Provably Secure Session Key Distribution— The Three Party Case

Mihir Bellare[*] Phillip Rogaway[†]

Abstract

We study session key distribution in the three-party setting of Needham and Schroeder. (This is the trust model assumed by the popular *Kerberos* authentication system.) Such protocols are basic building blocks for contemporary distributed systems—yet the underlying problem has, up until now, lacked a definition or provably-good solution. One consequence is that incorrect protocols have proliferated. This paper provides the first treatment of this problem in the complexity-theoretic framework of modern cryptography. We present a definition, protocol, and a proof that the protocol satisfies the definition, assuming the (minimal) assumption of a pseudorandom function. When this assumption is appropriately instantiated, our protocols are simple and efficient.

1 Introduction

The main goal of cryptography is to enable secure communication in a hostile environment: two parties, P_i and P_j, want to safely communicate over a network occupied by an active adversary. Usually, P_i and P_j will want to ensure the privacy and authenticity of the data they send to each other; to this end, they will encrypt and authenticate their transmissions. But before P_i and P_j can use these tools they will need to have *keys.* Indeed, without keys, cryptography simply cannot get off the ground.

Much research in modern cryptography has focused on how to *use* keys to provably achieve goals like encryption and signatures (private or public key). There is comparatively little theoretical work on key distribution itself, at least in the realistic model of an active adversary. In particular, central problems in this area still lack definitions and provably-good solutions.

This paper provides provable security for the three-party case of session key distribution. The ideas extend to treat other settings, but the three-party one is the most prevalent in present-day systems. In particular three-party session key distribution is the problem which the well-known *Kerberos* authentication system attempts to solve [20].

[*] IBM T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, New York 10598. e-mail: `mihir@watson.ibm.com`

[†] Dept. of Computer Science, Eng. II Bldg., University of California, Davis, California 95616. e-mail: `rogaway@cs.ucdavis.edu`

1.1 The problem: an informal description

In a distributed system communication between parties typically takes place in "sessions." A session is a relatively short period of interaction between two parties which has an associated "session key" used to protect it. The central notion we aim to capture is that of secure distribution of these session keys.

We let $\{P_1, \ldots, P_N\}$ denote the parties in the distributed system. A key feature of sessions is that a given pair of players, P_i and P_j, may simultaneously maintain multiple sessions (each with its own session key). Thus it is not really P_i and P_j which form the logical endpoints of a secure session; instead, it is an *instance* $\Pi^s_{i,j}$ of P_i and an *instance* $\Pi^t_{j,i}$ of P_j. We emphasize instances as a central aspect of the session key distribution problem, and one of the things that makes session key distribution different from many other problems.

It is the goal of a *session-key distribution protocol* to provide $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ with a session key $\sigma^{s,t}_{i,j}$ to protect their session. Instances $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ must come up with this key without knowledge of s, t, or whatever other instances may currently exist in the distributed system.

An active adversary attacks the network. She controls all the communication among the players: she can deliver messages out of order and to unintended recipients, concoct messages entirely of her own choosing, and start up entirely new instances of players. She may acquire session keys and corrupt players themselves. In the face of such a powerful adversary secure session key distribution is only possible when P_i and P_j have some information advantage over the adversary.

In this paper we realize this information advantage by way of a trusted *Key Distribution Center,* S, with whom each party P_i shares a *long-lived key,* K_i. Because of the involvement of the disinterested party S, this style of session-key distribution is called *three-party key distribution.*[1]

Later sections will formalize the problem appropriately, specifying, in particular, what are desirable properties of session keys and what makes them different from long-lived keys. But we comment that the above paragraphs, while still quite informal, are already at a level of precision beyond the problem's usual explication (in particular due to the consideration of instances).

A reader's first thought might be that it sounds easy to design a protocol to solve this problem. Perhaps the following will help indicate that this is not really so.

[1] Other ways of realizing the trust advantage (not addressed in this paper) are: for each (i, j), parties P_i and P_j might share a long-lived key $K_{i,j}$; or, each P_i might have a public key which all other parties know, and for which P_i holds the secret counterpart.

1.2 A troubled history

The earliest and most influential articulation of the three-party session key distribution problem is by Needham and Schroeder in 1978 [17]. They talk about the use of such protocols and describe a number of candidate ones. In the years following their paper tens of session-key distribution protocols appeared and were implemented. Essentially all of this work has been ad hoc: authors would devise a protocol and then repeatedly try to break and fix it. But Needham and Schroeder had prophetically ended their paper with a warning on this approach, saying that "protocols such as those developed here are prone to extremely subtle errors that are unlikely to be detected in normal operations. The need for techniques to verify the correctness of such protocols is great ...". Evidence of the authors' claim came unexpectedly when a bug was pointed out in their own "Protocol 1" (Denning and Sacco, [8]).[2] Many related protocols were eventually to suffer the same fate. As a result of a long history of such attacks there is finally a general consensus that session key distribution is not a goal adequately addressed by giving a protocol for which the authors can find no attacks.

A large body of work, beginning with Burrows, Abadi and Needham [7], aims to improve on this situation via the use of special-purpose logics. The aim is to demonstrate a lack of "reasoning problems" in a protocol being analyzed. The technique has helped to find errors in various protocols, but a proof that a protocol is "logically correct" does not imply that it is is right (once its abstract cryptographic operations are instantiated). Indeed it is easy to come up with concrete protocols which are logically correct but blatantly wrong. In contrast, we seek methods for which a proof of a protocol's correctness means a great deal more.

1.3 Provable security for three-party session key distribution

In this paper we bring provable security to three-party key distribution. We define the problem, specify a protocol for it, and prove our protocol correct, assuming only the existence of a pseudorandom function family.

Theorem. *If there exists a pseudorandom function family then there exists a secure protocol for three-party session key distribution.*

The formal statement is given by Theorem 3. It is worth pointing out that while there are a great many three-party key distribution protocols in the literature, we know of no earlier one which meets our definition of security.

The ideas needed to define three-party key distribution are different from those underlying familiar notions. A key element of the definition is to develop an appropriate *model.* Among the environmental characteristics which our model captures are the adversary's ability to start up new instances of any of the parties, the possibility that she may gain session keys, and the possibility that she may corrupt players. A second definitional element is the *partner function.*

[2] Insofar as there were no formal statements of what this protocol was supposed to do, it is not entirely fair to call it buggy; but the authors themselves regarded the protocol as having a problem worthy of fixing [18].

Our protocol's starting point, a pseudorandom function family, exists if one-way functions exist [14, 11]. Exploiting techniques of [15] we show (Theorem 7) that the existence of a secure three-party session key distribution implies the existence of a one-way function, and so our assumption is minimal.

1.4 Design for practice

Provably-secure protocols are not usually efficient. Ours is an exception to this rule. The protocol presented in this paper is efficient in terms of rounds, communication, and computation. Our protocol is actually simpler than previous ones in the literature. Earlier solutions, lacking definitions and proofs, often encumbered their protocols with irrelevant features. Related protocols designed with the same methodology retain these advantages.

The above efficiency was *designed* into our protocols in part through the choice of the underlying primitive—namely, a pseudorandom function. In practice, 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.

1.5 Related work

The most closely related work to the present one is a previous paper of ours on two-party entity authentication and authenticated key exchange [2], which provided the first provably secure solutions for these goals. This in turn built on the work of [3, 10] for the same problem. The work of [2] was the first to formalize the notion of instances, and our model, with the framework of an adversary talking to its oracles, extends that of [2]. However, the goals treated here and in [2] are very different. Entity authentication is the process by which parties can become convinced that they are talking to one another. Now on top of such a protocol one can often piggyback a two-party key distribution, but, fundamentally, entity authentication is simply irrelevant to key distribution (and conversely). Furthermore, the two-party entity authentication goal —with or without a key distribution— is of no obvious utility in the three-party setting of this paper.

Many protocols aim to distribute a key whose value depends only on the initially distributed set of long-lived keys and on the identities of those who want to have the shared key. Such key distributions can be non-interactive and information-theoretically secure. This approach begins with [5]; additional work includes [19, 6, 1, 16]. In these works the model does not recognize multiple instances of players: in effect, they distribute long-lived keys given long-lived keys stored in a more convenient manner.

The classic secret key exchange problem [9] is again entirely different: the adversary there is passive, and there is no notion of sessions or instances.

It is impossible to survey here the large body of suggested protocols for three-party session key distribution. One contemporary solution which influenced our thinking is IBM's *KryptoKnight* family of protocols [4]. These avoid many earlier pitfalls, but they still fall short of our definition of correctness.

1.6 Authentication versus key distribution

The tendency to link the entity authentication goal and the key distribution one is so entrenched that it is worth emphasizing that these problems are very different. As explained, entity authentication is about becoming convinced that you have been talking to an intended parter; key distribution is about getting a key to at most the two of you. One of our contributions is to disentangle these problems and identify one of them —key distribution— as the "right" goal for most applications. We comment that a definition for three-party entity authentication can be obtained by modifying the notion of two-party entity authentication given in [2] (if desired, this definition can then be extended to demand a key be distributed in the process, mimicking the treatment of authenticated key exchange [2]). But we believe that the above is not the best approach. The reason is that entity authentication is rarely useful in the absence of an associated key distribution, while key distribution, all by itself, is not only useful, but it is not appreciably more so when an entity authentication occurs along side. Most of the time the entity authentication is *irrelevant*: it just doesn't matter if you have been speaking to a given communication partner, in that by the time you become aware of this fact there will be no particular reason to believe that that partner is still "out there," anyway. All that matter is to securely distribute a key, and then use that key to protect the ensuing session.

2 Preliminaries

Notation. The distributed system we model has N *players* whom we give *identities* $I = \{1, \ldots, N\}$. The number N may be any polynomial function of the *security parameter, k*. Elements of I will be denoted by i, j, or sometimes by A, B. In addition there is a *key distribution center, S*— it is not a member of I, and is assigned identity 0. The adversary E is not any of the above entities; how she is modeled will be specified shortly.

If A is a probabilistic algorithm then $A(x, y, \cdots)$ denotes the probability space which to σ assigns the probability that A, on inputs $x, y, \cdots$, outputs σ.

We let $\{0,1\}^*$ denote the set of finite binary strings and $\{0,1\}^\omega$ the set of infinite ones. The empty string is written λ. When a, b, c, ... are strings, by $a \,.\, b \,.\, c \,.\, \cdots$ we denote any natural encoding of these strings such that each constituent string is efficiently recoverable from $a \,.\, b \,.\, c \,.\, \cdots$ in the context of its receipt. A function is *efficiently computable* if it can be computed in time polynomial in its first argument. We let "PPT" stand for "probabilistic polynomial time." A family of distributions $\{D_k\}_{k \in \mathbb{N}}$ is called *samplable* if there exists a PPT algorithm D, called a *generator*, such that $D(1^k)$ is the distribution D_k. We won't again bother to distinguish a samplable family of distributions from its generator. 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 $k > k_c$. A function is *nonnegligible* if it is not negligible. We write $a \leftarrow A$ for choosing a random sample from the finite set A; we write $s \leftarrow \{0,1\}^\omega$ for choosing an infinite string s with each bit equiprobably zero or one.

Protocols. We start off by specifying the "syntax" of a key distribution protocol. A *three-party key distribution protocol* (or simply a *protocol* in this paper) is a triple $P = (\Pi, \Psi, LL)$ of polynomial-time computable functions:

Π	—	specifies how (honest) players behave;
Ψ	—	specifies how S behaves; and
LL	—	specifies the (initial) distribution on the long-lived keys.

The domain and range of these functions is as follows. The function Π takes as input the following six arguments:

1^k	—	security parameter
$i \in I$	—	identity of sender
$j \in I$	—	identity of (intended) partner
$K_i \in \{0,1\}^*$	—	long-lived key of i
$conv \in \{0,1\}^*$	—	conversation so far
$r \in \{0,1\}^\omega$	—	random coin flips

while the value $(m, \delta, \sigma) = \Pi(1^k, i, j, K_{i,j}, conv, r)$ returned by Π consists of:

$m \in (\{0,1\}^* \cup \{*\})^2$	—	messages sent to S & j
$\delta \in \{\mathsf{A}, \mathsf{R}, *\}$	—	the decision
$\sigma \in \{0,1\}^* \cup \{*\}$	—	the private output

We require that the private output be string-valued when and only when the decision is A. The private output is called a *session key* when it is string-valued. We let $\Pi^{m\delta}(\cdot)$ denote the first two components of $\Pi(\cdot)$, $\Pi^{\delta}(\cdot)$ the second component of $\Pi(\cdot)$, and $\Pi^{\sigma}(\cdot)$ the third component of $\Pi(\cdot)$.

The function Ψ takes as inputs the following seven arguments:

1^k	—	security parameter
$i \in I$	—	identity of the initiator
$j \in I$	—	identity of the responder
$K_i \in \{0,1\}^*$	—	long-lived key of i
$K_j \in \{0,1\}^*$	—	long-lived key of j
$conv \in \{0,1\}^*$	—	the conversation so far
$r \in \{0,1\}^\omega$	—	random coin flips

while $m = (m_i, m_j) = \Psi(1^k, i, j, K_i, K_j, conv, r)$ has components:

$m_i \in \{0,1\}^* \cup \{*\}$	—	message to i
$m_j \in \{0,1\}^* \cup \{*\}$	—	message to j

Finally, the function LL takes as input:

1^k	—	the security parameter
$r \in \{0,1\}^\omega$	—	random coin flips

and the value returned by $LL(1^k, r)$ is a string $K \in \{0,1\}^*$. Such a string is called a "long-lived key" (LL-key).

Discussion. Let $(m, \delta, \sigma) = \Pi(1^k, i, j, k_{i,j}, conv, r)$. Then the first component of m is to be thought of as the string that i tries to send to S. The second component of m is what i tries to send to j. Either or both of these may be $*$, indicating that i sends no message to that entity. Similarly, S is permitted to send messages to two different players in a single round. We use the same $*$-convention to indicate S does not want to send a message to either or both player.

The above communication conventions are irrelevant to our notion of security; for example, one can always turn a protocol P into a protocol P' which sends only one message in each round and which meets our definition of security as long as P does.

The function δ is the *decision predicate.* Its values A, R and $*$ are supposed to suggest "accept," "reject," and "no decision yet reached," respectively. The string *conv* will be used to record the conversation so far; this will grow as the conversation progresses, in a manner we are about to describe.

3 The Model

We now formulate a communication model appropriate for defining authentication and key distribution goals in the distributed environment. The situation we address is one where communication between players is entirely controlled by the adversary. We build on and extend the model of [2].

The adversary is a probabilistic machine E which we think of as being equipped with an infinite collection of oracles— $\Pi^s_{i,j}$ and $\Psi^s_{i,j}$, for $i, j \in I$ and $s \in \mathsf{N}$. Oracle $\Pi^s_{i,j}$ models instance s of player i attempting to agree on a shared session key with player j. Oracle $\Psi^s_{i,j}$ models instance s of S in its role of trying to help distribute a key to i and j. Attacks that E can can mount are modeled via oracle queries which E writes on a special tape and to which she gets a response in unit time. We will first formally describe the kinds of queries and how they are answered; then we discuss the intuition behind each type of query. (We comment that oracles maintain state between queries, contrary to typical usage of this term.)

Running The Protocol. To execute $P = (\Pi, \Psi, LL)$ using adversary E and security parameter $k \in \mathsf{N}$, one begins by making the following initializations, for each $(i, j, s) \in I \times I \times \mathsf{N}$: $K_i \leftarrow LL(1^k)$, $r_E \leftarrow \{0,1\}^\omega$, $\not c\mathrm{oins}\Pi^s_{i,j} \leftarrow \{0,1\}^\omega$, $\not c\mathrm{oins}\Psi^s_{i,j} \leftarrow \{0,1\}^\omega$, $\mathrm{conv}\Pi^s_{i,j} \leftarrow \lambda$, and $\mathrm{conv}\Psi^s_{i,j} \leftarrow \lambda$. The experiment proceeds by running E on input $(1^k, r_E)$. Adversary E may make oracle queries, to be answered as described in Figure 1. For now, the reader should ignore query type 5; this query will be explained in Section 4. The remaining four queries are explained here.

Referring to Figure 1, queries not of the correct syntactic form are answered by λ. Referring to query type 4, by $\langle K_i, \not c\mathrm{oins}\Pi^s_{i,j}, \mathrm{conv}\Pi^s_{i,j} \rangle_{j,s}$ we mean an encoding of the strings K_i and $\not c\mathrm{oins}\Pi^s_{i,j}$ and $\mathrm{conv}\Pi^s_{i,j}$ where $(j, s) \in I \times \mathsf{N}$ and there has already been a $(\mathsf{SendPlayer}, i, j, s, \cdot)$ query made by E.[3]

Explanation. Let us describe the intent behind query types 1–4. When E writes a query $(\mathsf{SendPlayer}, i, j, s, x)$ or $(\mathsf{Reveal}, i, j, s)$ on its query tape, we think of that query as being answered by the oracle $\Pi^s_{i,j}$. When E writes a query $(\mathsf{SendS}, i, j, s, x)$ on its query tape, we think of that query as being answered by $\Psi^s_{i,j}$.

Under the defined model, all communication is under the adversary's control. She tells instances (oracles) when to start; obtains their transmissions; and delivers them as she chooses to other instances. She delivers messages out of order and to unintended recipients, concocts messages of her own choosing, and creates as many sessions (new instances) as she pleases. All these abilities are captured by the queries $(\mathsf{SendPlayer}, i, j, s, x)$ and $(\mathsf{SendS}, i, j, s, x)$. These model the adversary sending messages to individual players and to the key distribution center. What the adversary gets back from her query is the response which that entity would generate. Additionally (see row 1 of Figure 1), when making a $\mathsf{SendPlayer}$ query the adversary gets back an $\mathsf{A}/\mathsf{R}/*$ in order that she can "see" when an oracle accepts. It would be unrealistic not to provide the adversary this capability.

If an oracle $\Pi^s_{i,j}$ accepts it will have "inside it" a (private) session key, $\sigma^s_{i,j}$. We allow the adversary to expose this key with a $(\mathsf{Reveal}, i, j, s)$ query. What does this model? One of the main purposes of session keys is that the loss of one should only compromise the session which that key protects [8, 21]. (Indeed, failure to achieve security in the face of such a loss of session keys is the basis of the well-known "bug" of [17].) The session using the key will be using it for some purpose about which we known nothing. For example, a protocol may use the session key for message authentication, and such a protocol would not be wrong to reveal the session key when the session is over and the key is no longer useful for the adversary to forge messages of this session. Since we do not know that the key will *not* be revealed in the protocol which uses it, we must model this eventuality and let the adversary learn selected session keys.

A more severe type of loss is when a player's complete internal state becomes known to the adversary. This can happen (in the real world) when E breaks open the "secure boundary" which protects a player. It also happens when the player actually *is* the adversary, or is working for her. To model this possibility we allow a $(\mathsf{Corrupt}, i, K)$-query, from which the adversary learns the internal state of player i, and also substitutes some value K of her choice for the player's long-lived key K_i. From that point on, S will use the revised LL-key. If E later wants to modify i's LL-key to some new one K' she can issue a $(\mathsf{Corrupt}, i, K')$-query (ignoring the returned string). Because of these queries a protocol will be not be considered secure if, for example, an (uncorrupted) player j leaks K_j if he happens to engage in a conversation with some player i who has some cleverly chosen K_i. This capability captures possibilities that may exist for the adversary in the real world.

Notation. Adopting the oracle language from the above discussion, we will say that $\Pi^s_{i,j}$ has *accepted* if $\Pi^\delta(1^k, i, j, K_i, \mathrm{conv}\Pi^s_{i,j}, \not c\mathrm{oins}\Pi^s_{i,j}) = \mathsf{A}$; that it is *opened* if there has been a $(\mathsf{Reveal}, i, j, s)$ query; and that it is *unopened* if it is not opened. We say that i has been *corrupted* if there has been a $(\mathsf{Corrupt}, i, \cdot)$ query, and it is *uncorrupted* otherwise.

To exercise the new language: as we have defined an execution, all the oracles start off unopened. Initially no oracle holds a session key. When an oracle accepts it has a session key. The adversary could learn this session key in a couple ways. She could make a reveal query —which is an attack on a particular instance $\Pi^s_{i,j}$— and that would result in her getting back just the session key of the indicated oracle. Or she could make a corrupt query —which is an attack directed against *all* instances $\Pi^t_{i,j}$ of a player i— resulting in her getting back all of the state of all sessions of player i. From this state the adversary can calculate the session keys of all instances of i which hold session keys.

[3] As a minor detail, the string $\not c\mathrm{oins}^s_{i,j}$ is infinite, and so the encoding should include only the "used" portion of coins, or else it should be constructed to give efficient access to each bit of the constituent strings.

	On query of:	Return:	And then set:
1	$(\mathsf{SendPlayer}, i, j, s, x)$	$\Pi^{m\delta}(1^k, i, j, K_i, \mathrm{conv}\Pi^s_{i,j}, \not{c}\mathrm{oins}\Pi^s_{i,j})$	$\mathrm{conv}\Pi^s_{i,j} \leftarrow \mathrm{conv}\Pi^s_{i,j} \,.\, x \,.\, m$
2	$(\mathsf{SendS}, i, j, s, x)$	$\Psi(1^k, i, j, K_i, K_j, \mathrm{conv}\Psi^s_{i,j})$	$\mathrm{conv}\Psi^s_{i,j} \leftarrow \mathrm{conv}\Psi^s_{i,j} \,.\, x \,.\, m$
3	$(\mathsf{Reveal}, i, j, s)$	$\Pi^{\sigma}(1^k, i, j, K_i, \mathrm{conv}\Pi^s_{i,j}, \not{c}\mathrm{oins}\Pi^s_{i,j})$	
4	$(\mathsf{Corrupt}, i, K)$	$\langle K_i, \not{c}\mathrm{oins}\Pi^s_{i,j}, \mathrm{conv}\Pi^s_{i,j}\rangle_{j,s}$	$K_i \leftarrow K$
5	(Test, i, j, s)	Choose at random a bit b. If $b = 0$ return $\sigma \leftarrow Sn(1^k)$; if $b = 1$ return $\sigma \leftarrow \Pi^{\sigma}(1^k, i, j, K_i, \mathrm{conv}\Pi^s_{i,j}, \not{c}\mathrm{oins}\Pi^s_{i,j})$	

Figure 1: The queries which E can ask of its oracles.

Benign Adversaries. There is no communication in our model without an adversary. We model reliable communication by considering a *benign* adversary, one who faithfully relays flows just like a reliable channel. Specifically, to every (i, j, s, t, u) we associate an adversary called the (i, j, s, t, u)-*benign* adversary which is deterministic and confines her actions to faithfully conveying flows between $(\Pi^s_{i,j}, \Pi^t_{j,i}, \Psi^u_{i,j})$, beginning with $\Pi^s_{i,j}$. Details are omitted.

Transcript. We need some specialized language to talk about the manifest communication in a network. The reveal and $\mathsf{corrupt}$ queries do not correspond to communication among players, but the $\mathsf{SendPlayer}$ and SendS queries do. Thus we define a *communication record* as a pair $\langle q, y\rangle$, where q is a string $(\mathsf{SendPlayer}, i, j, s, x)$ or $(\mathsf{SendS}, i, j, s, x)$, and y is an arbitrary string. A *transcript* is a sequence of communication records. So far, this is just a syntactic notion. The set of all well-formed (syntactically-correct) transcripts is denoted $\mathcal{T}$. Note that $\mathcal{T}$ depends neither on the protocol nor on the adversary. In a given execution of E running P with security parameter k, there will be a particular transcript $T \in \mathcal{T}$ at the end of this execution, where the i-th communication record in T is the i-th $\mathsf{SendPlayer}$ or SendS query together with its response.

A Single Model For Many Goals. We have not yet defined any particular goal; we have only specified the adversarial model in which some goal might be formulated. Indeed, this same model can be used for many authentication and key-distribution goals.

4 Definition of Security

To be considered secure a key distribution protocol must protect the session keys which it distributes. But it must do something more: the protocol better distribute those keys! For example, the protocol which does *nothing* distributes all of its keys quite securely—but it is not a reasonable mechanism. To eliminate such nonsense we start off by demanding a basic "validity" property of any key distribution protocol.

4.1 Validity

To be a reasonable key distribution protocol the requirement is made that when communication channels are reliable, parties i and j, executing the protocol with S, should end up with a common key; moreover, this key should be distributed according to the desired distribution on session keys. More formally, a protocol $P = (\Pi, \Psi, LL)$ is called *valid* if for all $i, j \in I$ and $s, t, u \in \mathsf{N}$, when the protocol is run using the (i, j, s, t, u)-benign adversary, oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ always accept, and each outputs the *same* session key. When Sn is a generator we say that P is Sn-*valid* if these keys are $Sn(1^k)$-distributed when the security parameter is k. We emphasize that our notion of security depends on Sn; we will be defining a notion that "a protocol P securely distributes Sn-distributed keys when ..."

4.2 Protecting fresh session keys

We might like to formalize that, for a key distribution protocol to be good, the adversary must be unable to learn anything about the distributed session keys. But this is too much to hope for: after all, the type of adversary we have modeled can learn a session key just by issuing an appropriate query. The best we can hope for is that the adversary doesn't learn anything about a session key *unless* she explicitly asks for it. But asked *whom* (what oracle)? Certainly if the adversary asks $\Pi^s_{i,j}$ for its session key then the adversary learns the session key of $\Pi^s_{i,j}$. But a key distribution protocols is *supposed* to distribute keys, so presumably there may exist some *other* oracle $\Pi^t_{j,i}$ the loss of whose session key compromises that of $\Pi^s_{i,j}$. It is this *partner* of $\Pi^s_{i,j}$ which we now turn our attention to defining.

The Partner Function. Intuitively, the partner of an oracle is that other oracle with which it shares a key. Not every oracle will have a partner: for example, an oracle which never comes to hold a session key will never get a partner. In fact, even an oracle that does come to have a session key might not have a partner. To see this, consider the adversary who runs a protocol faithfully, except that she delivers messages one at a time and halts as soon as some oracle accepts. This (accepting) oracle holds a session key, but it can't have a partner (an oracle which who holds a matching key) since no other oracle holds a key. Perhaps the accepting oracle *would* get a partner if one more message was properly delivered—but this event may never happen.

It is convenient to define the partner of an oracle in a somewhat "syntactic" way. Roughly said, a *partner function* is a map f which (on a given execution) for each oracle $\Pi^s_{i,j}$ "points" to its partner (if any) $\Pi^t_{j,i}$. Formally, a partner function is a polynomial-time map $f : \mathcal{T} \times I \times I \times \mathsf{N} \to \mathsf{N} \times \{*\}$. We write $f^s_{i,j}(T)$ instead of $f(T, i, j, s)$.

Fix a protocol P, a partner function f, an adversary E, and a security parameter k. Run E. Let it terminate with some transcript, T. If $f^s_{i,j}(T) = t \in \mathsf{N}$ then we shall call oracle $\Pi^t_{j,i}$ the *partner* to $\Pi^s_{i,j}$. If $f^s_{i,j}(T) = *$ we say that $\Pi^s_{i,j}$ is unpartnered.

Note that the partner of an oracle $\Pi^s_{i,j}$ is an oracle of the form $\Pi^t_{j,i}$. It is unacceptable that an oracle representing an instance of i trying to come to a session key with an instance of j should be partnered with an oracle other than an instance of j trying to come to a session key with an instance of i.

We comment that our definition of a partner function ensures that the adversary "knows" which is the partner of any oracle. While one might have enlarged the domain of f in various ways, it is important to preserve this condition. Since a protocol will be deemed secure only if there exists a partner function with a certain property, demanding that the partner function have a particularly simple form, as we have done, only strengthens our notion of security.

We note that this definition of a partner function is less stringent than the idea of partnering by matching conversations given in [2]. In the latter, the partner of an oracle is identified by looking at the bits which flow among the oracles. Such a notion is not possible here, since we wish to define key distribution independently of authentication.

FRESHNESS. Fix a protocol P, a partner function f, an adversary E, and a security parameter k. Run E. Let it terminate with some transcript, T. Now ask: which are the oracles which hold a session key about which the adversary should not know? Our answer is given in the following definition:

Definition 1 *Oracle* $\Pi^s_{i,j}$ holds a fresh session key *(or* $\Pi^s_{i,j}$ *is* fresh*) if, at the end of the execution, the following are true:*

- $\Pi^s_{i,j}$ *has accepted;*
- $\Pi^s_{i,j}$ *is unopened;*
- $\Pi^s_{i,j}$*'s partner (if it has one) is unopened; and*
- *Players i and j are uncorrupted.*

Fresh oracles correspond to those for which the adversary cannot know the contained session key by "trivial" means. If an oracle holds a fresh session key, we need to protect it. Recall for the above definition that "unopened" and "uncorrupted" are technical terms, defined in Section 3. We emphasize that freshness depends on the partner function f.

TEST QUERIES. Now we are ready to formalize the secrecy of fresh keys. We do this along the lines of the definition of polynomial security of encryption [12]. We emphasize that it is *not* enough to say that the adversary doesn't know fresh session keys; we expect all properties of these keys to be well-hidden, too.

Fix a generator Sn and a partner function f. We augment the experiment of running P with E on security parameter k as follows. There will be one more initialization: $b \leftarrow \{0,1\}$, and there will be one new type of query, (Test, i, j, s). This query must be adversary's last query, and it must be asked of a fresh oracle, $\Pi^s_{i,j}$. The query is answered as specified in Figure 1, query type 5: depending on the value of b, we either return a random point drawn from $Sn(1^k)$, or else we return the session key which the adversary is indicating. The adversary will be trying to guess which of these two possibilities just happened.

After making final (Test) query the adversary outputs a bit, *guess*, which we think of as her guess of the bit b. Let Good-Guess be the event that $guess = b$. The quantity we are interested in is the advantage E has over random guessing. We define this advantage by

$$\mathrm{ADV}^{E}_{P,f,Sn}(k) = 2 \cdot \Pr[\mathsf{Good\text{-}Guess}] - 1.$$

The probability is over everything in the experiment we have described: the random coins of the key generator, the adversary, the $\Pi^s_{i,j}$ oracles, the $\Psi^s_{i,j}$ oracles, and the bit b. The advantage has been scaled to be in the range $[-1, 1]$. This scaling is only significant in an "exact" treatment of security; here we give an asymptotic treatment.

4.3 Main definition

Fixing a particular partner function f induces a notion of which keys will be considered fresh. We say a protocol protects session keys if there is some partner function under which no polynomial time adversary can succeed in saying something intelligent fresh session keys.

Definition 2 *Let Sn be a generator. A protocol P is a* secure three-party key distribution*, distributing Sn-distributed keys, if it is Sn-valid and there is a partner function f such that for every PPT adversary E,* $\mathrm{ADV}^{E}_{P,f,Sn}(\cdot)$ *is negligible.*

When Sn is clear or irrelevant, we omit mention of it. If P is a protocol and f is a partner function as above, we will say that f *witnesses* the security of P.

REMARK. The degree of protection of the distributed session key is very strong. Many protocols in the literature fail to meet this definition because, for example, the distributed session key is used to encrypt or authenticate certain flows of the key distribution protocol itself. Not only will a protocol P^* which makes such "premature" use of the distributed session key fail to meet our definition, but, in fact, it will be possible to construct a protocol Q which is secure following an "ideal" session key distribution but which is insecure following P^*.

5 A secure key distribution protocol

BASIC TOOLS. We begin with some basic tools. A *message authentication scheme* (private key signature scheme) is a pair of polynomial-time algorithms $(\mathsf{MAC}, \mathsf{VF})$, the first of which may be probabilistic. The function MAC takes a message x, a key a, and random coins r, and it produces a "message authentication code" (tag) $\mathsf{MAC}_a(x, r)$. The function VF takes a message x and a key a and it returns a bit $\mathsf{VF}_a(x)$, with 1 standing for accept and 0 for reject. We require that for any μ output with positive probability by $\mathsf{MAC}_a(x)$, it is the case that $\mathsf{VF}_a(x, \mu) = 1$.

For security we adapt the definition of [13]. An adversary E for a MAC scheme is an algorithm with a $\mathsf{MAC}_a(\cdot)$ oracle and a $\mathsf{VF}_a(\cdot)$ oracle. Define $success_E(k)$ as the probability that $E^{\mathsf{MAC}_a(\cdot),\mathsf{VF}_a(\cdot)}(1^k, r_E)$ finds an x^* for which $\mathsf{VF}_a(x^*) = 1$ for x^* unasked of $\mathsf{MAC}_a(\cdot)$. The probability is taken over $a \leftarrow \{0,1\}^k$, $r_E \leftarrow \{0,1\}^\omega$, and $r_1, r_2, \ldots \leftarrow \{0,1\}^\omega$, and then answering the i-th MAC query as follows: if the question was x, return $\mathsf{MAC}_a(x, r_i)$. The MAC scheme is *secure* if for every polynomial-time adversary E, $success_E(k)$ is negligible.

A pseudorandom function family $\{f_a(x)\}$ makes a good MAC: define $\mathsf{MAC}_a(x) = f_a(x)$ and define $\mathsf{VF}(x, \mu)$ by the predicate $(\mu = \mathsf{MAC}_a(x))$ [11].

A (private key) *encryption scheme* is a pair of polynomial-time functions $(\mathcal{E}_a(x), \mathcal{D}_a(y))$, the first of which is probabilistic and the second of which is deterministic. We may explicitly indicate the coins of $\mathcal{E}$ by writing $\mathcal{E}_a(x, r)$. We require that $\mathcal{D}(y) = x$ for every y output with positive probability by $\mathcal{E}_a(x)$.

For security we adapt the definition of [12]. For any adversary E and generator G we consider the experiment defined as follows. Choose $a \leftarrow \{0,1\}^k$, $\theta \leftarrow \{0,1\}$, $\sigma_0 \leftarrow G(1^k)$, $\sigma_1 \leftarrow G(1^k)$, and $r_E \leftarrow \{0,1\}^\omega$. Sample $\alpha \leftarrow \mathcal{E}_a(\sigma_\theta)$ and run $E^{\mathcal{E}_a(\cdot)}(\sigma_0, \sigma_1, \alpha, r_E)$ using a (probabilistic) oracle for $\mathcal{E}_a(\cdot)$. We say that adversary E *succeeds* if she output θ. The encryption scheme is *secure* if for every generator G and every polynomial time adversary E, the adversary succeeds with probability $\leq 0.5 + \epsilon(k)$ for some negligible function ϵ.

Secure (symmetric) encryption is straightforward given a pseudorandom function family: simply define $\mathcal{E}_a(x, r) = (r, f_a(r) \oplus x)$.

THE PROTOCOL **3PKD**. For any generator Sn, any encryption scheme $(\mathcal{E}, \mathcal{D})$, and any message authentication scheme $(\mathsf{MAC}, \mathsf{VF})$, we specify a key distribution protocol $P = \mathbf{3PKD}[(\mathsf{MAC}, \mathsf{VF}),\ (\mathcal{E}, \mathcal{D}),\ Sn]$. The long-lived key generator, $LL(1^k, r)$, returns a uniformly distributed $2k$-bit string K_i which we view as a pair (K_i^{enc}, K_i^{mac}) of keys for encrypting and signing, respectively. We assume that each player $i \in I$ is associated to a unique k-bit string, which we denote by i itself. Instructions for the players and for S are described in a pictorial form in Figure 2. We describe the meaning of this figure below.

In Step 1, party A chooses a random k-bit challenge R_A and sends it to B. In Step 2, party B chooses a random k-bit challenge R_B and sends $R_A \,.\, R_B$ to S. In Step 3, S runs generator $Sn(1^k)$ to get a session key σ which he will distribute. Then, using coins r_A (resp. r_B) selected at random, S probabilistically encrypts σ under A's encryption key K_A^{enc} (resp. B's encryption key K_B^{enc}) to get ciphertext α_A (resp. α_B). Then S computes μ_A (resp. μ_B), the MAC under key K_A^{mac} (resp. K_B^{mac}) of the string $A \,.\, B \,.\, R_A \,.\, \alpha_A$ (resp. $A \,.\, B \,.\, R_B \,.\, \alpha_B$). (Coins are flipped if the MAC requires this.) Here A and B denote the k-bit names of these players. In flow 3A (resp. 3B) S sends A (resp. B) the message $\alpha_A \,.\, \mu_A$ (resp. $\alpha_B \,.\, \mu_B$). In Step 4A (resp. 4B) Party A (resp. B) receives a message $\alpha'_A \,.\, \mu'_A$ (resp. $\alpha'_B \,.\, \mu'_B$) and accepts, with session key $\mathcal{D}_{K_A^{enc}}(\alpha'_A)$ (resp. $\mathcal{D}_{K_B^{enc}}(\alpha'_B)$), if and only if $\mathsf{VF}_{K_A^{mac}}(A \,.\, B \,.\, R_A \,.\, \alpha'_A, \mu'_A) = 1$ (resp. $\mathsf{VF}_{K_B^{mac}}(A \,.\, B \,.\, R_B \,.\, \alpha'_B, \mu'_A) = 1$). This completes the protocol description. The formal definitions of Π and Ψ are readily constructed from this.

MAIN THEOREM. Our main result is given by the following theorems.

Theorem 3 Let $(\mathsf{MAC}, \mathsf{VF})$ be a secure message authentication scheme, let $(\mathcal{E}, \mathcal{D})$ be a secure encryption scheme, and let Sn be a generator. Then protocol $\mathbf{3PKD}[(\mathsf{MAC}, \mathsf{VF}),\ (\mathcal{E}, \mathcal{D}),\ Sn]$ is a secure key distribution protocol, distributing Sn-distributed keys.

The proof appears in Appendix A. Using well-known results [14, 11] we obtain the following corollary.

Corollary 4 If there exists a one-way function then for any generator Sn there exists a secure key distribution protocol, distributing Sn-distributed keys.

COMMENTS. The specifics of our formulation have made it unnecessary for A to send her name in flow 1, or for B to send the names A and B in flow 2. In a "real" protocol, these "hints" would normally accompany those flows.

In a real protocol, one might prefer to employ only $A \leftrightarrow B \leftrightarrow S$ connectivity; in this case flow 3A should be "routed through" B. (In other words, the protocol **3PKD** is changed so that S sends only one message, to B, and B forwards half of this message along to A.) Alternatively, one might prefer $S \leftrightarrow A \leftrightarrow B$ connectivity, as in [17, 20]); in this case the protocol **3PKD** is changed so that messages are routed as $A \rightarrow B \rightarrow A \rightarrow S \rightarrow A \rightarrow B$. Changes in a protocol's message routing (to accommodate a desired connectivity graph) do not impact a protocol's provable security in any way. We thus suggest that protocols for alternative connectivity models ought differ only in their message routing.

Regardless of the routing variant one assumes, protocols in the literature for (timestamp-less) three-party session key distribution typically have an extra flow or when two compared to the same-routed version of **3PKD**. The extra communication was usually intended for entity authentication. Our protocol has fewer flows simply because it does not have entity authentication as a goal. See Section 1.6 for why we have made this choice. Also see the remark at the end of Section 4.3 to see why any additional flows for entity authentication should *not* be a challenge-response based on the distributed session key.

6 Basic properties

This section describes basic properties of a secure key distribution protocol. Proofs are omitted.

STRUCTURE OF THE PARTNER FUNCTION. One expects a number of characteristics for the partner function which were not explicitly demanded in its definition. The following theorem collects up some such properties which may always be assumed without loss of generality.

Theorem 5 Let P be a secure protocol. Then there exists a partner function f which witnesses this and which has the following additional properties:

(1) If $T_0, T \in \mathcal{T}$ with T_0 a prefix of T then $f_{i,j}^s(T_0) \in \{*, f_{i,j}^s(T)\}$.

(2) Suppose $f_{i,j}^s(T) = t$ and let $T_0 \in \mathcal{T}$ be one record longer than the longest prefix of T for which $f_{i,j}^s = *$. Then the last record $\langle q, y \rangle$ of T_0 is a Send query whose response indicates that $\Pi_{i,j}^s$ has accepted.

(3) Suppose $f_{i,j}^s(T) = t$. Then $f_{j,i}^t(T) \in \{*, s\}$.

The first condition says that an oracle's partner becomes defined at most once. The second condition says this can only happen right after an oracle accepts. The final condition says that if $\Pi_{i,j}^s$ is partnered to $\Pi_{j,i}^t$, then if $\Pi_{j,i}^t$ is partnered to anybody, that anybody is $\Pi_{i,j}^s$.

UNIQUENESS OF PARTNER. If both f and $\hat{f}$ are partner functions for a secure key distribution protocol P, then it seems as though f and $\hat{f}$ ought to be "essentially" the same. In particular, whenever one partner function names an oracle the other partner function should name the same oracle, if it names any oracle at all.

Theorem 6 Let P be a secure protocol and let f and $\hat{f}$ be partner functions which witness this. Let Tr be the random variable which returns the final transcript. Let $\text{DIFFERENTPARTNERS}(k)$ be the event there exists an $i, j \in I$, $s \in \mathsf{N}$, such that $f_{i,j}^s(\mathsf{Tr}) \neq *$ and $\hat{f}_{i,j}^s(\mathsf{Tr}) \neq *$ and $f_{i,j}^s(\mathsf{Tr}) \neq$

Flow 1.	$A \rightarrow B$:	R_A
Flow 2.	$B \rightarrow S$:	$R_A \,.\, R_B$
Flow 3A.	$S \rightarrow A$:	$\mathcal{E}_{K_A^{enc}}(\sigma, r_A)\ .\ \mathsf{MAC}_{K_A^{mac}}(A \,.\, B \,.\, R_A \,.\, \mathcal{E}_{K_A^{enc}}(\sigma, r_A))$
Flow 3B.	$S \rightarrow B$:	$\mathcal{E}_{K_B^{enc}}(\sigma, r_B)\ .\ \mathsf{MAC}_{K_B^{mac}}(A \,.\, B \,.\, R_B \,.\, \mathcal{E}_{K_B^{enc}}(\sigma, r_B))$

Figure 2: A terse representation of protocol **3PKD**. For a fuller description, see the accompanying text.

$\hat{f}^s_{i,j}(\mathsf{Tr})$. Then $\epsilon(k) = \Pr[\textsc{DifferentPartners}(k)]$ is negligible.

Assumption Minimality. For D a distribution on strings let h_D be the probability mass of the most probable string in D. A family of distributions D_k is called *non-degenerate* if for some $c > 0$ we have that $1 - h_{D_k} > k^{-c}$ for all large enough k. The following theorem can be established using techniques of [15].

Theorem 7 Suppose there exists a secure key distribution protocol which distributes $Sn(\cdot)$-distributed keys, where Sn is non-degenerate. Then there exists a one-way function.

References

[1] A. Beimel and B. Chor, "Interaction in key distribution schemes," Crypto 93.

[2] M. Bellare and P. Rogaway, "Entity authentication and key distribution," Crypto 93.

[3] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M. Yung, "Systematic design of two-party authentication protocols," Crypto 91.

[4] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva and M. Yung, "The KryptoKnight family of light-weight protocols for authentication and key distribution," *IEEE/ACM T. on Networking*, 3(1), February 1995.

[5] R. Blom, "An optimal class of symmetric key generation systems," Eurocrypt 84.

[6] C. Blundo, A. De Santis, A. Herzberg, S. Kutten and M. Yung, "Perfectly-secure key distribution for dynamic conferences," Crypto 92.

[7] M. Burrows, M. Abadi and R. Needham, "A logic for authentication," *ACM Transactions on computer systems*, Vol. 8, No. 1.

[8] D. Denning and G. Sacco, "Timestamps in key distribution protocols," *Communications of the ACM*, Vol. 24, No. 8, pp. 533–536, 1981.

[9] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Info. Theory* IT-22, 644-654 (November 1976).

[10] W. Diffie, P. van Oorschot and M. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, 2, 107–125 (1992).

[11] O. Goldreich, S. Goldwasser and S. Micali, "How to construct random functions," *Journal of the ACM*, Vol. 33, No. 4, 210–217, (1986).

[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, R. Impagliazzo, L. Levin, and M. Luby, "Construction of a pseudo-random generator from any one-way function." Manuscript. Earlier versions in STOC 89, STOC 90.

[15] R. Impagliazzo and M. Luby, "One-way functions are essential for complexity based cryptography," FOCS 89.

[16] T. Leighton and S. Micali, "Secret-key agreement without public-key cryptography," Crypto 93.

[17] 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.

[18] R. Needham and M. Schroeder, "Authentication revisited," *Operating Systems Review*, Vol. 21, No. 1, p. 7, January 1987.

[19] A. Shamir, "Identity-based cryptosystems and signature schemes," Crypto 91.

[20] J. Steiner, C. Newman and J. Schiller, "Kerberos: an authentication service for open network systems," *Proceedings of the USENIX Winter Conference*, pp. 191–202, 1988

[21] Y. Yacobi and Z. Shmuely. "On key distribution systems." Crypto 89.

A Proof of the Main Theorem

This section provides a sketch of the proof of Theorem 3.

Let E be an adversary who makes $Q = Q(k)$ oracle calls. We say that the i-th one of these calls occurs at *time* i.

Simplifying Lemmas and Assumptions. The *real* experiment refers to the experiment of running the protocol using E. The corresponding probability function will be denoted $\Pr[\cdot]$. Other probability functions will be introduced as needed.

We say that player A is *active* if there exist B and s such that some query was made to one of the following oracles: $\Pi^s_{A,B}, \Pi^s_{B,A}, \Psi^s_{A,B}, \Psi^s_{B,A}$. Similarly, a session number $s \in \mathsf{N}$ is active if there exist A, B such that a query was made to one of the following oracles: $\Pi^s_{A,B}, \Psi^s_{A,B}$.

We will make some simplifying and wlog assumptions. First, the (identities of the) set of active players are in the range $1, \ldots, Q$. (At most Q different players can be active, and by renaming we can make this assumption wlog). Similarly, the active session numbers are in the range $1, \ldots, Q$.

An *initiator* oracle is one who plays the role of A in the description of the protocol. That is, it sends out the first flow. A *responder* oracle is one who plays the role of B— it receives a flow to start.

It is convenient to use in our proof a stronger property of encryption than that given in our definition. Let $(\mathcal{E}, \mathcal{D})$ be an encryption scheme. Recall in the definition we choose a key a and a bit θ at random, and provide the eavesdropper with: (1a) an encryption $\alpha \leftarrow \mathcal{E}_a(\sigma_\theta)$ of σ_θ under a; and (1b) oracle access to the encryption algorithm $\mathcal{E}_a$. The job of the adversary is to predict θ. We now consider an augmented adversary called a *multiple* eavesdropper, who, in addition, gets to the above gets: (2a) an encryption $\beta \leftarrow \mathcal{E}_b(\sigma_\theta)$ of σ_θ under an independently chosen key b; and (2b) oracle access to $\mathcal{E}_b$. We will need the fact that the extra information does not help:

Lemma 8 If $(\mathcal{E}, \mathcal{D})$ is a secure encryption scheme then any PPT multiple eavesdropper has negligible advantage.

The above property of secure encryption scheme is standard, and we omit the proof.

Authenticity Of Flows. We begin by establishing some properties about what the oracle flows (almost always) look like in the real experiment when an oracle accepts. These properties depend only on the security of the message authentication scheme; the encryption scheme itself is completely irrelevant.

The intuition is as follows. Suppose an initiator oracle $\Pi^s_{A,B}$ accepts. For this to happen, the second flow has to look like $\alpha \,.\, \mu$ where μ is a MAC of $A \,.\, B \,.\, R_A \,.\, \alpha$ (α is a ciphertext). We claim it (almost certainly) must be that some S-oracle $\Psi^u_{A,B}$ at some time produced α (with a corresponding MAC). If not, we would be able to forge authentication tags. The formal statements follow, first for this case of an initiator oracle (as we have just described), and then for a responder oracle.

Define event $\mathsf{HAccI}^s_{A,B}$ ("honest, accepting initiator") as true if player A is uncorrupted and oracle $\Pi^s_{A,B}$ accepts in the role of an initiator.

Define event $\mathsf{AuthI}^s_{A,B}$ ("authenticity for initiator") as true if there exist times $\tau_1 < \tau_2 < \tau_3$, strings R_A, α, μ, μ', and a session number $u \in \mathsf{N}$ such that the following is true. At time τ_1, oracle $\Pi^s_{A,B}$ was started (in the role of initiator) and output nonce R_A. Then, at time τ_2, oracle $\Psi^u_{A,B}$ (was given some input and) output the message $\alpha \,.\, \mu$ for A (simultaneously, it output some message for B which we do not care about). Then, at time τ_3, oracle $\Pi^s_{A,B}$ received the flow $\alpha \,.\, \mu'$ and accepted (which implies $\mathsf{VF}_{K^{mac}_A}(A \,.\, B \,.\, R_A \,.\, \alpha,\ \mu') = 1$).

Lemma 9 For every $A, B \in I$ and every $s \in \mathsf{N}$ it is the case that $\Pr[\,\mathsf{HAccI}^s_{A,B} \ \wedge\ \neg\mathsf{AuthI}^s_{A,B}\,]$ is negligible.

Proof: Let A, B, s be such that the probability $\Pr[\,\mathsf{HAccI}^s_{A,B} \ \wedge\ \neg\mathsf{AuthI}^s_{A,B}\,]$ is non-negligible. We provide a forging algorithm F to break the message authentication scheme. Algorithm F has oracle access to $\mathsf{MAC}_a(\cdot)$ and $\mathsf{VF}_a(\cdot)$ where a was chosen at random. Algorithm F begins by choosing MAC and encryption keys for all players, except that no MAC key for A is chosen. Now it starts executing the experiment of running the protocol using E. In the execution, if an authentication tag under the MAC key of A is needed (this will be the case for flows output by oracles of the form $\Psi^u_{A,j}$ or $\Psi^v_{i,A}$) then F computes it by appealing to the oracle MAC_a. Similarly for verification. If at any point $\Pi^s_{A,B}$ accepts or A is corrupted then the execution stops. (We wouldn't be able to continue the simulation if A is corrupted because we can't return a). Else, it stops whenever the simulation of E is complete. One can check that the "view" of E in this experiment is the same as that in the original one at any point before the experiment stops. Note that times in which the the experiment stops due to $\Pi^s_{A,B}$ being corrupted are contributing nothing to $\mathsf{HAccI}^s_{A,B}$.

Suppose $\Pi^s_{A,B}$ is uncorrupted and has accepted. This means that at some time τ_1 it output some R_A, and at some time $\tau_3 > \tau_1$ it received some flow $\alpha \,.\, \mu'$ which it accepted. Let $y = A \,.\, B \,.\, R_A \,.\, \alpha$. The acceptance implies that $\mathsf{VF}_a(y, \mu') = 1$, i.e. μ' was a valid MAC for the string y. This (y, μ') is output by F as his (attempted) forgery. Now suppose $\mathsf{AuthI}^s_{A,B}$ is false. To show that (y, μ') is a successful forgery we need to check that y was (with high probability) never queried of $\mathsf{MAC}_a(\cdot)$ during the experiment.

The experiment stops at time τ_3 (since we stop when $\Pi^s_{A,B}$) accepts, so the check need only pertain to times prior to this. $\mathsf{AuthI}^s_{A,B}$ being true means y was never output by a Ψ oracle in between times τ_1 and τ_3, which means a query of the form $R_A \,.\, R_B$ was not made of any Ψ oracle in this period. So no MAC query of y was made between times τ_1 and τ_2. Finally, since R_A was a random k-bit string produced at time τ_1 by an (uncorrupted) oracle, the probability that $R_A \,.\, R_B$ was queried of a Ψ oracle before time τ_1 is at most $Q(k) \cdot 2^{-k}$. Putting everything together we can conclude that F succeeds in forgery with non-negligible probability. ■

We can proceed analogously for responder oracles, defining event $\mathsf{HAccR}^t_{B,A}$ ("honest, accepting initiator") and event $\mathsf{AuthR}^t_{B,A}$ ("authenticity for responder")— the definitions are omitted due to lack of space. Then just as above one can show:

Lemma 10 For every $B, A \in I$ and every $t \in \mathsf{N}$ it is the case that $\Pr[\,\mathsf{HAccR}^t_{B,A} \ \wedge\ \neg\mathsf{AuthR}^t_{B,A}\,]$ is negligible.

The Main Algorithm. We now present a multiple eavesdropper M which will be used later to show that the protocol protects session keys. Let us begin by giving some intuition about what M is trying to do.

Input to M are strings σ_0, σ_1 to be viewed as chosen according to Sn. Also, α, β which are the encryptions of σ_θ under a, b, respectively, θ to be thought of as chosen at random. M has oracle access to $\mathcal{E}_a, \mathcal{E}_b$, and is trying to predict θ. Algorithm M will run E. Oracle queries of E will be answered by M— the latter will itself "simulate" all the oracles to which E has access.

We know that eventually E points to a fresh oracle (with respect to a partner function whose definition is discussed below) and makes a test query. For example, say she points to a responder oracle $\Pi^t_{B,A}$. We would like that the encryption of the session key of this oracle be β. We will then return either σ_0 or σ_1 in the test query, so that E's answer can be taken as prediction for θ. To be able to implement this paradigm, we must be able to have β be the b-encrypted session key. By the previous lemmas we know the encrypted session key must have come from some Ψ-oracle. We pick a session u at random and bet one the oracle $\Psi^u_{A,B}$. When this oracle must speak, it distributes our special encrypted session keys (provided as M's input) instead of its choosing random ones.

The difficulty is for M to be able to simulate the execution of E without M's having the ability to decrypt under a, b. This means we cannot answer reveal queries pertaining to our special encrypted keys α, β. The algorithm below will simply **Fail** in those cases. Same for corrupt queries. Later, we will see that by making the right choice of partner function and using the lemmas of the previous section, we can prove that failure is sufficiently unlikely.

Algorithm $M^{\mathcal{E}_a(\cdot),\mathcal{E}_b(\cdot)}(\sigma_0, \sigma_1, \alpha, \beta)$.

Machine E picks $A, B \in \{1, \ldots, Q\}$ at random. It then picks $u \in \{1, \ldots, Q\}$ at random (a session number for S). For $i \in \{1, \ldots, Q\} - \{A, B\}$ pick encryption keys K_i^{enc} at random. Set $K_A^{enc} = K_B^{enc} = *$. For all $i = 1, \ldots, Q$ pick MAC-keys K_i^{mac} at random. Now for all $i = 1, \ldots, Q$ let $K_i = (K_i^{enc}, K_i^{mac})$. Although K_A^{enc} is formally $*$, think of it, intuitively, as being a. Similarly K_B^{enc} is, intuitively, b. Although M does not have these keys in its possession, it does have oracle access to the corresponding encryption functions, and will, in the simulation, give E the impression that $K_A^{enc} = a$ and $K_B^{enc} = b$.

Algorithm M starts executing the experiment of running the protocol using E. Algorithm M makes the necessary random choices (coins of E and of all oracles) and initializations (conversation variables) as in the real experiment. Now she starts running E. Oracle queries of E are answered as follows.

(1) (SendPlayer, i, j, s, x) — Compute the first two components (outgoing message m and decision δ) of $\Pi_{i,j}^s$ exactly as specified by the protocol. This is possible because this answer only requires the MAC-key of i, which is available to M.

(2) (SendS, i, j, v, x) — It must be the case that $\Psi_{i,j}^v$ has received $x = R_A \,.\, R_B$, for some R_A, R_B. If $(i, j, v) = (A, B, u)$ then M sets $\alpha_{i,j}^v = \alpha$ and $\beta_{j,i}^v = \beta$. Else it picks a session key $\sigma_{i,j}^v$ at random by running $Sn(1^k)$. Now:

- If $i \neq A$ then M has K_i^{enc} and can encrypt $\sigma_{i,j}^v$ by applying $\mathcal{E}_{K_i^{enc}}$. Else, M encrypts $\sigma_{i,j}^v$ by making an oracle call to $\mathcal{E}_a$. The result, in either case, is called $\alpha_{i,j}^v$.
- If $j \neq B$ then M has K_j^{enc} and can encrypt σ by applying $\mathcal{E}_{K_j^{enc}}$. Else, it encrypts σ by making an oracle call to $\mathcal{E}_b$. The result, in either case, is called $\beta_{j,i}^v$.

Now the appropriate MACs are computed (M has all keys for this) and the two appropriate outputs are made (one flow to i, another to j), according to the protocol.

(3) (Reveal, i, j, s) where $\Pi_{i,j}^s$ is an initiator. — We may assume $\Pi_{i,j}^s$ has accepted. If the last flow $\Pi_{i,j}^s$ received had the form $\alpha \,.\, \mu$, for some μ, then output **Fail** and halt. Else, let $\gamma \,.\, \mu$ be the last flow received by $\Pi_{i,j}^s$. If $\gamma = \alpha_{i,j}^v$ for some v then return $\sigma_{i,j}^v$. Else output **Fail** and halt.

(4) (Reveal, j, i, t) where $\Pi_{j,i}^t$ is a responder. — We may assume $\Pi_{j,i}^t$ has accepted. If the last flow $\Pi_{j,i}^t$ received had the form $\beta \,.\, \mu$ then output **Fail** and halt. Else, let $\gamma \,.\, \mu$ be the last flow received by $\Pi_{j,i}^t$. If $\gamma = \beta_{i,j}^v$ for some v then return $\sigma_{i,j}^v$. Else output **Fail** and halt.

(5) (Corrupt, i, K) — If $i \in \{A, B\}$ then output **Fail** and halt. Else, answer, and update, K_i, as in the real experiment. All the information to do this is available to M.

(6) (Test, i, j, s) where $\Pi_{i,j}^s$ is an initiator. — If $(i, j) \neq (A, B)$ then output **Fail** and halt. Else give σ_0 to E. Adversary E makes a prediction θ' (where, recall "0" is a bet of "real session key" and 1 is a bet of "random point in $Sn(1^k)$"). Algorithm M outputs (as its own prediction) this same bit θ', and then it halts.

(7) (Test, j, i, t) where $\Pi_{j,i}^t$ is a responder. — If $(j, i) \neq (B, A)$ then output **Fail** and halt. Else give σ_0 to E. Adversary E makes a prediction θ' which M outputs and halts.

The Partner Function. We define the partner function f which witnesses the security of the protocol. The value of $f_{i,j}^s(T)$ will be $*$ except where we now indicate.

The partner of a responder oracle: Look at the first two records of T associated to queries of $\Pi_{i,j}^s$. Suppose that the first of the two indicated records represents $\Pi_{i,j}^s$ in its role of a responder oracle—getting a query R_j. Suppose that the second of the indicated records shows $\Pi_{i,j}^s$ accepting. If both of these are so, then look to see if there is a unique t such that T indicates $\Pi_{j,i}^t$ generating a question of R_j. If so, set $f_{i,j}^s(T) = t$.

The partner of an initiator oracle: This case is just a little more complicated. Again, look at the first two records of T associated to queries of $\Pi_{i,j}^s$. Suppose that the first of these two records represents $\Pi_{i,j}^s$ in its role of initiator oracle: $\Pi_{i,j}^s$ got the question λ and gave an answer we denote R_i. Suppose that the second record indicated above shows a query (y_i, μ_i), getting a response $(*, \mathsf{A})$. If both of these are so, then look to see if T uniquely specifies some $\Psi_{i,j}^u$ sending out a message of the form (y_i, μ_i'), for some μ_i'. Look to see if this message was in response to some query of the form $i \,.\, j \,.\, R_i \,.\, R_j$, for some R_j. If so, then look to see if there is a unique t such that an oracle $\Pi_{j,i}^t$ generated a message of the form $i \,.\, j \,.\, R_i \,.\, R_j$. If yes, set $f_{i,j}^s(T) = t$.

Proof Of Protection Of Session Keys. We assume that the advantage $\text{Adv}_{P,f,Sn}^E(\cdot)$ of the adversary in the real experiment is non-negligible, and consider the following experiment, which we call Experiment X. Let $a, b \leftarrow \{0,1\}^k$ be random, and let σ_0, σ_1 be drawn randomly according to Sn. Flip a coin to get θ and let $\alpha \leftarrow \mathcal{E}_a(\sigma_\theta)$ and $\beta \leftarrow \mathcal{E}_b(\sigma_\theta)$. Run machine $M(\sigma_0, \sigma_1, \alpha, \beta)$ with oracle access to the encryption functions $\mathcal{E}_a(\cdot), \mathcal{E}_b(\cdot)$. Machine M is trying to predict θ.

We will now give a very high level argument that Experiment X yields success with non-negligible probability, contradicting the security of the encryption scheme (cf. Lemma 8). This will complete the proof.

Suppose that in the real experiment, $\Pi_{B,A}^t$ is the oracle pointed to, and it is a responder oracle. Since $\Pi_{B,A}^t$ is fresh, both A and B are uncorrupted. By Lemma 10 (applies since B is uncorrupted and has accepted) we know that except with negligible probability, the last flow to this oracle was authentic in the sense that the ciphertext in it emanated (with a correct MAC) from some $\Psi^u_{A,B}$.

Now turn to Experiment X. Our random choices of A, B, u in algorithm M imply that with probability at least $1/Q^3$ we have have "hit" the correct values. Now let us argue the failure probability is low.

Look at some $\Pi^s_{A,B}$. Suppose it accepts. By Lemma 9 (applies since A is uncorrupted) the flow it receives comes from some $\Psi^v_{A,B}$. If $v = u$ (this is a crucial step) we can check that $\Pi^s_{A,B}$ is the partner of $\Pi^t_{B,A}$ under f, so the freshness of $\Pi^t_{B,A}$ implies no reveal query was made of $\Pi^s_{A,B}$, and thus algorithm M will not fail. On the other hand if $v \neq u$ then a reveal query may be made, but M can answer it. So again M will not fail. We can put all this together to argue that Experiment X is successful non-negligibly often.

The case where the oracle pointed to is an initiator is omitted.