Key Agreement Protocols and their Security Analysis *

Simon Blake-Wilson[†] Don Johnson[‡] Alfred Menezes[§]

September 9, 1997

Abstract

This paper proposes new protocols for two goals: authenticated key agreement and authenticated key agreement with key confirmation in the asymmetric (public-key) setting. A formal model of distributed computing is provided, and a definition of the goals within this model supplied. The protocols proposed are then proven correct within this framework in the random oracle model. We emphasize the relevance of these theoretical results to the security of systems used in practice. Practical implementation of the protocols is discussed. Such implementations are currently under consideration for standardization [2, 3, 21].

Keywords: Diffie-Hellman, key agreement, provable security, random oracle paradigm.

1 Introduction

The *key agreement problem* is stated as follows: two entities wish to agree on keying information in secret over a distributed network. Since the seminal paper of Diffie and Hellman in 1976 [17], solutions to the key agreement problem whose security is based on the *Diffie-Hellman problem* in finite groups have been used extensively.

Suppose now that entity i wishes to agree on secret keying information with entity j. Each party desires an assurance that no party other than i and j can possibly compute the keying information agreed. This is the *authenticated key agreement (AK) problem*. Clearly this problem is harder than the key agreement problem in which i does not care who (or what) he is agreeing on a key with, for in this problem i stipulates that the key be shared with j and no-one else.

Several techniques related to the Diffie-Hellman problem have been proposed to solve the AK problem [23, 18, 1]. However, no practical solutions have been provably demonstrated to achieve this goal, and this deficiency has lead in many cases to the use of flawed protocols

*To be presented at the Sixth IMA International Conference on Cryptography and Coding, Cirencester, England, 17–19 December 1997.

[†]Dept. of Mathematics, Royal Holloway, University of London, Egham, Surrey TW20 0EX, United Kingdom. Email: phah015@rhbnc.ac.uk. The author is an EPSRC CASE student sponsored by Racal Airtech. Work performed while a visiting student at Auburn University funded by the Fulbright Commission.

[‡]Certicom Corp., 200 Matheson Blvd West, Mississauga, Ontario L5R 3L7, Canada. Email: djohnson@certicom.com

[§]Dept. of Discrete and Statistical Sciences, Auburn University, Auburn, AL 36849-5307, U.S.A. Email: menezal@mail.auburn.edu

(see [26, 16, 22, 25]). The flaws have, on occasion, taken years to discover; at best, such protocols must be employed with the fear that a flaw will later be uncovered.

Since in the AK problem, i merely desires that only j *can possibly* compute the key, and not that j has *actually* computed the key, solutions are often said to provide *implicit (key) authentication*. If i wants to make sure in addition that j really has computed the agreed key, then *key confirmation* is incorporated into the key agreement protocol, leading to so-called *explicit authentication*. The resulting goal is called *authenticated key agreement with key confirmation (AKC)*. It is a thesis of this paper that key confirmation essentially adds the assurance that i really is communicating with j to the AK protocol. Thus the goal of key confirmation is similar to the goal of *entity authentication*, as defined in [7]. More precisely, the incorporation of entity authentication into the AK protocol provides i the additional assurance that j *can* compute the key, rather than the (slightly) stronger assurance that j has *actually* computed the key.

Practical solutions that employ asymmetric techniques to solve the AK and AKC problems are clearly of fundamental importance to the success of secure distributed computing[1]. The motivation for this paper stems in part from the recent successes of the 'random oracle model' [8] in providing practical, provably good asymmetric schemes [8, 9, 11, 12, 27], and in part from the desire of various standards' bodies (in particular IEEE P1363 [21]) to lift asymmetric techniques in widespread use above the unsuccessful 'attack-response' design methodology. The goal of this paper is to make strides towards the provision of practical solutions for the AK and AKC problems which are provably good — firstly by providing clear, formal definitions of the goals of AK and AKC protocols, and secondly by furnishing practical, provably secure solutions in the random oracle model. The model of distributed computing adopted appears particularly powerful, and the definitions of security chosen particularly strong. The approach we take closely follows the approach of [7], where provable security is provided for entity authentication and authenticated key transport using symmetric techniques. Also relevant is the adaptation of techniques from [7] to the asymmetric setting found in [14].

Roughly speaking, the process of proving security comes in five stages:

1. specification of model;
2. definition of goals within this model;
3. statement of assumptions;
4. description of protocol;
5. proof that the protocol meets its goals within the model.

We believe that the goals of AK and AKC currently lack formal definition. It is one of our central objectives to provide such definitions.

We particularly wish to stress the important roles that appropriate assumptions, an appropriate model, and an appropriate definition of protocol security play in results of provable security—all protocols are provably secure in some model, under some definitions, or under some assumptions. Thus we believe that the emphasis in such work should be how appropriate the assumptions, definitions, and model which admit provable security are,

[1] Informally, by 'distributed computing' we refer to what we all think of as a computer network; that is a number of separate machines which can only communicate over public channels.

rather than the mere statement that such-and-such a protocol attains provable security. It is a central thesis of this work, therefore, that the model of distributed computing we describe models the environment in which solutions to the AK and AKC problems are required, and that the definitions given for the AK and AKC problems are the 'right' ones.

The remainder of the paper is organized as follows. §2 discusses the requirements of a secure key agreement protocol. §3 describes the model of distributed computing adopted. §4 discusses AKC protocols and introduces the protocols we propose. In §5 we turn our attention to the AK problem. §6 analyzes other properties of the protocols proposed to see whether the protocols attain additional desirable attributes. Finally, practical issues are discussed in §7, and §8 makes concluding remarks.

2 Properties of Key Agreement Protocols

There is a vast literature on key agreement protocols (see [25] for a survey). Unlike other primitives, such as encryption or digital signatures, it is not clear what constitutes an attack on a key agreement protocol. A number of distinct types of attacks have been proposed against previous schemes, as well as a number of less serious weaknesses. Therefore, before we can begin to analyze any protocol, it is necessary to identify what attacks a protocol should withstand, and what attributes are desirable for a protocol to have.

First we identify two types of attack:

1. *passive attacks*, where an adversary attempts to prevent a protocol from achieving its goal by merely observing honest entities carrying out the protocol;
2. *active attacks*, where an adversary additionally subverts the communications themselves in any way possible: by injecting messages, intercepting messages, replaying messages, altering messages, and the like.

Clearly it is essential for any secure protocol to withstand both passive and active attacks, since an adversary can reasonably be assumed to have these capabilities in a distributed network.

A number of desirable *attributes* of key agreement protocols have also been identified:

1. *known session keys.* A protocol still achieves its goal in the face of an adversary who has learned some previous session keys.
2. *(perfect) forward secrecy.* If long-term secrets of one or more entities are compromised, the secrecy of previous session keys is not affected.
3. *unknown key-share.* Entity i cannot be coerced into sharing a key with entity j without i's knowledge, i.e., when i believes the key is shared with some entity $l \neq j$.
4. *key-compromise impersonation.* Suppose i's secret value is disclosed. Clearly an adversary that knows this value can now impersonate i, since it is precisely this value that identifies i. However, it may be desirable that this loss does not enable an adversary to impersonate other entities to i.
5. *loss of information.* Compromise of other information that would not ordinarily be available to an adversary does not affect the security of the protocol. For example, in Diffie-Hellman type protocols, security is not compromised by loss of $\alpha^{S_i S_j}$ (where S_i represents entity i's long-term secret value).

6. *message independence*. Individual flows of a protocol run between two honest entities are unrelated.

Each attribute may be thought of as desirable for either AK or AKC protocols, or both. For example, we will argue in §5 that flaws in AKC protocols that exploit known session keys are a much more serious weakness than such flaws in AK protocols without key confirmation. Similarly, message independence is more desirable in AK protocols; conceptually AKC protocols inherently contain some message dependence.

Finally we mention that in some applications it may be desirable to demonstrate that a protocol is provably an *agreement*. Informally this means that neither party is able to affect the choice of key. In reality however, one entity selects its contribution to the key before the other, therefore enabling the other entity to test various selections of its contribution by calculating what the agreed key will be. To formalize this, we could say that this trial-and-error procedure is effectively the best way for either entity to effect the choice of key. While we will not discuss this further, heuristic arguments suggest that our protocols achieve such an *agreement* property.

3 Model of Distributed Environment

Before formal statements of the problems can be made, we need a formal model to work in. First some notation and language is introduced, and then the model itself is described. The model is a variant of the *Bellare-Rogaway model*, described in [7, 10]. The description we provide of the model is necessarily terse; see [7] for further details.

3.1 Set-up

$\{0,1\}^*$ denotes the set of finite binary strings, and λ denotes the empty string. $I = \{1, \ldots, N_1\}$ is the set of entities in this environment (the adversary is not included as an entity). The number of entities being dealt with is polynomial in the security parameter, k, so that $N_1 = T_1(k)$ for some polynomial function T_1. A real-valued function $\epsilon(k)$ is *negligible* if for every $c > 0$ there exists $k_c > 0$ such that $\epsilon(k) < k^{-c}$ for all $k > k_c$. A function $n(k)$ is *non-negligible* if it is not a negligible function.

Definition 1 A *protocol* is a pair $P = (\Pi, \mathcal{G})$ of probabilistic polytime computable functions (polytime in their first input):

Π specifies how (honest) players behave;

$\mathcal{G}$ generates key pairs for each entity.

The domain and range of these functions is as follows. Π takes as input:

1^k	—	the security parameter;
$i \in I$	—	identity of sender;
$j \in I$	—	identity of intended recipient;
$K_{i,j}$	—	i's key pair K_i together with j's public value;
tran	—	a transcript of the protocol run so far (i.e. the ordered set of messages transmitted and received by i so far in this run of the protocol).

$\Pi(1^k, i, j, K_{i,j}, tran)$ outputs a triple (m, δ, κ), where:

m $\in \{0,1\}^* \cup \{*\}$ is the next message to be sent from i to j ($*$ indicates no message is sent);

δ $\in \{\texttt{Accept}, \texttt{Reject}, *\}$ is i's current decision ($*$ indicates no decision yet reached);

κ is the agreed key.

Our protocols are described in terms of arithmetic operations in the subgroup generated by an element α of prime order q in the multiplicative group $\mathbb{Z}_p^* = \{1, 2, \ldots, p-1\}$, where p is a prime. In each case, an entity's private value is an element S_i of $\mathbb{Z}_q^* = \{1, 2, \ldots, q-1\}$, and the corresponding public value is $P_i = \alpha^{S_i} \bmod p$ [2], so that i's key pair is $K_i = (S_i, P_i)$. Note that the protocols can be described equally well in terms of the arithmetic operations in any finite group; of course, we would then have to convert our security assumptions on the Diffie-Hellman problem to that group.

$\mathcal{G}$ takes as input the security parameter 1^k and selects the triple of global parameters (p, q, α) to be used by all entities by running $\mathcal{G}_{DH}$, the parameter generator for a Diffie-Hellman scheme, on input 1^k. The operation of $\mathcal{G}_{DH}$ will be discussed in §4, when a Diffie-Hellman scheme is defined. $\mathcal{G}$ then picks a secret value S for each entity by making N_1 independent random samples from $\mathbb{Z}_q^*$, and calculates the public value $P = \alpha^S$ of each entity. $\mathcal{G}$ then forms a directory *public-info* containing the global parameters (p, q, α) and an entry corresponding to each entity — the entry corresponding to entity i consists of the pair (i, P_i) of i's identifier and i's public value. $\mathcal{G}$ outputs each entity's key pair along with the directory *public-info*.

$\mathcal{G}$ is a technical description of the key generation process. It is a formal model designed to capture the attributes of the techniques typically used to generate keys in a distributed environment. Of course, in real life, each entity will usually generate key pairs itself and then get them certified by a Certification Authority.

A generic execution of a protocol between two players is called a *run* of the protocol. While a protocol is formally specified by a pair of functions $P = (\Pi, \mathcal{G})$, in this paper it is informally specified by the description of a run between two arbitrary entities. Any particular run of a protocol is called a *session*. The word 'session' is often associated with anything specific to one particular execution of the protocol. For example, the keying information agreed in the course of a protocol run is referred to as a *session key*. The individual messages that form a protocol run are called *flows*.

3.2 Description of Model

Our adversary is afforded enormous power. She controls all communication between entities, and can at any time ask an entity to reveal its long-term secret key. Furthermore she may at any time initiate sessions between any two entities, engage in multiple sessions with the same entity at the same time, and in some cases ask an entity to enter a session with itself.

With such a powerful model it is not clear what it means for a protocol to be secure. Informally, we say that an AK protocol is secure if no adversary can learn anything about a session key held by an uncorrupted entity i (an entity whose long-term keying material

[2] The operator '$\bmod p$' will henceforth be omitted.

she has not revealed), provided that i has computed that session key in the belief that it is shared with another entity j (who is also uncorrupted). Again informally, we will say that an AKC protocol is secure if the protocol distributes a key just like an AK protocol, and has the additional property that an accepting entity i is assured that it has been involved in a real-time communication with j. Therefore to make an entity accept in an AKC protocol, the adversary effectively has to act just like a wire.

We now formalize the above discussion.

An *adversary*, E, is a probabilistic polytime Turing Machine taking as input the security parameter 1^k and the directory *public-info*. E has access to a collection of oracles:

$$\{\Pi^s_{i,j} : i \in I, j \in I, s \in \{1, \ldots, N_2\}\} \ .$$

Oracle $\Pi^s_{i,j}$ behaves as entity i carrying out a protocol session in the belief that it is communicating with j for the sth time (i.e. the sth run of the protocol between i and j). Each $\Pi^s_{i,j}$ oracle maintains its own variable *tran* to store its view of the run so far. E is equipped with a polynomial number of $\Pi_{i,j}$ oracles, so that $N_2 = T_2(k)$ for some polynomial function T_2. Each $\Pi^s_{i,j}$ oracle takes as initial input the security parameter 1^k, the key pair K_i assigned to entity i by $\mathcal{G}$, a *tran* value of λ, and the directory *public-info*.

E is allowed to make three types of queries of its oracles, as illustrated in the table below.

Query	Oracle reply	Oracle update
$\texttt{Send}(\Pi^s_{i,j}, x)$	$\Pi^{m\delta}(1^k, i, j, K_{i,j}, tran.x)$	$tran \leftarrow tran.x.m$
$\texttt{Reveal}(\Pi^s_{i,j})$	$\Pi^{\kappa}(1^k, i, j, K_{i,j}, tran)$	—
$\texttt{Corrupt}(i, K)$	K_i	$K_i \leftarrow K$

In the table, $\Pi^{m\delta}$ denotes the first two arguments of $\Pi^s_{i,j}$'s output, and Π^{κ} denotes the third. The `Send` query represents E giving a particular oracle message x as input. E initiates a session with the query $\texttt{Send}(\Pi^s_{i,j}, \lambda)$, i.e. by sending the oracle it wishes to start the session the empty string λ. `Reveal` tells a particular oracle to reveal whatever session key it currently holds. `Corrupt` tells all $\Pi^s_{i,j}$ oracles, for any $j \in I$, $s \in \{1, 2, \ldots, N_2\}$, to reveal entity i's long-term secret value to E, and further to replace K_i with any valid key pair K of E's choice. In addition, all oracles' copies of i's public value in the directory *public-info* are updated.

Our security definitions now take place in the context of the following experiment — the experiment of running a protocol $P = (\Pi, \mathcal{G})$ in the presence of an adversary E using security parameter k:

1. Toss coins for $\mathcal{G}$, E, all oracles $\Pi^s_{i,j}$, and any public random oracles;
2. Run $\mathcal{G}$ on input 1^k;
3. Initialize all oracles;
4. Start E on input 1^k and *public-info*.

Now when E asks oracle $\Pi^s_{i,j}$ a query, $\Pi^s_{i,j}$ calculates the answer using the description of Π. This definition of the experiment associated with a protocol implies that when we speak of

the probability that a particular event occurs during the experiment, then this probability is assessed over all the coin tosses made in step 1 above.

The first step in defining the security of a protocol is to show that the protocol is 'well-defined'. To assist in this process we sometimes need to consider the following particularly friendly adversary. For any pair of oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, the *benign adversary* on $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ is the deterministic adversary that always performs a single run of the protocol between $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, faithfully relaying flows between these two oracles.

An oracle $\Pi^s_{i,j}$ has *accepted* if $\Pi^\delta(1^k, i, j, K_{i,j}, tran) = \texttt{Accept}$, it is *opened* if there has been a $\texttt{Reveal}(\Pi^s_{i,j})$ query, and it is *corrupted* if there has been a $\texttt{Corrupt}(i, \cdot)$ query. $tran^s_{i,j}$ will be used to denote the current state of $\Pi^s_{i,j}$'s variable *tran*.

So far all we have done is describe the model. We are now ready to give formal definitions of the goals.

4 AKC

First, we'll look at the AKC problem. The model described in §3 provides the necessary framework for our security proofs; however, before we can prove anything about any protocol, a formal definition of the goal of a secure AKC protocol must be given.

4.1 Definition of Security

As stated in the introduction, a central thesis to this paper is that the goal of an AKC protocol is essentially identical to the goal of an authenticated key transport protocol [7, 14]. The intent of an AKC protocol is therefore to assure two specified entities that they are involved in a real-time communication with each other. Further, the protocol must provide the two entities with a key distributed uniformly at random from $\{0,1\}^k$. No adversary should be able to learn any information about the agreed key held by an uncorrupted entity i, provided the entity j that i believes it is communicating with is also uncorrupted.

MATCHING CONVERSATIONS. To formalize the notion that two oracles are involved in a real-time communication, the concept of matching conversation is defined. For simplicity we focus on the case where R, the number of flows in the protocol, is odd. The case where R is even is analogous. The idea of *matching conversations* was first formulated in [13], refined in [18], and later formalized in [7].

Fix an execution of an adversary E. For any oracle $\Pi^s_{i,j}$ its *conversation* can be captured by a sequence:

$$C = C^s_{i,j} = (\tau_1, \alpha_1, \beta_1), (\tau_2, \alpha_2, \beta_2), \ldots, (\tau_m, \alpha_m, \beta_m) \ .$$

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 queries.

If oracle $\Pi^s_{i,j}$ has $\alpha_1 = \lambda$, it is called an *initiator oracle*; otherwise it is called a *responder oracle*.

Definition 2 ([7]) Fix a number of flows $R = 2\rho - 1$ and an R-flow protocol P. Run P in the presence of an adversary E and consider two oracles $\Pi^s_{i,j}$, an initiator oracle, and $\Pi^t_{j,i}$, a responder oracle, that engage in conversations C and C' respectively.

1. C' is said to be a *matching conversation* to C if there exist $\tau_0 < \tau_1 < \cdots < \tau_{R-1}$ and $\alpha_1, \beta_1, \ldots, \beta_{\rho-1}, \alpha_\rho$ such that C is prefixed by:

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

   and C' is prefixed by:

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

2. C is said to be a *matching conversation* to C' if there exist $\tau_0 < \tau_1 < \cdots < \tau_R$ and $\alpha_1, \beta_1, \ldots, \beta_{\rho-1}, \alpha_\rho$ such that C' is prefixed by:

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

   and C is prefixed by:

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

If C is a matching conversation to C' and C' is a matching conversation to C, then $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ are said to have had *matching conversations*.

Roughly speaking, this definition captures when the adversary E merely acts like a wire. In the first case, E faithfully carries all $\Pi^s_{i,j}$'s messages (except possibly the last) to $\Pi^t_{j,i}$, and then relays the replies back. The second case implies the first, but in addition $\Pi^s_{i,j}$'s last message is relayed to $\Pi^t_{j,i}$.

PROTOCOL SECURITY. Matching conversations now provide the necessary formalism to define the assurance provided to entity i during an AKC protocol that it has been involved in a real-time communication with entity j. Let $\texttt{No-Matching}^E(k)$ denote the event that, when protocol P is run against adversary E, there exists an oracle $\Pi^s_{i,j}$ which accepted but there is no oracle $\Pi^t_{j,i}$ which has engaged in a matching conversation to $\Pi^s_{i,j}$. Further, we require $i, j \notin \mathcal{C}$ (where $\mathcal{C}$ denotes the set of entities corrupted by the adversary E during the experiment).

The notion that no adversary can learn information about session keys is formalized along the lines of polynomial indistinguishability. Specifically at the end of its execution, the adversary should be unable to gain more than a negligible advantage when it tries to distinguish the actual key held by an uncorrupted entity from a key sampled at random from $\{0,1\}^k$.

Therefore we make the following addendum to the experiment. Call a $\Pi^s_{i,j}$ oracle *fresh* if it has accepted, neither i nor j has been corrupted, it remains unopened, and there is no opened oracle $\Pi^t_{j,i}$ with which it has had a matching conversation. After the adversary has asked all the queries it wishes to make, E selects any fresh $\Pi^s_{i,j}$ oracle. E asks this oracle a single new query:

$$\texttt{Test}(\Pi^s_{i,j}) \ .$$

To answer the query, the oracle flips a fair coin $b \leftarrow \{0,1\}$, and returns the session key $\kappa_{i,j}^s$ if $b = 0$, or else a random key sampled from $\{0,1\}^k$ if $b = 1$. The adversary's job is now to guess b. To this end, E outputs a bit `Guess`. Let $\texttt{Good-Guess}^E(k)$ be the event that $\texttt{Guess} = b$. Then we define:

$$advantage^E(k) = |Pr[\texttt{Good-Guess}^E(k)] - \tfrac{1}{2}| \ .$$

Definition 3 A protocol $P = (\Pi, \mathcal{G})$ is a *secure AKC protocol* if:

1. In the presence of the benign adversary on $\Pi_{i,j}^s$ and $\Pi_{j,i}^t$, both oracles always accept holding the same session key κ, and this key is distributed uniformly at random on $\{0,1\}^k$;

and if for every adversary E:

2. If uncorrupted oracles $\Pi_{i,j}^s$ and $\Pi_{j,i}^t$ have matching conversations then both oracles accept and hold the same session key κ;
3. The probability of $\texttt{No-Matching}^E(k)$ is negligible;
4. $advantage^E(k)$ is negligible.

The first condition says that in the presence of a benign adversary, oracles always accept holding the same, randomly distributed key. The second says that in the presence of any adversary if two entities behave correctly, and the transmissions between them are not tampered with, then both accept and hold the same key. The third says that essentially the only way for any adversary to get an uncorrupted entity to accept in a run of the protocol with any other uncorrupted entity is by relaying communications like a wire. The fourth says that no adversary can learn any information about a session key held by a fresh oracle.

This definition is identical to the definition of an authenticated key transport protocol as defined in [7]. While it seems strange to suggest that the definitions of security for two goals that have traditionally been regarded as distinct should in fact be identical, the justification for this security definition for AKC is straightforward. It is intuitively what we require from an AKC protocol. Further, a review of the literature reveals that a number of the attacks proposed on previous protocols can be explained by the observation that the protocols concerned do not meet this definition.

4.2 Description of Primitives

Before we can specify a particular protocol, the various primitives our protocols employ must be described and the security of each of these primitives defined. The two primitives used are *message authentication codes* (MACs) and *Diffie-Hellman schemes* (DHSs). The proposed protocols use MACs to provide key confirmation because this seems to be the least restrictive primitive to employ; in particular, a MAC is unlikely to be subject to the same export restrictions as an encryption scheme. Of course, some applications may wish to use another primitive to achieve confirmation. This is perfectly realistic: for example, if the agreed session key is later to be used for encryption, it seems sensible to employ an encryption scheme to achieve key confirmation, rather than waste time implementing a MAC.

MESSAGE AUTHENTICATION CODES. Provably secure message authentication codes have been discussed in [6, 5, 4]. We restrict attention to MACs that have key space uniformly distributed on $\{0,1\}^k$.

Definition 4 ([4]) A *message authentication code* is a deterministic polytime algorithm $MAC_{(\cdot)}(\cdot)$. To authenticate a message m, an entity with key κ' computes:

$$(m, a) = MAC_{\kappa'}(m) \ .$$

The *authenticated message* is the pair (m, a); a is called the *tag* on m. To verify (m, a) is indeed an authenticated message, any entity with key κ' checks that $MAC_{\kappa'}(m)$ does indeed equal (m, a).

An adversary F (of the MAC) is a probabilistic polytime algorithm which has access to an oracle that computes MACs under a randomly chosen key κ'. The output of F is a pair (m, a) such that m was not queried of the MACing oracle.

Definition 5 ([4]) A MAC is a *secure MAC* if for every adversary F of the MAC, the function $\epsilon(k)$ defined by

$$\epsilon(k) = Pr[\kappa' \leftarrow \{0,1\}^k; (m, a) \leftarrow F : (m, a) = MAC_{\kappa'}(m)]$$

is negligible.

Roughly speaking, this means a MAC is secure only if the probability of forging a valid tag on any message that has not yet been authenticated using a call to the MACing oracle is negligible. Thus we require a MAC to withstand an adaptive chosen-message attack. Note that since the MACing algorithm is deterministic, each message m has a unique tag a under κ'. This is important in conjunction with matching conversations — it means that an adversary who sees an authenticated message (m, a) is unable to alter a to another valid tag a' for m.

DIFFIE-HELLMAN SCHEMES. The assumption that the Diffie-Hellman problem is hard is common in the cryptographic literature. In order to formalize what we mean by 'the Diffie-Hellman problem is hard', we first define a Diffie-Hellman scheme.

Definition 6 A *Diffie-Hellman scheme* (DHS) is a pair of polytime algorithms, $(\mathcal{G}_{DH}, calc)$, the first being probabilistic. On input 1^k, $\mathcal{G}_{DH}$ generates a triple of global parameters (p, q, α). p and q are primes such that q divides $p-1$, and α is an element of order q in $\mathbb{Z}_p^*$. *calc* exponentiates in $\mathbb{Z}_p^*$ — it takes as input $((p, q, \alpha), g, x)$ where the triple (p, q, α) has been generated by $\mathcal{G}_{DH}$, g is in $\mathbb{Z}_p^*$, and x is an integer satisfying $0 \leq x \leq p-2$. *calc* outputs $g^x \bmod p$.

An adversary F (of the Diffie-Hellman scheme) is a probabilistic polytime algorithm which takes as input a parameter set (p, q, α) generated using $\mathcal{G}_{DH}$, and a pair $(\alpha^{R_1}, \alpha^{R_2})$ for R_1 and R_2 chosen independently at random from $\mathbb{Z}_q^*$. The output of F is an element g of $\mathbb{Z}_p^*$.

Definition 7 A secure DHS is one for which the function $\epsilon(k)$ defined by

$$\epsilon(k) = Pr[(p,q,\alpha) \leftarrow \mathcal{G}_{DH}(1^k); R_1, R_2 \leftarrow \mathbb{Z}_q^*; g \leftarrow F((p,q,\alpha),(\alpha^{R_1},\alpha^{R_2})) : g = \alpha^{R_1 R_2}]$$

is negligible for every adversary F.

The Diffie-Hellman problem is *hard* if there exists a secure Diffie-Hellman scheme. This formal definition corresponds precisely with our intuitive notion that the Diffie-Hellman problem is 'hard' (in subgroups of prime order) — i.e. it is extremely unlikely that anyone can guess $\alpha^{R_1 R_2}$ given only α^{R_1} and α^{R_2}.

The proofs in this paper assume that i does not enter protocol runs with itself. This condition can be removed provided the DHS being employed is also secure against an adversary that takes as input a pair $(\alpha^{R_1}, \alpha^{R_1})$ rather than $(\alpha^{R_1}, \alpha^{R_2})$. Call a secure DHS $*$-*secure* if it remains secure against this modified opponent. Work done by Maurer and Wolf [24] suggests that secure DHSs and $*$-secure DHSs are equivalent.

RANDOM ORACLES. The security proofs of this paper take place in the 'random oracle model' [8]. All parties involved in the protocols are supplied with a 'black-box' random function

$$\mathcal{H}(\cdot) : \{0,1\}^* \longrightarrow \{0,1\}^k \ .$$

All random oracles in this paper will map finite strings to strings of length k. Following [8], 2^∞ will denote the set of all random oracles. It is often convenient to think of $\mathcal{H}$ defined in terms of its coin tosses in the following way. When $\mathcal{H}$ is queried for the first time, say on string x, it returns the string of length k corresponding to its first k coin tosses as $\mathcal{H}(x)$. When queried with a second string, say x', first $\mathcal{H}$ compares x and x'. If $x' = x$, $\mathcal{H}$ again returns its first k coin tosses $\mathcal{H}(x)$. Otherwise $\mathcal{H}$ returns its second k tosses as $\mathcal{H}(x')$. And so on.

Of course, a random oracle is a theoretical construct designed to facilitate security analysis. In instantiations, $\mathcal{H}$ will be modeled by a hash function H (or some more complex 'key derivation function' [21]). We emphasize that the security proofs take place in the random oracle model, and that instantiating a random oracle using a specific function is a heuristic step, known as 'the random oracle paradigm'. We refer the reader to [8] for further discussion of the 'random oracle paradigm'; here we merely echo the assertion of those authors that 'it is important to neither over-estimate nor under-estimate what the random-oracle paradigm buys you in terms of security guarantees'. Also [12] contains an excellent discussion of the implications of 'provable security in the random oracle model'.

4.3 Protocol 1

We can now describe the first AKC protocol proposed. It is represented graphically in Figure 1. Use $\in_R$ to denote an element chosen independently at random, and commas to denote a unique encoding through concatenation (or any other unique encoding). $\mathcal{H}_1$ and $\mathcal{H}_2$ represent independent random oracles. When entity i wishes to initiate a run of P with entity j, i selects $R_i \in_R \mathbb{Z}_q^*$ and sends α^{R_i} to j. On receipt of this string, j checks that $2 \leq \alpha^{R_i} \leq p-1$ and $(\alpha^{R_i})^q = 1$, then chooses $R_j \in_R \mathbb{Z}_q^*$, and computes α^{R_j} and $\kappa' = \mathcal{H}_1(\alpha^{S_i S_j})$. Finally, j uses κ' to compute $MAC_{\kappa'}(2, j, i, \alpha^{R_j}, \alpha^{R_i})$, and sends this

authenticated message to i. (Recall that $MAC_{\kappa'}(m)$ represents the pair (m, a), not just the tag a.) On receipt of this string, i checks that the form of this message is correct, and that $2 \leq \alpha^{R_j} \leq p-1$ and $(\alpha^{R_j})^q = 1$. i then computes κ' and verifies the authenticated message it received. If so, i accepts, and sends back to j $MAC_{\kappa'}(3, i, j, \alpha^{R_i}, \alpha^{R_j})$. Upon receipt of this string, j checks the form of the message, verifies the authenticated message, and accepts. Both parties compute the agreed session key as $\kappa = \mathcal{H}_2(\alpha^{R_i R_j})$. If at any stage, a check or verification performed by i or j fails, then that party terminates the protocol run, and rejects.

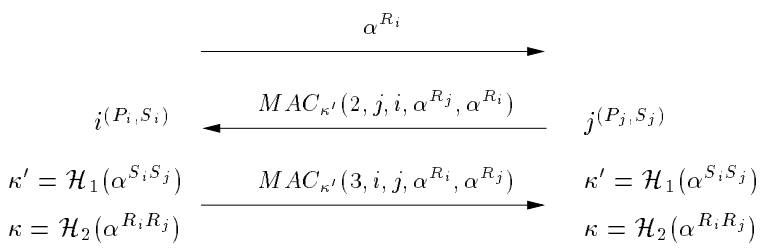

Figure 1: Protocol 1

Theorem 8 Protocol 1 is a secure AKC protocol provided the DHS and MAC are secure and $\mathcal{H}_1$ and $\mathcal{H}_2$ are independent random oracles.

The proof of this theorem appears in Appendix A.

COMMENTS. In practice, entity i may wish to append its identity to the first flow of Protocol 1. Doing so in no way affects the security proof. We omit this identity because certain applications may desire to identify the entities involved at the packet level rather than the message level — in this instance, identifying i again is therefore superfluous.

Note that entities use two distinct keys in Protocol 1 — one key for confirmation, and a different key as the session key for subsequent use. This separation appears important. In particular, the common practice of using the same key both for confirmation and as the session key is clearly dangerous if this means the same key is used by more than one primitive.

It is easy to show that the probability that a $\Pi^s_{i,j}$ oracle with $i, j \notin \mathcal{C}$ has a matching conversation with more than one $\Pi_{j,i}$ oracle in a run of Protocol 1 is negligible (the same is true of all the protocols described in this paper).

Protocol 1 is different from most proposed AKC protocols in the manner that entities employ their long-term secret values and session-specific secret values. Most proposed protocols use both long-term secrets and short-term secrets in the formation of all keys. In Protocol 1, long-term secrets and short-term secrets are used in quite independent ways. Long-term secrets are used only to form a session-independent confirmation key and short-term secrets only to form the agreed session key. Conceptually this approach has both advantages and disadvantages over more traditional techniques. On the plus side, the use of long-term keys and short-term keys is distinct, serving to clarify the effects of a key compromise — compromise of a long-term secret is fatal to the security of future sessions,

and must be remedied immediately, whereas compromise of a short-term secret effects only that particular session. On the negative side, both entities must maintain a long-term shared secret key κ' in Protocol 1.

Separation of an AK phase of this AKC protocol appears impossible.

4.4 Protocol 2

Protocol 2 is an AKC protocol designed to deal with some of the disadvantages of Protocol 1. It is represented graphically in Figure 2. The actions performed by entities i and j are similar to those of Protocol 1, except that the entities use both their short-term and long-term values in the computation of both the keys they employ. Specifically, the entities use $\kappa' = \mathcal{H}_1(\alpha^{R_iR_j}, \alpha^{S_iS_j})$ as their MAC key for this session, and $\kappa = \mathcal{H}_2(\alpha^{R_iR_j}, \alpha^{S_iS_j})$ as the agreed session key.

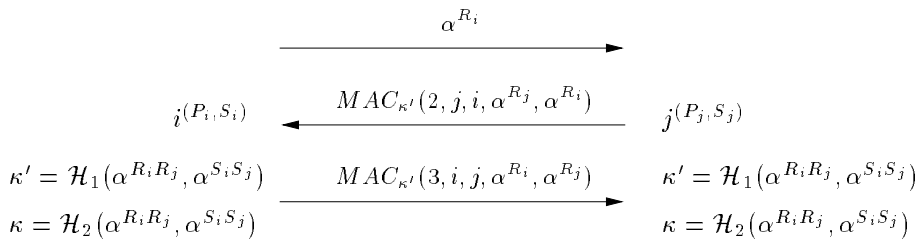

Figure 2: Protocol 2

Theorem 9 Protocol 2 is a secure AKC protocol provided the DHS and MAC are secure and $\mathcal{H}_1$ and $\mathcal{H}_2$ are independent random oracles.

The proof of this theorem appears in Appendix B.

COMMENTS. Unlike Protocol 1, both long-term secrets and both short-term secrets are used in Protocol 2 to form each key. While this makes the effect of a compromise of one of these values less clear, it also means that there is no long-term shared key used to MAC messages in every session between i and j. However, the two entities do still share a long-term secret value $\alpha^{S_iS_j}$. This value must therefore be carefully guarded against compromise, along with S_i and S_j themselves. Conceptually it is possible to separate the AK phase and the key confirmation phase in Protocol 2. This will be the subject of §5.

5 AK

DEFINITION OF SECURITY. In the past, defining the goal of an AK protocol has proved difficult. The clarity of the definition provided for AKC protocols (Definition 3 in §4) allows us to separate out a definition of security for AK protocols. Informally, we require an AK protocol to distribute a key to two specified entities in such a way that no adversary can learn any information about the agreed key. This is translated into the formal language of our model as follows.

Definition 10 A protocol P is a *secure AK* protocol if:

1. In the presence of the benign adversary on $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$, both oracles always accept holding the same session key κ, and this key is distributed uniformly at random on $\{0,1\}^k$;

and if for every adversary E:

2. If uncorrupted oracles $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have matching conversations then both oracles accept and hold the same session key κ;
3. $advantage^E(k)$ is negligible.

Conditions 1 and 2 say that a secure AK protocol does indeed distribute a key of the correct form. Condition 3 says that no adversary can learn any information about the key held by a fresh oracle.

PROTOCOL 3. Our first attempt at specifying a secure AK protocol tries to separate an AK phase from Protocol 2. Figure 3 contains a graphical representation of the actions taken by i and j in a run of Protocol 3.

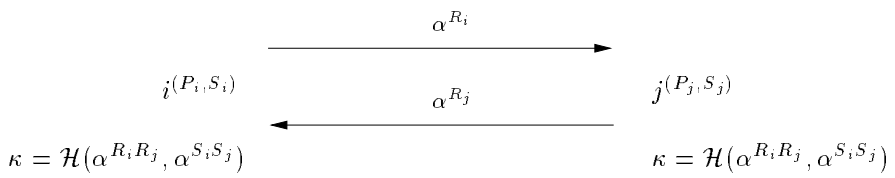

Figure 3: Protocol 3

Theorem 11 Protocol 3 is a secure AK protocol as long as E makes no `Reveal` queries, and provided that the DHS and MAC are secure and $\mathcal{H}$ is a random oracle .

The proof of this theorem appears in Appendix C.

COMMENTS. To see that Protocol 3 is not a secure AK protocol if an adversary can reveal unconfirmed session keys, notice the following attack. E begins two runs of the protocol, one with $\Pi^s_{i,j}$, and one with $\Pi^u_{i,j}$. Suppose $\Pi^s_{i,j}$ sends α^{R_i}, and $\Pi^u_{i,j}$ sends $\alpha^{R'_i}$. E now forwards α^{R_i} to $\Pi^u_{i,j}$, and $\alpha^{R'_i}$ to $\Pi^s_{i,j}$. E can now discover the session key $\kappa = \mathcal{H}(\alpha^{R_iR'_i}, \alpha^{S_iS_j})$ held by $\Pi^s_{i,j}$ by revealing the (same) key held by $\Pi^u_{i,j}$.

Theorem 11 really says that care must be taken when separating authenticated key agreement from key confirmation. Protocol 3 above is not a secure AK protocol in the full model of distributed computing we've been adopting, but can nonetheless be turned into a secure AKC protocol, as in Protocol 2. At issue here is whether it is realistic to expect that an adversary can learn keys that have not been confirmed. Indeed, studying the list of suggested reasons for session key compromise in [16], it can be seen that the majority of the scenarios discussed lead to the disclosure of *confirmed* keys.

Therefore, although in this paper we have tried to separate the goals of AK and AKC, the principle that Theorem 11 suggests is that no key agreed in an AK protocol should

be used without key confirmation. The only reason we have endeavored to separate authenticated key agreement from key confirmation is to allow flexibility in how a particular implementation chooses to achieve key confirmation. For example, architectural considerations may require key agreement and key confirmation to be separated — some systems may provide key confirmation during a 'real-time' telephone conversation subsequent to agreeing a session key over a computer network, while others may instead prefer to carry out confirmation implicitly by using the key to encrypt later communications.

The reason that we have specified the use of a subgroup of prime order by the DHSs in this paper is to avoid various known session key attacks on AK protocols that exploit the fact that a key may be forced to lie in a small subgroup of $\mathbb{Z}_p^*$. Note however that this condition is not necessary for the security proofs to work — from the point of view of the security proofs, we could equally well have made assumptions about DHSs defined in $\mathbb{Z}_p^*$ rather than a subgroup of $\mathbb{Z}_p^*$.

Theorem 11 testifies to the strength of our definition for security of an AK protocol. Notice in particular that, as is the case with Protocol 3, many previous AK protocols (e.g., those of [23]) do not contain asymmetry in the formation of the agreed key to distinguish which entity involved is the protocol's initiator, and which is the protocol's responder. Such protocols certainly will not meet the security requirements of Definition 10.

PROTOCOL 4. We speculate that the following protocol meets the full rigor required by Definition 10. Again, instead of describing the actions of i and j verbally, we illustrate these actions in Figure 4.

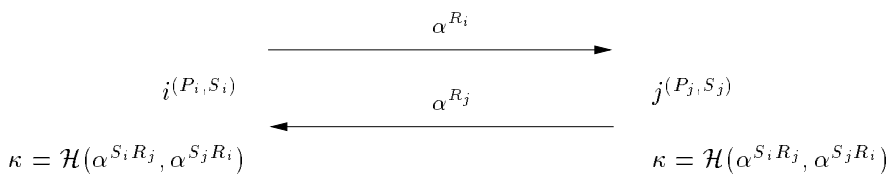

Figure 4: Protocol 4

Conjecture 12 Protocol 4 is a secure AK protocol provided the DHS and MAC are secure and $\mathcal{H}$ is a random oracle.

COMMENTS. While at first glance, Protocol 4 may look almost identical to the well-known MTI protocol [23], where the shared value computed is $\alpha^{S_i R_j + S_j R_i}$, notice the following important distinction. Entity i calculates a different key in Protocol 4 depending on whether i believes it is the initiator or responder. In the first case, i computes $\kappa = \mathcal{H}(\alpha^{S_i R_j}, \alpha^{S_j R_i})$, and in the second case $\kappa = \mathcal{H}(\alpha^{S_j R_i}, \alpha^{S_i R_j})$. As we remarked above, such asymmetry is essential in a secure AK protocol under Definition 10. Of course, such asymmetry is not always desirable — a particular environment may require that i calculate the same key no matter whether i is the initiator or responder. In such a case, Definition 10 would require (slight) modifications.

If indeed it can be shown that Protocol 4 is a secure AK protocol, then we imagine it can be turned into a secure AKC protocol in the same spirit as Protocol 2.

6 Attributes of Protocols

In this section, we discuss which of the attributes described in §2 the proposed protocols possess.

6.1 Known Session Keys

The `Reveal` query is designed to capture the notion that an adversary may learn previous session keys — in the model of distributed computing adopted, E may learn previous session keys by simply asking for them. The security definitions we have given for the AK and AKC problems demand that no adversary be able to learn any information about the session key held by a fresh oracle even when the adversary employs its `Reveal` query to learn other session keys. Any protocol for either AK or AKC which is secure under the definitions given will therefore resist known session key attacks.

6.2 Forward Secrecy

Intuitively, both the AKC protocols proposed achieve forward secrecy. Consider, for example, Protocol 1. Suppose entity i's secret value S_i is compromised at some time τ. Protocol 1 is a secure AKC protocol, so i is assured that it engaged in a matching conversation with entity j during the agreement of any session key, provided this agreement took place before time τ. Therefore such an agreed key is of the form $\mathcal{H}(\alpha^{R_i R_j})$ for some R_i and R_j chosen at independently at random by entities i and j — this means that an adversary still faces the Diffie-Hellman style problem of working out $\alpha^{R_i R_j}$ from α^{R_i} and α^{R_j} to learn any information about the key. The fact that E later learns S_i is clearly of no help to her.

Note that the above argument includes the scenario in which E has learned some previous session keys as well as compromising S_i at some time. Of course, by this we mean that E still faces a tough problem learning any session key agreed before time τ that she has not discovered through other means (such as the `Reveal` query).

6.3 Unknown Key-Share

First we discuss the relevance of the *unknown key-share* attribute. Suppose E can coerce an entity i into holding session key κ in the belief that it is shared with entity e, and also coerce j to hold κ in the belief that it is shared with i. Now e can claim to i that any communications j sent to i using κ (e.g., a MACed message) in fact originated with e. Also, e can decrypt any messages that j encrypts using κ and sends to i. This kind of 'attack' has been proposed against a number of previous schemes. We propose this attribute for the first time as a 'generic' class of attacks.

Prevention of these unknown key-shares has also been built into the model that we've adopted. For suppose entity i can be coerced into sharing a key with e, when really i shares the key with j. In the model of distributed computing, this corresponds to a $\Pi_{i,e}$ oracle and a $\Pi_{j,i}$ oracle holding the same session key. An adversary E could therefore reveal the key held by $\Pi_{i,e}$, pick $\Pi_{j,i}$ to ask its `Test` query, and in this way defeat the AK or AKC protocol.

6.4 Key-Compromise Impersonation

Protocols 1, 2, and 3 fail to achieve the *key-compromise impersonation* attribute. That is, if entity i discloses its secret value S_i then an adversary E is not just able to impersonate i to any entity, but also can impersonate any entity j to i, since in this event E is able to compute the 'secret value' component $\alpha^{S_i S_j}$ involved in a run of the protocol between i and j. Procotol 4, on the other hand, would appear to possess the key-compromise attribute.

6.5 Loss of Information

Is it possible that loss of some information other than i's secret value can compromise the protocols? Unfortunately this is the case in each of Protocols 1, 2, and 3, since loss of $\alpha^{S_i S_j}$ certainly allows an adversary to impersonate i to j and vice versa. Thus none of these protocols remains secure if this supplementary information is lost. It is not clear what effect loss of other supplementary information might have — for example, if entity i were to inadvertently disclose some bits of S_i.

6.6 Message Independence

Both the AK protocols proposed attain message independence — that is in a bona fide run of the protocols, the individual flows are unrelated.

The AKC protocols, on the other hand, do not achieve message independence. This is unsurprising, since by definition the goal of key confirmation is similar to the goal of entity authentication. Flows sent by j in a protocol with such a goal necessarily contain information specific to this particular run which has been selected by i in order to prevent replay attacks. Thus while our AKC protocols do not achieve message independence, it appears that such a property is inherent to all protocols that achieve key confirmation.

7 Practicalities

7.1 'Real-World' Implications

What are the implications of these theoretical results to the 'real world'?

Until the recent advent of 'practice-oriented provable security', systems which offered any degree of provable security were impractical due to the large computational overheads incurred by their operation. As in [8, 9, 11, 27], this is not the case here. All the protocols in this paper are examples of the 'unified model' of key agreement, which it is our task to present. Practical implementations of the unified model are as efficient as any implementations used in practice; indeed the unified model is currently under consideration for standardization [2, 3, 21].

However, while the results of this paper ensure *theoretical* correctness of the protocols, the theoretical proofs take place in the random oracle model. Therefore the security of a *practical* implementation of any of the protocols relies on the ability of a hash function to instantiate a random oracle. The potential for such an instantiation to introduce weaknesses has led to criticism of the random oracle paradigm. Let us address some common concerns.

Firstly, as with all proofs in the random oracle model, our results guarantee security against *generic* attacks — attacks which do not exploit any special properties of the hash function instantiating the random oracle. It is precisely such generic attacks that have caused the downfall of many previous key agreement protocols. Let us therefore emphasize that such attacks are prohibited by our results within the model of distributed computing employed.

Secondly, let's consider the typical cost of generic attacks on key agreement protocols. In the case of signature and encryption schemes which employ hash functions, generic attacks usually carry a high computational overhead, so it is unclear whether a non-generic attack that exploits the structure of the hash function used will involve as much work as a generic attack. In contrast, generic attacks on key agreement protocols typically involve almost no computational burden — non-generic attacks on the other hand will require the traditionally greater computational expense of exploiting a weakness in the hash function being used.

Thirdly, let's examine the strength of the requirements made on the instantiation of the random oracle in implementations of the protocols. All the protocols essentially employ a random oracle mapping *fixed-length* inputs to *fixed-length* outputs. Therefore the security of the hash function is not stretched by the need to produce arbitrary length outputs. In most implementations a single application of the hash function chosen will produce sufficiently many output bits. Furthermore, the use of the hash function is likely to be infrequent, since the hash function needs to be used only once or twice each time a new key is agreed. This is in contrast to the use of hash functions in practical implementations of provably secure signature schemes or encryption schemes, where the frequent need to use the hash function makes its efficiency vital. In this instance, an implementation may choose to use a less efficient, but (supposedly) more secure hash function construct. As a concrete suggestion, hash functions and MACs whose construction employs an underlying block cipher could be chosen. Using such instantiations, the security of practical implementations of the protocols can be made to rest on, say, the security of DES and the Diffie-Hellman problem, both cryptographic schemes that have withstood 20 years of extensive investigation.

Finally, we must discuss the justification for using a hash function in a key agreement protocol (aside from its ability to facilitate provable security in the random oracle model). Traditionally, the most significant bits of a Diffie-Hellman number have been used as the agreed session key. While recent work [15] has shown that some of the most significant bits of a Diffie-Hellman number are as hard to compute as the entire number, it is not clear what this exact number of hard bits is. Moreover, it is not known whether these most significant bits are pseudorandom. Thus it seems sensible that a hash function should be used to 'distill' pseudorandomness from the whole Diffie-Hellman number. Hashing in this way also renders known key attacks less damaging — for while it may be unclear whether revealing previous Diffie-Hellman numbers calculated using S_i and S_j enables information about S_i and S_j or other session keys calculated using S_i and S_j to be inferred, the one-way property of hash functions adds to any confidence one may have that disclosure of previous session keys gives nothing away when the agreed value has been hashed to form the session key.

In summary, when employing the proposed protocols in practice, an implementor is assured that no subtle flaws exist in the form of the protocols, and further that an attack on their implementation is likely to incur the heavy computational burden associated with

breaking one of the underlying cryptographic primitives. No currently employed protocol can give such assurances, and as discussed in §1, a large number of flaws have been found in the protocols previously proposed. Thus, not only do our protocols provably achieve the goals of AK and AKC in the random oracle model, but in addition, practical implementations of the protocols that employ a hash function to instantiate the random oracle offer superior security assurances compared to any currently in use.

7.2 Implementation Issues

This subsection discusses some practical issues, such as efficiency, that may arise when implementing the protocols.

One issue is how to instantiate the random oracles. SHA-1 [20] should provide sufficient security for most applications. It can be used in various ways to provide instantiations of independent random oracles. For example, an implementation of Protocol 1 may choose to use:

$$\mathcal{H}_1(x) := \text{SHA-1}(01, x) \quad \text{and} \quad \mathcal{H}_2(x) := \text{SHA-1}(10, x) \ .$$

A particularly efficient instantiation of the random oracles used in Protocol 2 is possible using SHA-1 or RIPEMD-160 [19]. Suppose 80-bit session keys and MAC keys are required. Then the first 80 bits of SHA-1$(\alpha^{R_iR_j}, \alpha^{S_iS_j})$ can be used as κ' and the second 80 bits used as κ. Of course, such efficient implementations may not offer the highest conceivable security assurance of any instantiation.

It is easy to make bandwidth savings in implementations of the AKC protocols. Instead of sending the full authenticated messages (m, a) in flows 2 or 3, in both cases the entity can omit much of m, leaving the remainder of the message to be inferred by its recipient.

In some applications, it may not be desirable to carry out a protocol run each time a new session key is desired. Considering specifically Protocol 2 by way of example, entities may wish to compute the agreed key as:

$$\mathcal{H}_2(\alpha^{R_iR_j}, \alpha^{S_iS_j}, counter) \ .$$

Then instead of running the whole protocol each time a new key is desired, most of the time the counter is simply incremented. Entities need then only resort to using the protocol itself every now and then to gain some extra confidence in the 'freshness' of the session keys they're using.

In Protocols 1, 2, and 3, performance and security reasons may make it desirable to use a larger (and presumably more secure) group for the static Diffie-Hellman number $(\alpha_1^{S_iS_j})$ than for the ephemeral Diffie-Hellman number $(\alpha_2^{R_iR_j})$ calculation. The larger group is desirable because the static number will be used more often. The static numbers may be cached to provide a speed up in session key calculation.

Finally, note that a practical instantiation of $\mathcal{G}$ using certificates should check knowledge of the secret value before issuing a certificate on the corresponding public value. We believe that this is a sensible precaution in any implementation of a Certification Hierarchy.

8 Conclusions and Further Work

This paper has proposed formal definitions of secure AK and AKC protocols within a formal model of distributed computing. The 'unified model' of key agreement has been introduced, and several variants of this model have been demonstrated to provide provably secure AK and AKC protocols in the random oracle model. Strong evidence has been supplied that practical implementations of the protocols also offer superior security assurances than those currently in use, while maintaining similar computational overheads.

The definitions we have suggested for secure AK and AKC protocols are new, and the first question to ask is: are these the correct definitions for AK and AKC? We have supplied justification for the definitions we've chosen; further debate of the appropriateness of these definitions is clearly required.

A number of other questions are suggested by our results. Is the model of distributed computing adopted ideal? What impact do security proofs have on protocols? Can these methods be applied to protocols with different security goals? For which other goals would implementors like to see proven secure solutions? At a more concrete level: is it possible to remove, or at least minimize, the random oracle assumptions on which the security proofs rely? Can the reductions in the proofs be used to obtain meaningful measures of exact security [11]?

9 Acknowledgements

The authors are grateful to Karl Brincat, Mike Burmester, and Peter Wild for comments on an early draft of this work, and to Phil Rogaway for comments and an enlightening conversation at PKS'97. The authors also wish to thank Rick Ankney of CertCo for his contributions to the description of the unified model in X9.42.

References

[1] N. Alexandris, M. Burmester, V. Chrissikopoulos, and D. Peppes. Key agreement protocols: two efficient models for provable security. In S.K. Katsikas, D. Gritzalis, editors, *Information Systems Security, Facing the Information Society of the 21st century, IFIP SEC '96*, Chapman & Hall, pages 227–236, 1996.

[2] ANSI X9.42-1996: *Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Algorithm Keys Using Diffie-Hellman.* September 1996. Working Draft.

[3] ANSI X9.63-1997: *Public Key Cryptography for the Financial Services Industry: Elliptic Curve Key Agreement and Key Transport Protocols.* 1997. Working Draft.

[4] M. Bellare, R. Canetti, and H. Krawczyk. Keying hash functions for message authentication. In *Advances in Cryptology: Crypto '96*, pages 1–15, 1996.

[5] M. Bellare, R. Guerin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In *Advances in Cryptology: Crypto '95*, pages 15–28, 1995.

[6] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In *Advances in Cryptology: Crypto '94*, pages 341–358, 1994.

[7] M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Advances in Cryptology: Crypto '93*, pages 232–249, 1993. A full version of this paper is available at `http://www-cse.ucsd.edu/users/mihir`

[8] M. Bellare and P. Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security*, pages 62–73, 1993.

[9] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Advances in Cryptology: Eurocrypt '94*, pages 92–111, 1995.

[10] M. Bellare and P. Rogaway. Provably secure session key distribution—the three party case. In *Proceedings of the 27th ACM Symposium on the Theory of Computing*, pages 57–66, 1995.

[11] M. Bellare and P. Rogaway. The exact security of digital signatures – how to sign with RSA and Rabin. In *Advances in Cryptology: Eurocrypt '96*, pages 399–416, 1996.

[12] M. Bellare and P. Rogaway. Minimizing the use of random oracles in authenticated encryption schemes. In *Proceedings of PKS'97*, 1997.

[13] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In *Advances in Cryptology: Crypto '91*, pages 44–61, 1991.

[14] S. Blake-Wilson and A.J. Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. To appear in *Security Protocols Workshop '97*, Springer Verlag, 1997.

[15] D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *Advances in Cryptology: Crypto '96*, pages 129–142, 1996.

[16] M. Burmester. On the risk of opening distributed keys. In *Advances in Cryptology: Crypto '94*, pages 308–317, 1994.

[17] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6): 644–654, November 1976.

[18] W. Diffie, P.C. van Oorschot, and M.J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2: 107–125, 1992.

[19] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: a strengthened version of RIPEMD. In *Fast Software Encryption, Third International Workshop*, pages 71–82, 1996.

[20] FIPS 180-1. Secure hash standard. *Federal Information Processing Standards Publication 180-1*, 1995.

[21] IEEE P1363. *Standard for Public-Key Cryptography*. July 1997. Working Draft.

[22] M. Just and S. Vaudenay. Authenticated multi-party key agreement. In *Advances in Cryptology: Asiacrypt '96*, pages 36–49, 1996.

[23] T. Matsumoto, Y. Takashima, and H. Imai. On seeking smart public-key-distribution systems. *The Transactions of the IECE of Japan*, E69: 99–106, 1986.

[24] U.M. Maurer and S. Wolf. Diffie-Hellman oracles. In *Advances in Cryptology: Crypto '96*, pages 268–282, 1996.

[25] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*, chapter 12. CRC Press, 1996.

[26] J.H. Moore. Protocol failure in cryptosystems. Chapter 11 in *Contemporary Cryptology: the Science of Information Integrity*, G. J. Simmons, editor, pages 541–558, IEEE Press, 1992.

[27] D. Pointcheval and J. Stern. Security proofs for signature schemes. In *Advances in Cryptology: Eurocrypt '96*, pages 387–398, 1996.

A Proof of Theorem 8

Theorem 13 (8) Protocol 1 is a secure AKC protocol provided the DHS and MAC are secure and $\mathcal{H}_1$ and $\mathcal{H}_2$ are independent random oracles.

Proof. We deal with each condition of Definition 3 in turn.

Conditions 1 and 2: The first two conditions follow immediately from the description of P and the assumption that $\mathcal{H}_2$ is a random oracle.

Condition 3: Let's turn to the third condition. Consider an arbitrary adversary E, and suppose, by way of contradiction, that $Pr[\texttt{No-Matching}^E(k)]$ is non-negligible. We say that E *succeeds* if at the end of E's experiment, there exists an oracle $\Pi^s_{i,j}$ $(i,j \notin \mathcal{C})$ which has accepted but no $\Pi_{j,i}$ oracle has had a matching conversation to $\Pi^s_{i,j}$. Further in this case we say that E has succeeded *against* $\Pi^s_{i,j}$. Hence, by assumption:

$$Pr[E \text{ succeeds}] = n(k)$$

for some non-negligible $n(k)$ by assumption. Now call A_k the event that, during E's experiment, there exists a pair $i,j \in I$ with $i,j \notin \mathcal{C}$ for which $\alpha^{S_i S_j}$ is queried of $\mathcal{H}_1$ either by E or by any oracle except $\Pi_{i,j}$ or $\Pi_{j,i}$ oracles.

Case 1: Suppose that $Pr[A_k] = n_1(k)$ is non-negligible. In this case we construct from E an adversary F of the DHS that wins its experiment with non-negligible probability.
F's operation: F takes as input (p, q, α) generated by $\mathcal{G}_{DH}(1^k)$ and $(\alpha^{S'}, \alpha^{S''})$ for S', S'' chosen at random from $\mathbb{Z}_q^*$, and must try to guess $\alpha^{S'S''}$ (cf. Definition 7).

F picks at random a pair $i, j \in I$, guessing that E or an oracle other than a $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle will query $\mathcal{H}_1$ with $\alpha^{S_i S_j}$. F must perform E's experiment. Instead of running $\mathcal{G}$, F makes its own input (p, q, α) the global parameters to be used by P, and chooses all entities' secret values at random itself, except for i's and j's. F makes $\alpha^{S'}$ i's public value (so $S_i = S'$) and $\alpha^{S''}$ j's public value (so $S_j = S''$), forms the directory *public-info*, and starts E.

F must answer all the oracle queries involved in E's experiment. F answers all distinct $\mathcal{H}_1$ and $\mathcal{H}_2$ queries at random (just as a real random oracle would). F answers all E's `Reveal` queries as specified by Π, all E's `Send` queries as specified by Π except for `Send` queries to $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles, and all E's `Corrupt` queries as specified by Π provided neither i nor j is being corrupted.

When E asks a $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle a `Send` query, instead of computing $\mathcal{H}_1(\alpha^{S_i S_j})$ and using this as the long-term MAC key shared by the two entities, F picks a key κ' at random from $\{0,1\}^k$ to 'represent' $\mathcal{H}_1(\alpha^{S_i S_j})$ (of course, F doesn't know $\alpha^{S_i S_j}$ — that's why there's a problem!). F then uses κ' when determining these oracles responses.

If E asks a `Corrupt` query to i or j then F gives up.

Now let $T_3(k)$ denote a polynomial bound on the number of distinct $\mathcal{H}_1$ calls made by E and its oracles during the experiment. F picks $l \in_R \{1, \ldots, T_3(k)\}$, guessing that the lth distinct call made to $\mathcal{H}_1$ by E or any oracle (except for $\Pi_{i,j}$ or $\Pi_{j,i}$ oracles) will be on $\alpha^{S_i S_j}$. When the lth distinct $\mathcal{H}_1$ call is made (say on g), F stops and outputs g as its guess at $\alpha^{S_i S_j}$.

If E halts before the lth distinct $\mathcal{H}_1$ query is made, F gives up.

Only one problem remains. $\mathcal{H}_1$ may have been queried on $\alpha^{S_i S_j}$ at some time before the lth distinct $\mathcal{H}_1$ call. In this case, F will have answered the call at random, and its answer may have been in contradiction to the key κ' it's using to represent $\mathcal{H}_1(\alpha^{S_i S_j})$. The problem is that E is not guaranteed to halt in this eventuality. To sidestep this potential problem, let $T_4(k)$ denote a polynomial bound on E's runtime under ordinary circumstances. If F runs E for longer than $T_4(k)$, F gives up, concluding that it must have missed an $\mathcal{H}_1$ query on $\alpha^{S_i S_j}$.
Analysis: Observe that if the lth distinct $\mathcal{H}_1$ query made by E or its oracles is on $\alpha^{S_i S_j}$, then F certainly wins its experiment. We conclude that the probability F outputs the correct value $g = \alpha^{S_i S_j}$ is at least:

$$\frac{n_1(k)}{(T_1(k))^2 T_3(k)}$$

which is non-negligible. This contradicts the assumed security of the DHS. We conclude that $n_1(k)$ is negligible.

Case 2: Let $n_2(k)$ be the probability that E succeeds against at least one initiator oracle, and $n_3(k)$ be the probability that E succeeds against at least one responder oracle but no initiator oracles. We have:

$$n(k) = n_2(k) + n_3(k) \quad .$$

So there are two subcases to consider.

Case 2(a): Suppose $n_2(k)$ is non-negligible. In this case we construct from E an adversary F of the MAC.

F's operation: F has access to a MACing oracle that computes MACs under a key κ'' which was chosen at random from $\{0,1\}^k$. F's task is to compute a valid authenticated message (m, a), where m was not queried of its oracle (cf. Definition 5).

F performs E's experiment. F runs $\mathcal{G}$ on input 1^k — $\mathcal{G}$ chooses a parameter set (p, q, α) and secret values for all the entities. $\mathcal{G}$ calculates all public values and forms the directory *public-info*.

F now starts E on input 1^k and *public-info*. F picks $i, j \in_R I$ and $s \in_R \{1, \ldots, T_2(k)\}$, guessing that E will succeed against initiator $\Pi^s_{i,j}$ oracle.

F answers all E's queries itself. To answer queries of $\mathcal{H}_1$ and $\mathcal{H}_2$, F itself picks replies at random, with the exception of $\mathcal{H}_1$ queries on $\alpha^{S_i S_j}$ (note that F can compute $\alpha^{S_i S_j}$). If $\mathcal{H}_1$ is queried on $\alpha^{S_i S_j}$ by E or an oracle that's not a $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle, then F gives up. F's actions when $\mathcal{H}_1$ is queried on $\alpha^{S_i S_j}$ by a $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle are specified below.

F answers E's `Reveal` queries and `Corrupt` queries as specified by Π. However if E asks i or j a `Corrupt` query F gives up.

F also answers `Send` queries not sent to $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles as specified by Π. To answer `Send` queries of $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles, F answers as specified by Π, except that instead of calculating $\kappa' = \mathcal{H}_1(\alpha^{S_i S_j})$ and using this key to MAC messages, F calls its own MACing oracle to compute its response. (F thus implicitly uses κ'' to 'represent' κ'.) F therefore needs to call its MACing oracle to calculate flows on behalf of $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles, and also to decide whether or not such oracles should accept.

If E does not invoke $\Pi^s_{i,j}$ as an initiator oracle, then F gives up.

On the other hand, if E does invoke $\Pi^s_{i,j}$ as an initiator oracle, then at some time τ_0, $\Pi^s_{i,j}$ receives λ and responds with α^{R_i}. If $\Pi^s_{i,j}$ does not at some later time receive a flow of the form (m, a) where $m = (2, j, i, \alpha^{R_j}, \alpha^{R_i})$ for some α^{R_j}, then F gives up.

However, if $\Pi^s_{i,j}$ is to accept, it must later receive a flow of this form. In this event, provided F has not called its MACing oracle previously on m, then F stops and outputs (m, a) as its guess at a valid forgery. If F has previously called its MACing oracle to compute the flow then F gives up.

Analysis: Suppose E does succeed against initiator $\Pi^s_{i,j}$. In this event, F outputs a valid forgery and wins its experiment, provided E or some other oracle (except $\Pi_{i,j}$ or $\Pi_{j,i}$ oracles) has not called $\mathcal{H}_1$ on $\alpha^{S_i S_j}$, and provided F has not previously calculated the flow that makes $\Pi^s_{i,j}$ accept on behalf of some $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle.

Certainly, by Case 1, the probability that $\mathcal{H}_1$ has been called on $\alpha^{S_i S_j}$ is negligible.

Furthermore, the probability that F has called its MACing oracle to produce the flow is also negligible. For F could only have called on this message on behalf of a responder $\Pi^t_{j,i}$ which received α^{R_i} as its own first flow, or on behalf of an initiator $\Pi^u_{i,j}$ with $u \neq s$ which also chose α^{R_i} and needs to decide whether or not it should accept. The probability the call was made by a responder $\Pi^t_{j,i}$ before τ_0 is negligible since R_i was chosen at random (note that $\frac{1}{q-1}$ is certainly negligible, since the DHS is secure), and if the call was made after τ_0, then $\Pi^t_{j,i}$ has had a matching conversation to $\Pi^s_{i,j}$. The probability the call was made by

$\Pi^u_{i,j}$ is negligible since in this event, $\Pi^u_{i,j}$ and $\Pi^s_{i,j}$ have independently chosen the same R_i.

We conclude that F constructed in this way wins its experiment with probability at least:

$$\frac{n_2(k)}{(T_1(k))^2 T_2(k)} - \lambda(k)$$

for some negligible $\lambda(k)$ — this is still non-negligible, and therefore contradicts the assumed security of the MAC. Thus $n_2(k)$ must be negligible.

Case 2(b): Suppose $n_3(k)$ is non-negligible. Again we construct from E an adversary F of the MAC.

F's operation: The operation of F is similar to the operation of the MAC adversary constructed during Case 2(a), except that this time, F picks $i, j \in_R I$ and $t \in_R \{1, \ldots, T_2(k)\}$, guessing that E will succeed against responder $\Pi^t_{j,i}$ oracle and not succeed against any initiator oracles.

F answers queries just like the previous adversary we constructed — calling its own MACing oracle as necessary to answer `Send` queries to $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles.

This time, if E does not invoke $\Pi^t_{j,i}$ as a responder oracle, or if E succeeds against some initiator oracle, then F gives up.

On the other hand, if E does invoke $\Pi^t_{j,i}$ as a responder oracle, then at some time τ_1 the oracle must receive α^{R_i} for some R_i, and reply with:

$$MAC_{\kappa''}(2, j, i, \alpha^{R_j}, \alpha^{R_i})$$

for some $R_j \in_R \mathbb{Z}^*_q$ (with the MAC actually computed by F's MACing oracle).

If $\Pi^t_{j,i}$ does not at some later time $\tau_3 > \tau_1$ receive a message of the form (m, a) with $m = (3, i, j, \alpha^{R_i}, \alpha^{R_j})$, then F gives up.

However, if $\Pi^t_{j,i}$ is to accept, it must later receive a flow of this form. If F has not previously called its MACing oracle on m, then F outputs (m, a) as its guess at a valid forgery. If F has already made a call on m, then F gives up.

Analysis: Suppose E does succeed against responder $\Pi^t_{j,i}$ and against no initiator oracles. In this event, F outputs a valid forgery and wins its experiment, provided E or some other oracle (except $\Pi_{i,j}$ or $\Pi_{j,i}$ oracles) has not called $\mathcal{H}_1$ on $\alpha^{S_i S_j}$, and provided F has not previously calculated the flow that makes $\Pi^t_{j,i}$ accept on behalf of some $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle.

Certainly, by Case 1, the probability that $\mathcal{H}_1$ has been called on $\alpha^{S_i S_j}$ is negligible.

Furthermore, the probability that F has called its MACing oracle to produce the flow is also negligible. For F could only have called on this message on behalf of an initiator $\Pi^s_{i,j}$ which sent α^{R_i} as its own first flow, or on behalf of a responder $\Pi^u_{j,i}$ with $u \neq t$ which also chose α^{R_j} and needs to decide whether or not it should accept. The probability the call was made by an initiator $\Pi^s_{i,j}$ is negligible since such a $\Pi^s_{i,j}$ has accepted, so by assumption there exists $\Pi^v_{j,i}$ which has had a matching conversation to $\Pi^s_{i,j}$. The probability $v \neq t$ is negligible, since then $\Pi^v_{j,i}$ and $\Pi^t_{j,i}$ have independently chosen the same R_j, and $v = t$ is excluded, since then $\Pi^s_{i,j}$ has had a matching conversation to $\Pi^t_{j,i}$. The probability the call was made by $\Pi^u_{j,i}$ is negligible since in this event, $\Pi^u_{j,i}$ and $\Pi^t_{j,i}$ have again independently chosen the same R_j.

We conclude that F constructed in this way wins its experiment with probability at least:

$$\frac{n_3(k)}{(T_1(k))^2 T_2(k)} - \lambda(k)$$

for some negligible $\lambda(k)$ — this is still non-negligible, and therefore contradicts the assumed security of the MAC. Thus $n_3(k)$ must be negligible.

Together Cases 2(a) and 2(b) contradict the assumption that $n(k)$ is non-negligible. We conclude that $Pr[\texttt{No-Matching}^E(k)]$ is negligible for all adversaries E.

Condition 4: We argue by contradiction. Fix an arbitrary adversary E and suppose that $advantage^E(k)$ is non-negligible. We say that E succeeds (against $\Pi^s_{i,j}$) if E picks $\Pi^s_{i,j}$ to ask its `Test` query and outputs the correct bit `Guess`. Thus

$$Pr[E \text{ succeeds}] = \tfrac{1}{2} + n(k)$$

for some non-negligible $n(k)$ by assumption. Now call A_k the event that E picks some $\Pi^s_{i,j}$ oracle to ask its `Test` query such that some $\Pi^t_{j,i}$ oracle has had a matching conversation to $\Pi^s_{i,j}$. Clearly

$$Pr[E \text{ succeeds}] = Pr[E \text{ succeeds}|A_k]Pr[A_k] + Pr[E \text{ succeeds}|\overline{A}_k]Pr[\overline{A}_k] \ .$$

Condition 3 ensures that $Pr[\overline{A}_k] = \lambda(k)$ is negligible. Hence

$$\tfrac{1}{2} + n(k) \leq Pr[E \text{ succeeds}|A_k]Pr[A_k] + \lambda(k) \ .$$

Therefore $Pr[A_k] = 1 - \lambda(k)$ and

$$Pr[E \text{ succeeds}|A_k] = \tfrac{1}{2} + n_1(k)$$

for some non-negligible $n_1(k)$. Now, given event A_k, the key held by $\Pi^s_{i,j}$ will be of the form $\mathcal{H}_2(\alpha^{R_i R_j})$ for R_i chosen at random by $\Pi^s_{i,j}$ and R_j chosen at random by $\Pi^t_{j,i}$. Call B_k the event that $\mathcal{H}_2$ has been queried on $\alpha^{R_i R_j}$ by E or some oracle other than $\Pi^s_{i,j}$ or $\Pi^t_{j,i}$. Then

$$Pr[E \text{ succeeds}|A_k] = Pr[E \text{ succeeds}|A_k \wedge B_k]Pr[B_k|A_k] + Pr[E \text{ succeeds}|A_k \wedge \overline{B}_k]Pr[\overline{B}_k|A_k].$$

Since $\mathcal{H}_2$ is a random oracle, and $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ remain unopened by definition, $Pr[E \text{ succeeds}|\overline{B}_k \wedge A_k] = \frac{1}{2}$. Thus

$$\tfrac{1}{2} + n_1(k) \leq Pr[E \text{ succeeds}|A_k \wedge B_k]Pr[B_k|A_k] + \tfrac{1}{2}$$

so that $Pr[B_k|A_k] \geq n_1(k)$. We conclude that given E picks some $\Pi^s_{i,j}$ for which there exists some $\Pi^t_{j,i}$ that has had a matching conversation to $\Pi^s_{i,j}$, then the probability that $\mathcal{H}_2$ has previously been queried on $\alpha^{R_i R_j}$ by E or some oracle other than $\Pi^s_{i,j}$ or $\Pi^t_{j,i}$ is non-negligible.

Therefore we use E to construct an adversary F of the DHS.
F's operation: F takes as input (p, q, α) generated by $\mathcal{G}_{DH}(1^k)$ and $(\alpha^{R'}, \alpha^{R''})$ for R', R'' chosen an random from $\mathbb{Z}^*_q$, and must try to guess $\alpha^{R'R''}$. F makes (p, q, α) the global

parameters for P and picks all entities' secret values at random. F forms the directory *public-info* and starts E.

Now F picks $i, j \in_R I$ and $s, t \in_R \{1, \ldots, T_2(k)\}$, guessing that E will select $\Pi^s_{i,j}$ to ask its `Test` query after $\Pi^t_{j,i}$ has had a matching conversation to $\Pi^s_{i,j}$.

F now answers all $\mathcal{H}_1$ and $\mathcal{H}_2$ oracle queries at random, just like a real random oracle would.

F answers `Corrupt` queries as specified by Π, except that if E asks i or j a `Corrupt` query, F gives up.

F also answers `Reveal` queries as specified by Π, except that if E asks $\Pi^s_{i,j}$ or $\Pi^t_{j,i}$ a `Reveal` query, then F gives up.

Finally, F also answers all `Send` queries as specified by Π, except for `Send` queries to $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$. When E asks $\Pi^s_{i,j}$ its first `Send` query, instead of taking a random sample to form its challenge, $\Pi^s_{i,j}$ chooses $\alpha^{R'}$ (so $R_i = R'$). Similarly, F has $\Pi^t_{j,i}$ choose $\alpha^{R''}$ (so $R_j = R''$).

If E does not make its queries in such a way that $\Pi^t_{j,i}$ has a matching conversation to $\Pi^s_{i,j}$, then F gives up. On the other hand, if E does make its queries in this way, then $\Pi^s_{i,j}$ will accept (holding the key $\mathcal{H}_2(\alpha^{R_i R_j})$, although of course F doesn't know $\alpha^{R_i R_j}$ and so can't actually compute this key).

Now let $T_3(k)$ denote a polynomial bound on the number of distinct $\mathcal{H}_2$ queries made by E and its oracles. F picks $l \in_R \{1, \ldots, T_3(k)\}$, guessing that the lth distinct $\mathcal{H}_2$ call made during the experiment will be on $\alpha^{R_i R_j}$. When the lth distinct $\mathcal{H}_2$ call is made (say on g), then F stops and outputs g as its guess at $\alpha^{R_i R_j}$.

If E and its oracles do not make l distinct $\mathcal{H}_2$ oracle calls before E asks its `Test` query, then F gives up.

Analysis: Suppose E does pick $\Pi^s_{i,j}$ to ask its `Test` query after $\Pi^t_{j,i}$ has had a matching conversation to $\Pi^s_{i,j}$. Then, as we have seen, with non-negligible probability, E or some other oracle has called $\mathcal{H}_2$ on $\alpha^{R_i R_j}$. Hence the probability that F succeeds is at least:

$$\frac{n_1(k)}{(T_1(k))^2 (T_2(k))^2 T_3(k)} - \mu(k)$$

for some negligible $\mu(k)$ — this is still non-negligible, and therefore contradicts the assumed security of the DHS. We conclude that $n_1(k)$ must be negligible, and thus that $advantage^E(k)$ must be negligible. □

B Proof of Theorem 9

B.1 Preliminaries

Before we give the proof itself, some preliminaries are required. The first is the definition of a modified MAC adversary called a $*$-adversary. Roughly speaking, the $*$-adversary has access to a MACing oracle which MACs messages under an element of an ordered list $L = [\kappa_1, \kappa_2, \ldots, \kappa_l]$ of keys chosen independently at random. The list is unknown to the $*$-adversary, who can however specify which element in the list the oracle should use to MAC a particular message. The $*$-adversary's goal is to forge a valid tag on any message

(that has not yet been authenticated) under some key $\kappa_i \in L$, where i is also known to the $*$-adversary. This idea is made precise below.

A $*$-*adversary* $\overline{F}$ of a MAC is a probabilistic polytime algorithm with access to a MACing oracle. The MACing oracle is supplied with a *private* random oracle $\mathcal{P}$ (to which the adversary has only indirect access through its queries to the MACing oracle). $\overline{F}$'s queries to its MACing oracle take the form (s, m) where s is a seed to be used by the MACing oracle to compute a key using $\mathcal{P}$, and m is the message to be MACed. To answer a query (s, m), the MACing oracle first calls $\mathcal{P}$ on input s, to get:

$$\kappa' = \mathcal{P}(s) \ .$$

It then calculates:

$$MAC_{\kappa'}(m) = (m, a)$$

and returns (s, m, a) to $\overline{F}$. The output of $\overline{F}$ is now a triple (s, m, a) such that $\overline{F}$ has not previously queried its MACing oracle on (s, m).

Definition 14 (5') A MAC is a $*$-*secure MAC* if for every $*$-adversary $\overline{F}$ of the MAC, the function $\epsilon(k)$ defined by

$$\epsilon(k) = Pr[\mathcal{P} \leftarrow 2^{\infty}; (s, m, a) \leftarrow \overline{F} : MAC_{\kappa'}(m) = (m, a) \ \text{ where } \kappa' = \mathcal{P}(s)]$$

is negligible.

It is easy to see that $*$-secure MACs are also secure. The following lemma shows that $*$-secure MACs and secure MACs are equivalent.

Lemma 15 If a MAC is secure, then it is also $*$-secure.

Proof. Suppose, by way of contradiction, that $\overline{F}$ is a $*$-adversary of a secure MAC which succeeds with non-negligible probability. We use $\overline{F}$ to build an (ordinary) adversary F of the MAC. By assumption

$$Pr[\overline{F} \text{ succeeds}] = n(k)$$

for some non-negligible $n(k)$.

Let $T_3(k)$ denote a polynomial bound on the number of distinct seeds on which $\overline{F}$ makes a MAC query. Provided the message space is non-trivial, we can assume without loss of generality that $\overline{F}$ always makes a MAC call during its experiment on the seed s that forms part of its eventual output (s, m, a).

F's operation: F must perform $\overline{F}$'s experiment. F picks $l \in_R \{1, \ldots, T_3(k)\}$, guessing that $\overline{F}$'s eventual output will take the form (s, m, a) where s is the lth distinct seed on which $\overline{F}$ queried its MACing oracle.

F answers $\overline{F}$'s queries as follows. If $\overline{F}$ queries (s', m') where s' is not the lth distinct seed queried by $\overline{F}$, then F picks a key κ'' at random to represent $\mathcal{P}(s')$, and returns

$$(s', m', a') \ \text{ where } (m', a') = MAC_{\kappa''}(m') \ .$$

If $\overline{F}$ queries (s, m) where s is the lth distinct seed queried by $\overline{F}$, then F calls its own MACing oracle on m, receiving (m, a) in return. F then answers $\overline{F}$'s query with:

$$(s, m, a) \ .$$

If $\overline{F}$ does not make queries on l distinct seeds, then F gives up.

Otherwise, $\overline{F}$ outputs (σ, μ, α). If $\sigma = s$, then F outputs (μ, α) as its guess at a valid authenticated message. Otherwise, if $\sigma \neq s$, F gives up.

Analysis: If $\overline{F}$ does indeed output a 'good' triple (σ, μ, α) where σ was the lth distinct seed on which $\overline{F}$ queried its oracle, then F certainly succeeds. Therefore the probability that F succeeds is at least

$$\frac{n(k)}{T_3(k)}$$

which is still non-negligible. This contradicts the assumed security of the MAC. □

During the main proof that Protocol 2 is secure, we will in some cases show that a successful adversary E of P can be used to build a successful $*$-adversary $\overline{F}$ of the MAC. Lemma 15 above demonstrates that the existence of such an $\overline{F}$ which succeeds with non-negligible probability contradicts the assumed security of the MAC.

B.2 The proof

Recall that in Protocol 2, keys are formed as $\mathcal{H}_1(\alpha^{R_i R_j}, \alpha^{S_i S_j})$ and $\mathcal{H}_2(\alpha^{R_i R_j}, \alpha^{S_i S_j})$. In what follows, it will sometimes be helpful to think of $\mathcal{H}_1$ and $\mathcal{H}_2$ as taking two distinct inputs — the first $\alpha^{R_i R_j}$ and the second $\alpha^{S_i S_j}$. This is certainly 'well-defined', since we have stipulated that the encodings used are unique.

Theorem 16 (9) Protocol 2 is a secure AKC protocol provided the DHS and MAC are secure and $\mathcal{H}_1$ and $\mathcal{H}_2$ are independent random oracles.

The proof of this theorem is in many respects analogous to the proof of Theorem 8. We draw the reader's attention to this analogy in case it is helpful to compare the two proofs.

Proof. Again, take each condition of Definition 3 in turn.

Conditions 1 and 2: The first two conditions follow immediately from the description of P and the assumption that $\mathcal{H}_2$ is a random oracle.

Condition 3: Consider an arbitrary adversary E, and suppose $Pr[\texttt{No-Matching}^E(k)] = n(k)$ is non-negligible. Call A_k the event that during E's experiment, there exists a pair $i, j \in I$ with $i, j \notin \mathcal{C}$ for which $\mathcal{H}_1$ or $\mathcal{H}_2$ is queried with $\alpha^{S_i S_j}$ as second input either by E or by any oracle except $\Pi_{i,j}$ or $\Pi_{j,i}$ oracles.

Case 1: Suppose that $Pr[A_k] = n_1(k)$ is non-negligible. E can be used to construct an adversary F of the DHS that wins its experiment with non-negligible probability.

F's operation: F takes as input (p, q, α) and $(\alpha^{S'}, \alpha^{S''})$ and must try to guess $\alpha^{S'S''}$.

F picks a pair $i, j \in_R I$, guessing that E or an oracle other than a $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle will query $\mathcal{H}_1$ or $\mathcal{H}_2$ with $\alpha^{S_i S_j}$ as the second input. F performs E's experiment — making

(p, q, α) the global parameters for P, and choosing all entities' secret values at random, except for i's and j's. F makes $\alpha^{S'}$ i's public value (so $S_i = S'$) and $\alpha^{S''}$ j's public value (so $S_j = S''$). F starts E.

F answers all $\mathcal{H}_1$ and $\mathcal{H}_2$ queries at random just like a real random oracle would. F answers all `Send` and `Reveal` queries as specified by Π, except for these queries made to $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles. F also answers all `Corrupt` queries not made to i or j as specified by Π.

If E asks i or j a `Corrupt` query, then F gives up.

When E asks a $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle a `Send` query, instead of computing the values $\mathcal{H}_1(\alpha^{R_i R_j}, \alpha^{S_i S_j})$ and $\mathcal{H}_2(\alpha^{R_i R_j}, \alpha^{S_i S_j})$ and using them as the keys used by the oracle, F must pick κ' and κ at random to 'represent' $\mathcal{H}_1(\alpha^{R_i R_j}, \alpha^{S_i S_j})$ and $\mathcal{H}_2(\alpha^{R_i R_j}, \alpha^{S_i S_j})$ respectively, since F doesn't actually know $\alpha^{S_i S_j}$. F then uses κ' and κ when determining the oracle's actions.

If F asks a `Reveal` query to a $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle, then of course instead of revealing $\mathcal{H}_2(\alpha^{R_i R_j}, \alpha^{S_i S_j})$, the oracle must reveal the κ F has chosen to represent $\mathcal{H}_2(\alpha^{R_i R_j}, \alpha^{S_i S_j})$.

Now let $T_3(k)$ denote a polynomial bound on the number of calls made by E and its oracles to $\mathcal{H}_1$ and $\mathcal{H}_2$ that have distinct second inputs. F picks $l \in_R \{1, \ldots, T_3(k)\}$, guessing that the lth distinct second input on which $\mathcal{H}_1$ or $\mathcal{H}_2$ are queried will be $\alpha^{S_i S_j}$. When the call is made on the lth distinct second input to either $\mathcal{H}_1$ or $\mathcal{H}_2$ (say on g), F stops and outputs g as its guess at $\alpha^{S_i S_j}$.

If E halts before the lth distinct second input is queried, F gives up.

As before, one problem remains. $\mathcal{H}_1$ or $\mathcal{H}_2$ may have been queried with $\alpha^{S_i S_j}$ as second input at some time before the lth distinct second input call. In this case, F will have answered at random, and its answer may have been in contradiction to one of the keys that has been used by some $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle. The problem is that E is not guaranteed to halt in this eventuality. To sidestep this potential problem, let $T_4(k)$ denote a polynomial bound on E's runtime under ordinary circumstances. If F runs E for longer than $T_4(k)$, F gives up, concluding that it must have missed an $\mathcal{H}_1$ or $\mathcal{H}_2$ query with second input $\alpha^{S_i S_j}$.
Analysis: Observe that if the lth distinct $\mathcal{H}_1$ or $\mathcal{H}_2$ second input query made by E or its oracles is on $\alpha^{S_i S_j}$, then F certainly wins its experiment. We conclude that the probability F outputs the correct value $g = \alpha^{S_i S_j}$ is at least:

$$\frac{n_1(k)}{(T_1(k))^2 T_3(k)}$$

which is non-negligible. This contradicts the assumed security of the DHS. We conclude that $n_1(k)$ is negligible.

Case 2: Let $n_2(k)$ be the probability that E succeeds against at least one initiator oracle, and $n_3(k)$ be the probability that E succeeds against at least one responder oracle but no initiator oracles. We have:

$$n(k) = n_2(k) + n_3(k) \ .$$

So there are two subcases to consider.

Case 2(a): Suppose $n_2(k)$ is non-negligible. In this case we construct from E a $*$-adversary $\overline{F}$ of the MAC.

$\overline{F}$'s operation: $\overline{F}$ performs E's experiment. $\overline{F}$ runs $\mathcal{G}$ on input 1^k — $\mathcal{G}$ chooses a parameter set (p, q, α) and secret values for all the entities. $\mathcal{G}$ calculates all public values and forms the directory *public-info*.

$\overline{F}$ now starts E on input 1^k and *public-info*. $\overline{F}$ picks $i, j \in_R I$ and $s \in_R \{1, \ldots, T_2(k)\}$, guessing that E will succeed against initiator $\Pi^s_{i,j}$ oracle.

$\overline{F}$ answers all E's queries itself. To answer queries of $\mathcal{H}_1$ and $\mathcal{H}_2$, $\overline{F}$ itself picks replies at random, with the exception of $\mathcal{H}_1$ queries with second input $\alpha^{S_i S_j}$. If $\mathcal{H}_1$ is queried on second input $\alpha^{S_i S_j}$ by E or an oracle that's not a $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle, then $\overline{F}$ gives up. F's actions when $\mathcal{H}_1$ is queried on second input $\alpha^{S_i S_j}$ by a $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle are specified below.

$\overline{F}$ answers E's `Reveal` queries and `Corrupt` queries as specified by Π. However if E asks i or j a `Corrupt` query $\overline{F}$ gives up.

$\overline{F}$ also answers `Send` queries not sent to $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles as specified by Π. To answer `Send` queries of $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles, $\overline{F}$ answers as specified by Π, except that instead of calculating $\kappa' = \mathcal{H}_1(\alpha^{R_i R_j}, \alpha^{S_i S_j})$ each time and using this key to MAC messages, $\overline{F}$ calls its own MACing oracle on the message under the seed $s = \alpha^{R_i R_j}$ to compute its response. ($\overline{F}$ is thus implicitly using $\kappa'' = \mathcal{P}(\alpha^{R_i R_j})$ to 'represent' κ'.) $\overline{F}$ therefore needs to call its MACing oracle to calculate flows on behalf of $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles, and also to decide whether or not such oracles should accept. Note that provided $\mathcal{H}_1$ is only called by $\Pi_{i,j}$ and $\Pi_{j,i}$ on second input $\alpha^{S_i S_j}$, then $\mathcal{P}(s) = \mathcal{H}_1(s, \alpha^{S_i S_j})$ essentially forms a private random oracle shared by $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles.

If E does not invoke $\Pi^s_{i,j}$ as an initiator oracle, then $\overline{F}$ gives up.

On the other hand, if E does invoke $\Pi^s_{i,j}$ as an initiator oracle, then at some time τ_0, $\Pi^s_{i,j}$ receives λ and responds with α^{R_i}. If $\Pi^s_{i,j}$ does not at some later time receive a flow of the form (m, a) where $m = (2, j, i, \alpha^{R_j}, \alpha^{R_i})$ for some α^{R_j}, then $\overline{F}$ gives up.

However, if $\Pi^s_{i,j}$ is to accept, it must later receive a flow of this form. In this event, provided $\overline{F}$ has not called its MACing oracle previously on m under the seed $s = \alpha^{R_i R_j}$, then $\overline{F}$ stops and outputs (s, m, a) as its guess at a valid forgery. If $\overline{F}$ has previously called its MACing oracle to compute the flow under this seed then $\overline{F}$ gives up.

Analysis: Suppose E does succeed against initiator $\Pi^s_{i,j}$. In this event, $\overline{F}$ outputs a valid forgery and wins its experiment, provided E or some other oracle has not called $\mathcal{H}_1$ on second input $\alpha^{S_i S_j}$, and provided $\overline{F}$ has not previously calculated the flow that makes $\Pi^s_{i,j}$ accept on behalf of some $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle.

Certainly, by Case 1, the probability that $\mathcal{H}_1$ has been called on second input $\alpha^{S_i S_j}$ is negligible.

Furthermore, the probability that $\overline{F}$ has called its MACing oracle to produce the flow is also negligible. For $\overline{F}$ could only have called on this message on behalf of a responder $\Pi^t_{j,i}$ which received α^{R_i} as its own first flow, or on behalf of an initiator $\Pi^u_{i,j}$ with $u \neq s$ which also chose α^{R_i} and needs to decide whether or not it should accept. The probability the call was made by a responder $\Pi^t_{j,i}$ before τ_0 is negligible since R_i was chosen at random, and if the call was made after τ_0, then $\Pi^t_{j,i}$ has had a matching conversation to $\Pi^s_{i,j}$. The probability the call was made by $\Pi^u_{i,j}$ is negligible since in this event, $\Pi^u_{i,j}$ and $\Pi^s_{i,j}$ have independently chosen the same R_i.

We conclude that $\overline{F}$ constructed in this way wins its experiment with probability at

least:

$$\frac{n_2(k)}{(T_1(k))^2 T_2(k)} - \lambda(k)$$

for some negligible $\lambda(k)$ — this is still non-negligible, and therefore contradicts the assumed security of the MAC. Thus $n_2(k)$ must be negligible.

Case 2(b): Suppose $n_3(k)$ is non-negligible. Again we construct from E a $*$-adversary $\overline{F}$ of the MAC.

$\overline{F}$'s operation: The operation of $\overline{F}$ is similar to the operation of the MAC $*$-adversary constructed during Case 2(a), except that this time, $\overline{F}$ picks $i, j \in_R I$ and $t \in_R \{1, \ldots, T_2(k)\}$, guessing that E will succeed against responder $\Pi^t_{j,i}$ oracle and not succeed against any initiator oracles.

$\overline{F}$ answers queries just like the previous adversary we constructed — calling its own MACing oracle as necessary to answer `Send` queries to $\Pi_{i,j}$ and $\Pi_{j,i}$ oracles.

This time, if E does not invoke $\Pi^t_{j,i}$ as a responder oracle, or if E succeeds against some initiator oracle, then $\overline{F}$ gives up.

On the other hand, if E does invoke $\Pi^t_{j,i}$ as a responder oracle, then at some time τ_1 the oracle must receive α^{R_i} for some R_i, and reply with:

$$MAC_{\kappa''}(2, j, i, \alpha^{R_j}, \alpha^{R_i})$$

for some $R_j \in_R \mathbb{Z}_q^*$ (with the MAC actually computed by $\overline{F}$'s MACing oracle in answer to a call on (s, m') with $s = \alpha^{R_i R_j}$ and $m' = (2, j, i, \alpha^{R_j}, \alpha^{R_i})$).

If $\Pi^t_{j,i}$ does not at some later time $\tau_3 > \tau_1$ receive a message of the form (m, a) with $m = (3, i, j, \alpha^{R_i}, \alpha^{R_j})$, then $\overline{F}$ gives up.

However, if $\Pi^t_{j,i}$ is to accept, it must later receive a flow of this form. If $\overline{F}$ has not previously called its MACing oracle on (s, m), then F outputs (s, m, a) as its guess at a valid forgery. If $\overline{F}$ has already made a call on (s, m), then $\overline{F}$ gives up.

Analysis: Suppose E does succeed against responder $\Pi^t_{j,i}$ and against no initiator oracles. In this event, $\overline{F}$ outputs a valid forgery and wins its experiment, provided E or some other oracle has not called $\mathcal{H}_1$ on second input $\alpha^{S_i S_j}$, and provided $\overline{F}$ has not previously calculated the flow that makes $\Pi^t_{j,i}$ accept on behalf of some $\Pi_{i,j}$ or $\Pi_{j,i}$ oracle.

Certainly, by Case 1, the probability that $\mathcal{H}_1$ has been called on second input $\alpha^{S_i S_j}$ is negligible.

Furthermore, the probability that $\overline{F}$ has called its MACing oracle to produce the flow is also negligible. For $\overline{F}$ could only have called on this message on behalf of an initiator $\Pi^s_{i,j}$ which sent α^{R_i} as its own first flow, or on behalf of a responder $\Pi^u_{j,i}$ with $u \neq t$ which also chose α^{R_j} and needs to decide whether or not it should accept. The probability the call was made by an initiator $\Pi^s_{i,j}$ is negligible since such a $\Pi^s_{i,j}$ has accepted, so by assumption there exists $\Pi^v_{j,i}$ which has had a matching conversation to $\Pi^s_{i,j}$. The probability $v \neq t$ is negligible, since then $\Pi^v_{j,i}$ and $\Pi^t_{j,i}$ have independently chosen the same R_j, and $v = t$ is excluded, since then $\Pi^s_{i,j}$ has had a matching conversation to $\Pi^t_{j,i}$. The probability the call was made by $\Pi^u_{j,i}$ is negligible since in this event, $\Pi^u_{j,i}$ and $\Pi^t_{j,i}$ have again independently chosen the same R_j.

We conclude that $\overline{F}$ constructed in this way wins its experiment with probability at least:

$$\frac{n_3(k)}{(T_1(k))^2 T_2(k)} - \lambda(k)$$

for some negligible $\lambda(k)$ — this is still non-negligible, and therefore contradicts the assumed security of the MAC. Thus $n_3(k)$ must be negligible.

Together Cases 2(a) and 2(b) contradict the assumption that $n(k)$ is non-negligible. We conclude that $Pr[\texttt{No-Matching}^E(k)]$ is negligible for all adversaries E.

Condition 4: Fix an arbitrary adversary E and suppose that $advantage^E(k)$ is non-negligible. Thus

$$Pr[E \text{ succeeds}] = \tfrac{1}{2} + n(k)$$

for some non-negligible $n(k)$ by assumption. Now call A_k the event that E picks some $\Pi^s_{i,j}$ oracle to ask its `Test` query such that some $\Pi^t_{j,i}$ oracle has had a matching conversation to $\Pi^s_{i,j}$. Clearly

$$Pr[E \text{ succeeds}] = Pr[E \text{ succeeds}|A_k]Pr[A_k] + Pr[E \text{ succeeds}|\overline{A}_k]Pr[\overline{A}_k] \ .$$

Condition 3 ensures that $Pr[\overline{A}_k] = \lambda(k)$ is negligible. Hence

$$\tfrac{1}{2} + n(k) \leq Pr[E \text{ succeeds}|A_k]Pr[A_k] + \lambda(k) \ .$$

Therefore $Pr[A_k] = 1 - \lambda(k)$ and

$$Pr[E \text{ succeeds}|A_k] = \tfrac{1}{2} + n_1(k)$$

for some non-negligible $n_1(k)$. Now, given event A_k, the key held by $\Pi^s_{i,j}$ will be of the form $\mathcal{H}_2(\alpha^{R_iR_j}, \alpha^{S_iS_j})$ for R_i chosen at random by $\Pi^s_{i,j}$ and R_j chosen at random by $\Pi^t_{j,i}$. Call B_k the event that $\mathcal{H}_2$ has been queried on $(\alpha^{R_iR_j}, \alpha^{S_iS_j})$ by E or some oracle other than $\Pi^s_{i,j}$ or $\Pi^t_{j,i}$. Then

$$Pr[E \text{ succeeds}|A_k] = Pr[E \text{ succeeds}|A_k \wedge B_k]Pr[B_k|A_k] + Pr[E \text{ succeeds}|A_k \wedge \overline{B}_k]Pr[\overline{B}_k|A_k].$$

Since $\mathcal{H}_2$ is a random oracle, and $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ remain unopened by definition, $Pr[E \text{ succeeds}|A_k \wedge \overline{B}_k] = \frac{1}{2}$. Thus

$$\tfrac{1}{2} + n_1(k) \leq Pr[E \text{ succeeds}|A_k \wedge B_k]Pr[B_k|A_k] + \tfrac{1}{2}$$

so that $Pr[B_k|A_k] \geq n_1(k)$. We conclude that given E picks some $\Pi^s_{i,j}$ for which there exists some $\Pi^t_{j,i}$ that has had a matching conversation to $\Pi^s_{i,j}$, then the probability that $\mathcal{H}_2$ has previously been queried on $(\alpha^{R_iR_j}, \alpha^{S_iS_j})$ is non-negligible. In particular, this means that the probability that either $\mathcal{H}_1$ or $\mathcal{H}_2$ has been queried on first input $\alpha^{R_iR_j}$ is non-negligible (specifically at least $n_1(k)$).

Therefore we use E to construct an adversary F of the DHS.
F's operation: F takes as input (p, q, α) generated by $\mathcal{G}_{DH}(1^k)$ and $(\alpha^{R'}, \alpha^{R''})$. F makes (p, q, α) the global parameters for P and picks all entities' secret values at random. F forms the directory *public-info* and starts E.

Now F picks $i, j \in_R I$ and $s, t \in_R \{1, \ldots, T_2(k)\}$, guessing that E will select $\Pi^s_{i,j}$ to ask its `Test` query after $\Pi^t_{j,i}$ has had a matching conversation to $\Pi^s_{i,j}$.

F now answers all $\mathcal{H}_1$ and $\mathcal{H}_2$ oracle queries at random, just like a real random oracle would.

F answers `Corrupt` queries as specified by Π, except that if E asks i or j a `Corrupt` query, F gives up.

F also answers `Reveal` queries as specified by Π, except that if E asks $\Pi^s_{i,j}$ or $\Pi^t_{j,i}$ a `Reveal` query, then F gives up.

Finally, F also answers all `Send` queries as specified by Π, except for `Send` queries to $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$. When E asks $\Pi^s_{i,j}$ its first `Send` query, instead of taking a random sample to form its challenge, $\Pi^s_{i,j}$ instead chooses $\alpha^{R'}$ (so $R_i = R'$). Similarly, F has $\Pi^t_{j,i}$ choose $\alpha^{R''}$ (so $R_j = R''$). Furthermore, if E makes its queries in such a way that $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ have matching conversations, then F must also choose a key κ' to represent $\mathcal{H}_1(\alpha^{R_i R_j}, \alpha^{S_i S_j})$, and use this key when deciding how these oracle should respond to `Send` queries.

If E does not make its queries in such a way that $\Pi^t_{j,i}$ has a matching conversation to $\Pi^s_{i,j}$, then F gives up. On the other hand, if E does make its queries in this way, then $\Pi^s_{i,j}$ will accept (holding the key $\mathcal{H}_2(\alpha^{R_i R_j}, \alpha^{S_i S_j})$, although of course F doesn't know $\alpha^{R_i R_j}$ and so can't actually compute this key).

Now let $T_3(k)$ denote a polynomial bound on the number of $\mathcal{H}_1$ and $\mathcal{H}_2$ queries on distinct first inputs made by E and its oracles. F picks $l \in_R \{1, \ldots, T_3(k)\}$, guessing that the lth distinct first input on which $\mathcal{H}_1$ or $\mathcal{H}_2$ is called during the experiment will be on $\alpha^{R_i R_j}$. When the lth distinct first input is called is made (say on g), then F stops and outputs g as its guess at $\alpha^{R_i R_j}$.

If E and its oracles do not make l distinct first input calls to $\mathcal{H}_1$ and $\mathcal{H}_2$ before E asks its `Test` query, then F gives up.

One problem remains. $\mathcal{H}_1$ may have been called on $(\alpha^{R_i R_j}, \alpha^{S_i S_j})$ before $\mathcal{H}_1$ or $\mathcal{H}_2$ is called on the lth distinct first input. In this case, F will have picked an answer at random, and its answer may have been in contradiction to the keys that it has used to represent this call. The problem is that E is not guaranteed to halt in this eventuality. To sidestep this potential problem, let $T_4(k)$ denote a polynomial bound on E's runtime under ordinary circumstances. If F runs E for longer than $T_4(k)$, F gives up, concluding that it must have missed an $\mathcal{H}_1$ query on first input $\alpha^{R_i R_j}$.

Analysis: Suppose E does pick $\Pi^s_{i,j}$ to ask its `Test` query after $\Pi^t_{j,i}$ has had a matching conversation to $\Pi^s_{i,j}$. Then, as we have seen, with non-negligible probability, E or some other oracle has called $\mathcal{H}_1$ or $\mathcal{H}_2$ on first input $\alpha^{R_i R_j}$. Hence the probability that F succeeds is at least:

$$\frac{n_1(k)}{(T_1(k))^2 (T_2(k))^2 T_3(k)} - \mu(k)$$

for some non-negligible $\mu(k)$ — this is still non-negligible, and therefore contradicts the assumed security of the DHS. We conclude that $n_1(k)$ must be negligible, and thus that $advantage^E(k)$ must be negligible. □

C Proof of Theorem 11

This theorem is only a stepping-stone to Theorem 8, and the ideas used in the proof have already been seen in the previous proofs, so we just give a sketch of the construction here.

Proof.(sketch)

The first two conditions of Definition 10 follow immediately from the description of P and the assumption that $\mathcal{H}$ is a random oracle.

Consider the third condition. Fix an arbitrary adversary E that makes no `Reveal` queries, and suppose that $advantage^E(k)$ is non-negligible. E picks some $\Pi^s_{i,j}$ oracle to ask its `Test` query. The key held by such an oracle will be of the form $\mathcal{H}(\alpha^{R_i R_j}, \alpha^{S_i S_j})$. Since i and j are by definition uncorrupted, and by assumption E does not ask any `Reveal` queries, if E is to succeed with non-negligible probability, then it must itself have queried $\mathcal{H}$ on second input $\alpha^{S_i S_j}$ at some time. Therefore such an adversary can be used to construct an adversary of the DHS which succeeds with non-negligible probability, in much the same way as such an adversary was constructed in Case 1 of the proof of Theorem 9. The existence of such an adversary contradicts the assumed security of the DHS, so we conclude that $advantage^E(k)$ must be negligible for all adversaries E that make no `Reveal` queries. □