any two principals, A and B, among a set of principals I who share a key a.*

Our first protocol (called "MAP1," for "mutual authentication protocol one") is represented by Figure 1. Alice (A) begins by sending Bob (B) a random challenge R_A of length k. Bob responds by making up a random challenge R_B of length k and returning $[B\,.\,A\,.\,R_A\,.\,R_B]_a$. Alice checks that this message is of the right form and is correctly tagged as coming from B. If it is, Alice sends Bob the message $[A\,.\,R_B]_a$ and accepts. Bob checks that this message is of the right form and is correctly tagged as coming from A, and, if it is, he accepts. We stress that checking the message is of the right form, for A in the second flow, includes checking that the nonce present in the message is indeed the same nonce she sent in the first flow; similarly for B with respect to checking the third flow. We comment that $A = B$ is permitted; these are any two identities in the set I. The proof of the following Theorem 4 appears in Appendix A.

Theorem 4. *(MAP1 is a secure MA) Suppose f is a pseudorandom function family. Then protocol MAP1 described above and based on f is a secure mutual authentication.*

5 Authenticated Key Exchange

Fix $S = \{S_k\}_{k \in \mathbb{N}}$ with each S_k a distribution over $\{0,1\}^{\sigma(k)}$, for some polynomial $\sigma(k)$. The intent of an AKE will be both to authenticate entities *and* to distribute a "session key" sampled from S_k. When a player accepts, his private output will be interpreted as the session key which he has computed. Formally, the session key α will be defined by Π_3. For simplicity, we assume that an accepting player always has a string-valued private output of the right length (that is, if $\Pi_2 = \mathsf{A}$ then $\Pi_3 \in \{0,1\}^{\sigma(k)}$), while a non-accepting player has a session key of $*$ (that is, if $\Pi_2 \in \{\mathsf{R}, *\}$ then $\Pi_3 = *$).

Compromise of a session key should have minimal consequences. For example, its revelation should not allow one to subvert subsequent authentication, nor should it leak information about other (as yet uncompromised) session keys. To capture this requirement we extend the interaction of the adversary with its oracles by adding a new type of query, as follows: we say that the adversary can learn a session key $\alpha_{i,j}^s$ of an oracle $\Pi_{i,j}^s$ by issuing to the oracle a distinguished reveal query, which takes the form (i, j, s, reveal). The oracles answers $\alpha_{i,j}^s$. To quantify the power of an adversary who can perform this new type of query, we make the following definitions. Initially, each oracle $\Pi_{i,j}^s$ is declared *unopened*, and so it remains until the adversary generates a reveal query (i, j, s, reveal). At this point, the oracle is declared *opened*. We say that an oracle $\Pi_{i,j}^s$ is *fresh* if the following three conditions hold: First, $\Pi_{i,j}^s$ has accepted. Second, $\Pi_{i,j}^s$ is unopened. Third, there is no opened oracle $\Pi_{j,i}^t$ which engaged in a matching conversation with $\Pi_{i,j}^s$. When oracle $\Pi_{i,j}^s$ is fresh, we will also say that "the oracle holds a fresh session key."

We want that the adversary should be unable to understand anything interesting about a fresh session key. This can be formalized along the lines of security of probabilistic encryption; the particular formalization we will adapt is that of (polynomial) indistinguishability of encryptions [12, 8, 9]. We demand that at the end of a secure AKE the adversary should be unable to distinguish a fresh session key α from a random element of S_k. After the adversary has asked all the (i, j, s, x) and (i, j, s, reveal) queries that she wishes to ask, the adversary asks of a fresh oracle $\Pi_{i,j}^s$ a single query (i, j, s, test). The query is answered by flipping a fair coin $b \leftarrow \{0,1\}$ and returning $\alpha_{i,j}^s$ if $b = 0$, or else a random sample from S_k if $b = 1$. The adversary's job is to guess b. To this end, she outputs a bit Guess, and then terminates. Let $\text{Good-Guess}^E(k)$ be the event that $\text{Guess} = b$, when the protocol is executed with security parameter k; in other words, this is the probability the adversary has correctly identified whether she was given the real session key or just a sample from S_k. Let

$$\mathit{advantage}^E(k) = \max\left\{ 0,\ \Pr\left[\text{Good-Guess}^E(k)\right] - \tfrac{1}{2} \right\} .$$

Definition 5. (Authenticated Key Exchange (AKE)) *Protocol Π is a* secure AKE *over $S = \{S_k\}_{k \in \mathbb{N}}$ if Π is a secure mutual authentication protocol, and, in addition, the following are true:*

(1) (Benign adversary $\Rightarrow$ keys according to S_k) *Let B be any benign adversary and let $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ be its chosen oracles in the experiment with security parameter k. Then both oracles always accept, $\alpha^s_{i,j} = \alpha^t_{j,i}$, and moreover this random variable is distributed according to S_k.*

(2) (Session key is protected) *Let E be any polynomial time adversary. Then advantage$^E(k)$ is negligible.*

The first condition says that if flows are honestly conveyed then a session key is agreed upon, and this key is properly distributed. The second condition says that the adversary can't tell this session key from a random string of the same distribution.

Since the protocol is assumed to be a secure mutual authentication, we know that if oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have matching conversations then they both accept. From the first condition it follows that they will also have the same session key.

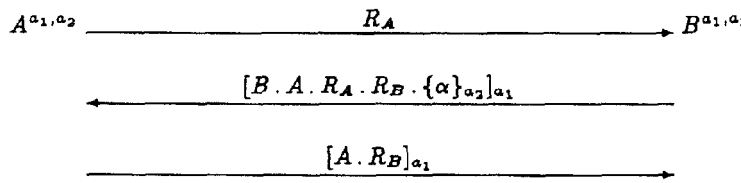

Fig. 2. *Protocol AKEP1: The value α is the session key distributed.*

We now present a protocol for AKE. Let $S = \{S_k\}$ be a family of samplable distributions on $\{0,1\}^{\sigma(k)}$. The parties share a $2k$ bit LL-key which we denote a_1, a_2. The first part, a_1, is taken as the key to the pseudorandom function family f, yielding a PRF f_{a_1}: $\{0,1\}^{\leq L(k)} \to \{0,1\}^k$ to be used for message authentication; this time, $L(k) = 5k + \sigma(k)$ will suffice. The second part, a_2, is used as a key to another pseudorandom family f' with the property that f'_{a_2}: $\{0,1\}^k \to \{0,1\}^{\sigma(k)}$. A probabilistic encryption of string $\alpha \in \{0,1\}^{\sigma(k)}$ is defined by $\{\alpha\}_{a_2} \stackrel{\text{def}}{=} (r, f'_{a_2}(r) \oplus \alpha)$, with r selected at random. Party B chooses the session key α from S_k and sets Text_2 to be $\{\alpha\}_{a_2}$. The strings Text_1 and Text_3 of MAP2 are set to λ. This protocol, which we call AKEP1, is shown in Figure 2. It is important that a_2 (the key used for encryption) be distinct from a_1 (the shared key used for the message authentication). Formally, the LL-key generator $\mathcal{G}$ provides the parties $i \in I$ with a $2k$-bit shared key. The two keys need not be independent, however; the generator could set $a_i = f_a(i)$ $(i = 1, 2)$ where a is a random k-bit key and f_a is a pseudorandom function. The proof of the following theorem is given in Appendix B.

Theorem 6. *Let $S = \{S_k\}$ be samplable, and suppose f, f' are pseudorandom function families with the parameters specified above. Then the protocol AKEP1 based on f, f' is a secure AKE over S.*

A more efficient (in terms of communication complexity) AKE protocol may be devised by using an "implicit" key distribution. In this case, the flows between A and B are the same as in MAP1 and one (or more) of the parameters already present in its flows (say R_B) is used to define the session key. Specifically, let $S = \{S_k\}$ be a family of distributions given by $S_k = g(U_k)$, for some deterministic, polynomial-time computable function g, where U_k is the uniform distribution on k-bit strings; for example $S_k = U_k$ and g the identity, the most useful choice in practice. Again the parties share a $2k$ bit LL-key a_1, a_2, with a_1 being used as the key in MAP1 (so $L(k) = 4k$). Let f' be a pseudorandom *permutation* family [18]; f'_{a_2} specifies a permutation on $\{0,1\}^k$. Define AKEP2 by having its flows be identical to MAP1 with a_1 being used for message authentication. Each accepting party outputs session key $\alpha = g(f'_{a_2}(R_B))$. This protocol, which we call AKEP2, is shown in Figure 3. Modifying the proof of Theorem 6 we can show the following:

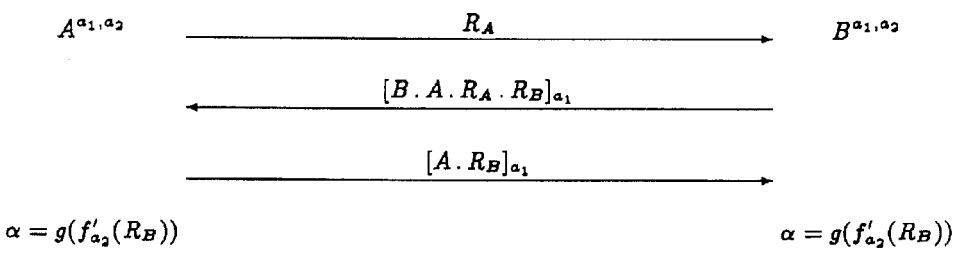

Fig. 3. *Protocol AKEP2: The Implicit Key Exchange Protocol. The value α is the session key "implicitly" distributed.*

Theorem 7. *Let $S = \{S_k\}$ be given by $S_k = g(U_k)$, for some polynomial time g. Suppose f, f' are a pseudorandom function family and a pseudorandom permutation family, with the parameters specified above. Then the protocol AKEP2 based on f, f' is a secure AKE over S.*

6 From Theory to Practice

Cryptographic practice provides good PRFs on particular input lengths l (for example, DES for $l = 64$ [18, 19]). In contrast, our protocols need PRFs for arbitrary input lengths. In devising such PRFs we prefer not to rely purely on heuristics but instead to give provably-correct constructions of arbitrary length PRFs based on fixed length PRFs and collision-free hash functions. The lemmas underlying our constructions are from [1] and are summarized with additional material in Appendix D (omitted here due to lack of space; see our full paper). The exception is the third construction given below; we'll discuss it when we get there.

PRIMITIVES. The algorithm of the DES specifies for each 64 bit key a a permutation DES_a from $\{0,1\}^{64}$ to $\{0,1\}^{64}$. The viewpoint adopted here —suggested by Luby and Rackoff [18, 19]— is to regard DES as a pseudorandom permutation, with respect to practical computation.

The MD5 function [22] maps an arbitrary string x into a 128-bit string $\mathrm{MD5}(x)$. It is intended that this function be a collision-free hash function, with respect to practical computation.

NOTATION. Let g_a denote a PRF of l bits to l bits. Suppose y has length a multiple of l bits, and write it as a sequence of l bit blocks, $y = y_1 \ldots y_n$. The cipher block chaining (CBC) operator defines

$$\mathrm{CBC}_a^g(y_1 \ldots y_n) = \begin{cases} g_a(y_1) & \text{if } n = 1 \\ g_a(\mathrm{CBC}_a^g(y_1 \ldots y_{n-1}) \oplus y_n) & \text{otherwise} \end{cases}$$

Let H denote a collision free hash function of $\{0,1\}^*$ to $\{0,1\}^{2l}$. Let $H_1(x)$ and $H_2(x)$ denote the first l bits of $H(x)$ and the last l bits of $H(x)$, respectively. Finally $(x)_l$ will denote some standard padding of x to string of length a multiple of l bits; for example, always add a 1 and then add enough zeroes to get to a length which is a multiple of l.

CONSTRUCTIONS. For concreteness, we suggest three constructions of a PRF f_a mapping long inputs to short outputs. Below, let $l = 64$, $g = \mathrm{DES}$, and $H = \mathrm{MD5}$ The key a has length 64 bits.

(1) *The CBC PRF.* Let $f_a(x)$ be the first $l/2$ bits of $\mathrm{CBC}_a^g((x)_l \,.\, |(x)_l|)$, where $|y|$ is the length of y encoded as an l-bit string. This construction is justified by Lemma 12 of Appendix D (omitted).[6]

(2) *The CBC/Hash PRF.* Let $f_a(x)$ be the first $l/2$ bits of $g_a(\, g_a(H_1(x)) \oplus H_2(x)\,) = \mathrm{CBC}_a^g(H(x))$. This construction is justified by Corollary 14 (omitted). In software this is significantly more efficient than the CBC construction, requiring one hash and two DES operations.

(3) *The Pure Hash PRF.* Let $f_a(x)$ be the first $l/2$ bits of $H(x \,.\, a)$. This construction was suggested in [24] as a message authentication code; we suggest the stronger assumption that it is a PRF. However no standard assumption about H of which we are aware can be used to justify the the security of this construction, and it should be viewed more as a heuristic than the two constructions suggested above.[7]

Similar constructions can be given using other primitives; for example the SHA instead of MD5, etc.

---

[6] Lemma 12 does not require us to drop the last $l/2$ bits of the output. We drop them for two reasons. The first is efficiency. The second is specific to DES and will not be discussed here.

[7] See [2] for another viewpoint.

We stress the importance in security considerations of the CBC and Hash Lemmas of Appendix D; the lack of such lemmas has lead in the past to more complex assumptions about the security of CBC and other constructions (e.g., [3, Definition 2.1]).

Acknowledgments

We thank Bob Blakley, Oded Goldreich, Amir Herzberg, Phil Janson, and the member of the CRYPTO 93 committee for all of their comments and suggestions. Especially we acknowledge Oded as suggesting that we formulate the security of fresh keys along the lines of polynomial indistinguishability (instead of the equivalent semantic security formulation we had before); and Amir for suggesting that we give Proposition 3, specify MAP1 in a way that does not assume authenticating entities are distinct agents from a set of cardinality two, and that to avoid the efficiency loss from padding we explicitly specify our pseudorandom functions as acting on $\{0,1\}^{\leq L(k)}$ instead of $\{0,1\}^{L(k)}$.

References

1. M. Bellare, U. Feige, J. Kilian, M. Naor and P. Rogaway, "The security of cipher block chaining," manuscript (1993).
2. M. Bellare and P. Rogaway, "Random oracles are practical: a paradigm for designing efficient protocols," *Proceedings of 1st ACM Conference on Computer and Communications Security*, November 1993.
3. R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M. Yung, "Systematic design of two-party authentication protocols," *Advances in Cryptology — Proceedings of CRYPTO 91*, Springer-Verlag, 1991.
4. M. Blum and S. Micali, "How to generate cryptographically strong sequences of pseudo-random bits," *SIAM Journal on Computing* **13**(4), 850-864 (November 1984).
5. M. Burrows, M. Abadi and R. Needham, "A logic for authentication," DEC Systems Research Center Technical Report 39, February 1990. Earlier versions in *Proceedings of the Second Conference on Theoretical Aspects of Reasoning about Knowledge*, 1988, and *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, 1989.
6. W. Diffie, P. Van Oorschot and M. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, 2, 107–125 (1992).
7. U. Feige, A. Fiat and A. Shamir, "Zero knowledge proofs of identity," *Journal of Cryptology*, Vol. 1, pp. 77–94 (1987).
8. O. Goldreich, "Foundations of cryptography," class notes, Technion University, Computer Science Department, Spring 1989.
9. O. Goldreich, "A uniform complexity treatment of encryption and zero-knowledge," *Journal of Cryptology*, Vol. 6, pp. 21-53 (1993).
10. O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions," *Journal of the ACM*, Vol. 33, No. 4, 210–217, (1986).

11. O. Goldreich, S. Goldwasser and S. Micali, "On the cryptographic applications of random functions," *Advances in Cryptology — Proceedings of CRYPTO 84*, Springer-Verlag, 1984.
12. S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences* Vol. 28, 270-299 (April 1984).
13. S. Goldwasser, S. Micali and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal of Computing*, Vol. 17, No. 2, 281–308, April 1988.
14. J. Håstad, "Pseudo-random generators under uniform assumptions," *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, ACM (1990).
15. R. Impagliazzo and M. Luby, "One-way functions are essential for complexity based cryptography," *Proceedings of the 30th Annual IEEE Symposium on the Foundations of Computer Science*, IEEE (1989).
16. R. Impagliazzo, L. Levin and M. Luby, "Pseudo-random generation from one-way functions," *Proceedings of the 21st Annual ACM Symposium on the Theory of Computing*, ACM (1989).
17. ISO/IEC 9798-2, "Information technology – Security techniques – Entity authentication – Part 2: Entity authentication using symmetric techniques." Draft 12, September 1992.
18. M. Luby and C. Rackoff, "How to construct pseudorandom permutations from pseudorandom functions," *SIAM J. Computing*, Vol. 17, No. 2, April 1988.
19. M. Luby and C. Rackoff, "A study of password security," manuscript.
20. R. Molva, G. Tsudik, E. Van Herreweghen and S. Zatti, "*KryptoKnight* authentication and key distribution system," ESORICS 92, Toulouse, France, November 1992.
21. R. Needham and M. Schroeder, "Using encryption for authentication in large networks of computers," *Communications of the ACM*, Vol. 21, No. 12, 993–999, December 1978.
22. R. Rivest, "The MD5 message-digest algorithm," IETF Network Working Group, RFC 1321, April 1992.
23. J. Rompel, "One-way functions are necessary and sufficient for secure signatures," *Proceedings of the 22nd Annual ACM Symposium on the Theory of Computing*, ACM (1990).
24. G. Tsudik, "Message authentication with one-way hash functions," *Proceedings of Infocom 92.*
25. P. Van Oorschot, "Extending cryptographic logics of belief to key agreement protocols," *Proceedings of 1st ACM Conference on Computer and Communications Security*, November 1993.
26. Yao, A. C., "Theory and applications of trapdoor functions," *Proceedings of the 23rd Annual IEEE Symposium on the Foundations of Computer Science*, IEEE (1982).

A Proof of Theorem 4

We prove that MAP1 is a secure mutual authentication protocol under the assumption that f is a PRF. The first condition of Definition 2 is easily verified; it merely says that when the messages between A and B are faithfully relayed to one another, each party accepts. We now prove that the second condition holds.

Fix an adversary E. Recall that the domain of our PRF is $\{0,1\}^{\leq L(k)}$ and its range is $\{0,1\}^k$. In the following, Π will denote MAP1. In what follows we will be considering a variety of experiments involving the running of E with its oracles. In order to avoid confusion, we will refer to the experiment of running E with MAP1 (the experiment about which we wish to prove our theorem) as the "real" experiment.

MAP1 WITH A g ORACLE. Let g be a function of $\{0,1\}^{\leq L(k)}$ to $\{0,1\}^k$. Let $[x]_g = (x, g(x))$. MAP1^g denotes the protocol in which, instead of a shared secret a, the parties share an oracle for g, and they compute $[x]_g$ wherever MAP1 asks them to compute $[x]_a$. We define the experiment of running E for MAP1^g to be the same as the experiment of running E for MAP1 except for the following difference. There is no shared secret a; instead, the oracles $\Pi^s_{i,j}$ all have access to a common g oracle and compute their flows according to MAP1^g. Note E is not given access to the g oracle. When $g = f_a$ for randomly chosen a, this experiment coincides with the real experiment. Of interest in our proof is the case of g being a truly random function; we call this the random MAP1 experiment.

THE RANDOM MAP1 EXPERIMENT. In the random MAP1 experiment we select g as a random function of $\{0,1\}^{\leq L(k)}$ to $\{0,1\}^k$, and then run the experiment of running E with MAP1^g. Recall that No-Matching$^E(k)$ denotes the event that there exists an oracle $\Pi^s_{i,j}$ who accepts although no oracle $\Pi^t_{j,i}$ engaged in a matching conversation; we will refer to it also as the event that the adversary is successful. Recall that an initiator oracle is one who sends a first flow (that is, it plays the role of A in Figure 1) while a responder oracle is one who plays the opposite role (namely that of B in the same Figure). Let $T_E(k)$ denote a polynomial bound on the number of oracle calls made by E, and assume wlog that this is at least two.

Lemma 8. *The probability that the adversary E is successful in the random* MAP1 *experiment is at most $T_E(k)^2 \cdot 2^{-k}$.*

Proof: We split the examination of acceptance into two cases.

Claim 1: Fix A, B, s. The probability that $\Pi^s_{A,B}$ accepts without a matching conversation, given that it is an initiator oracle, is at most $T_E(k) \cdot 2^{-k}$.

Proof. Suppose at time τ_0 oracle $\Pi^s_{A,B}$ sent the flow R_A. Let $\mathcal{R}(\tau_0)$ denote the set of all $R'_A \in \{0,1\}^k$ for which there exist τ, t such that $\Pi^t_{B,A}$ was given R'_A as first flow at a time $\tau < \tau_0$. If $\Pi^s_{A,B}$ is to accept, then at some time $\tau_2 > \tau_0$ it must receive $[B \,.\, A \,.\, R_A \,.\, R_B]_g$ for some R_B. If no oracle previously output this flow, the probability that the adversary can compute it correctly is at most 2^{-k}. So consider the case where some oracle did output this flow. The form of the flow implies that the oracle which output it must be a $\Pi^t_{B,A}$ oracle which received R_A as its own first flow. The probability of this event happening before time τ_0 is bounded by the probability that $R_A \in \mathcal{R}(\tau_0)$, and this probability is at most $[T_E(k) - 1] \cdot 2^{-k}$. If it happened after time τ_0 then we would have a matching conversation. We conclude that the probability that $\Pi^s_{A,B}$ accepts but there is no matching conversation is at most $T_E(k) \cdot 2^{-k}$. □

Claim 2: Fix B, A, t. The probability that $\Pi^t_{B,A}$ accepts without a matching conversation, given that it is a responder oracle, is at most $T_E(k) \cdot 2^{-k}$.

Proof. Suppose at time τ_1 oracle $\Pi^t_{B,A}$ received the flow R_A and responded with $[B \,.\, A \,.\, R_A \,.\, R_B]_g$. If $\Pi^t_{B,A}$ is to accept, then at some time $\tau_3 > \tau_1$ it must receive $[A \,.\, R_B]_g$. If no oracle previously output this flow, the probability that the adversary can compute it correctly is at most 2^{-k}. We must now consider the case where some oracle did output this flow. The form of the flow implies that the oracle which output it must be a $\Pi^s_{A,C}$ oracle.

The interaction of a $\Pi^s_{A,C}$ oracle with E has in general the form

$$(\tau_0, \lambda, R'_A), (\tau_2, [C \,.\, A \,.\, R'_A \,.\, R'_B]_g, [A \,.\, R'_B]_g)$$

for some $\tau_0 < \tau_2$. For any such interaction, except with probability 2^{-k}, there is a $\Pi^u_{C,A}$ oracle which output $[C \,.\, A \,.\, R'_A \,.\, R'_B]_g$ at some time. If $(u, C) \neq (t, B)$ then the probability that $R'_B = R_B$ is at most $[T_E(k)-2] \cdot 2^{-k}$, and thus the probability that the flow $[A \,.\, R'_B]_g$ leads $\Pi^t_{B,A}$ to accept is at most $[T_E(k) - 2] \cdot 2^{-k}$. On the other hand suppose $(u, C) = (t, B)$. It follows that $\tau_0 < \tau_1 < \tau_2 < \tau_3$, $R'_A = R_A$ and $R'_B = R_B$; that is, the conversations match. We conclude that the probability that $\Pi^t_{B,A}$ accepts but there is no matching conversation is at most $T_E(k) \cdot 2^{-k}$. □

The probability that there exists an oracle which accepts without a matching conversation is at most $T_E(k)$ times the bound obtained in the claims, which is $T_E(k)^2 \cdot 2^{-k}$ as desired. □

See our full paper for the argument that these lemmas yield the theorem.

B Proof of Theorem 6

We prove that AKEP1 is a secure authenticated key exchange protocol under the assumption that f, f' are PRFs.

The proof that AKEP1 is a secure mutual authentication protocol is analogous to the proof of Theorem 4 given in Appendix A and is omitted. Condition (1) of Definition 5 is easily verified: the session key α is chosen in AKEP1 according to S_k and so in the presence of a benign adversary the oracles certainly accept, and with this same key. We concentrate on the proof that condition (2) of Definition 5 is satisfied.

Fix an adversary E. Recall that we are using two PRFs: $f_{a_1}\colon \{0,1\}^{\leq L(k)} \to \{0,1\}^k$ and $f'_{a_2}\colon \{0,1\}^k \to \{0,1\}^{\sigma(k)}$. The first is for the authentication and the second is to encrypt the session key. In what follows Π will denote AKEP1, and the "real" experiment will denote the experiment of running E for AKEP1.

AKEP1 WITH A g' ORACLE. Let g' be a function mapping: $\{0,1\}^k$ to $\{0,1\}^{\sigma(k)}$. Let $\mathcal{E}_{g'}(\alpha, r) = (r, g'(r) \oplus \alpha)$. Let $\{\alpha\}_{g'}$ be the random variable resulting from picking $r \in \{0,1\}^k$ at random and outputting $\mathcal{E}_{g'}(\alpha, r)$. AKEP1$^{g'}$ denotes the protocol in which the parties share a secret a_1 and an oracle for g'. Whenever

AKEP1 asks them to compute $\{\alpha\}_{a_2}$ they compute $\{\alpha\}_{g'}$. The experiment of running E for AKEP1$^{g'}$ is the same as the experiment of running E for AKEP1 except that the second part of the shared key, namely a_2, is absent, and instead the oracles $\Pi^s_{i,j}$ all have access to a common g' oracle and compute their flows according to AKEP1$^{g'}$. E does not have access to g'. When $g' = f'_{a_2}$ for randomly chosen a_2, this experiment coincides with the real experiment.

THE RANDOM AKEP1 EXPERIMENT. In the random AKEP1 experiment we select g' as a random function of $\{0,1\}^k$ to $\{0,1\}^{\sigma(k)}$, and then run the experiment of running E with AKEP1$^{g'}$. As before, let $T_E(k)$ denote a polynomial bound on the number of oracle calls made by E.

Lemma 9. *In the random AKEP1 experiment, advantage$^E(k)$ is negligible.*

Proof: Let $c > 0$ be a constant. We will show that *advantage*$^E(k) \leq k^{-c}$ for all sufficiently large k.

A *view* of E consists of all the oracle queries made by E, the responses to them, and E's own coin tosses; that is precisely what E sees. We denote by *view*(k) the random variable whose value is the view of the interaction of E with its oracles. A particular view will usually be denoted ξ. We will be interested in two properties ξ may possess. If for any accepting oracle there exists an oracle with a matching conversation then we say ξ is *authentic*. If $(r_1, y_1), \ldots, (r_n, y_n)$ denote the encryptions output by oracles in the transcript and $r_1, \ldots, r_n$ are distinct then we say ξ is *non-colliding*. Recall that b denotes the bit flipped in our answer to a test query in the definition of measuring *advantage*$^E(k)$.

Now fix a particular authentic and non-colliding view ξ. Suppose E is pointing to (fresh) oracle $\Pi^s_{A,B}$. Since $\Pi^s_{A,B}$ has accepted and ξ is authentic, there is an oracle $\Pi^t_{B,A}$ which engaged in a matching conversation. This means the encryption for this conversation was selected by one of the oracles (specifically, the one who played the role of the responder). The oracle's being fresh means that any matching partner is unopened. Since ξ is non-colliding it follows that conditioned on *view*$(k) = \xi$, the key $\alpha^s_{A,B}$ is uniformly distributed over S_k, and E's advantage in predicting the bit b is 0.

Let N_k denote the set of non-authentic views and C_k the set of colliding views. We claim that AKEP1$^{g'}$, with g' chosen at random, still remains a secure mutual authentication; the proof of this is analogous to the proof of Theorem 4 and hence is omitted. Based on this claim, we know that the probability of N_k is at most $k^{-c}/2$ for large enough k. On the other hand the probability of C_k is at most $T_E(k)^2 \cdot 2^{-k}$ which is at most $k^{-c}/2$ for large enough k. Combined with the above we conclude that E's advantage is at most k^{-c}. □

See our full paper for the argument that this lemma yields the theorem.